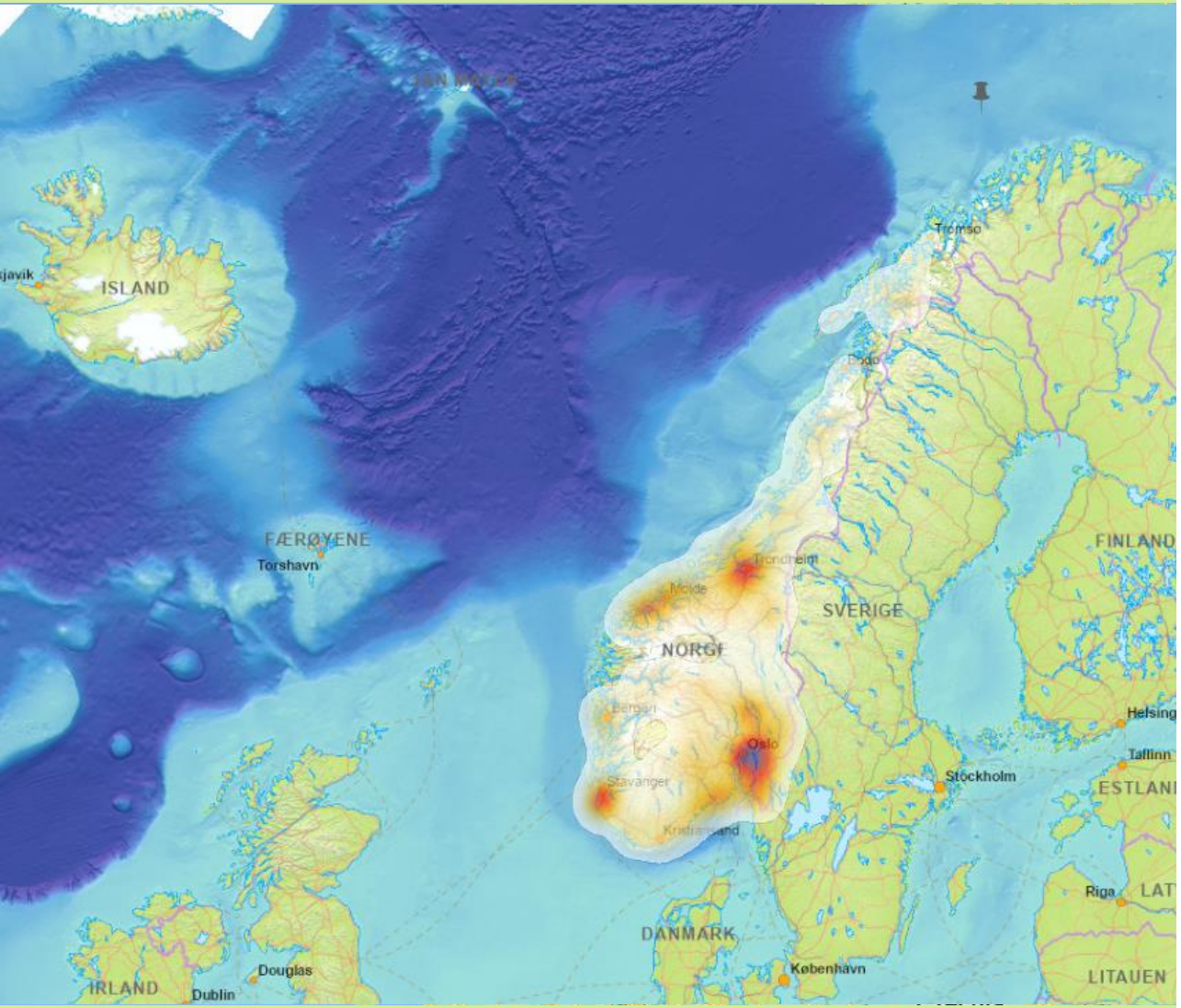
eika.

Implementing ML Ops in insurance

A case study using a complex multi-model Customer Lifetime Value system

Sindre Henriksen, Eika Forsikring
Øyvind Klåpbakken, Eika Forsikring
Fredrik Wollert Hansen, Eika Forsikring

Prompt: Machine learning engineers working on an important, revolutionary problem, dark cyberpunk style



Eika Forsikring

- Established in 1999, based in Hamar
- Bancassurance with relatively rural customer base
- Gross Written Premium ≈€331m (3.9b NOK)
- Small analytics team:
  - 3 data scientists
  - 2 pricing actuaries
  - 8 analysts/engineers/architects

What is ML Ops?

- Principles, processes, technologies for operationalising ML
- To ML what DevOps is to development
- What works for 2 models does not work for 20
- Key objectives:
  - Increased development speed
  - Reduced errors in development and production
  - Faster time to market
  - Faster model updates and automatic retraining
- Monitoring and observability

- 6 pure premium models (autocalibrated boosted decision trees*; 45% of GWP)
- Churn models (logistic regression)
- Various other model components (CPI forecasts, time value of money, administration costs, …)

Customer Lifetime Value

* Denuit, Charpentier, & Trufin (2021). Autocalibration and Tweedie-dominance for insurance pricing with machine learning.
Ciatto et al. (2022). Does autocalibration improve goodness of lift?
Hainaut, Trufin, & Denuit (2022). Response versus gradient boosting trees, GLMs and neural networks under Tweedie loss and log-link.
Wüthrich, M. (2023). Model selection with Gini indices under auto-calibration.

Accelerating Machine Learning

(while making fewer errors)

Feature store

- Most ML projects start with feature engineering
- Feature stores standardise features, make them reusable, and enforce production quality code
- A single interface for retrieving features irrespective of whether you are in a development or production environment

eika.

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts *

Eika's feature store

*Emily Riederer. https://emilyriederer.netlify.app/post/column-name-contracts/

eika.

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts *

AMT-PersOrg- Mean_claim_cost- M05- 0_to_6_months

An **amount**, as opposed to an integer/count (**N**_), or categorical variable (**CAT**_)

*Emily Riederer. https://emilyriederer.netlify.app/post/column-name-contracts/

Eika's feature store

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts *

AMT- **PersOrg-**Mean_claim_cost- M05- 0_to_6_months

Entity is a **person** or **organisation** (as opposed to an agreement, object, or other entity)

*Emily Riederer. https://emilyriederer.netlify.app/post/column-name-contracts/

Eika's feature store

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts *

AMT- PersOrg- **Mean_claim_cost** -M05- 0_to_6_months

The metric being used (in this case the mean claim cost)

Eika's feature store

*Emily Riederer. https://emilyriederer.netlify.app/post/column-name-contracts/

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts *

AMT- PersOrg- Mean_claim_cost- **M05-** 0_to_6_months

The product code for which this feature applies (M05 = car insurance)

Eika's feature store

*Emily Riederer. https://emilyriederer.netlify.app/post/column-name-contracts/

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts *

AMT- PersOrg- Mean_claim_cost- M05- **0_to_6_months**

The time period for which the aggregation happens (in this case we take the mean over the last 6 months)

Eika's feature store

*Emily Riederer. https://emilyriederer.netlify.app/post/column-name-contracts/

eika.

Eika's feature store

- Batch-only
- Stored in an Azure SQL database
- Column names as contracts

AMT- PersOrg- Mean_claim_cost- M05- 0_to_6_months

The time period for which the aggregation happens (in this case we take the mean over the last 6 months)

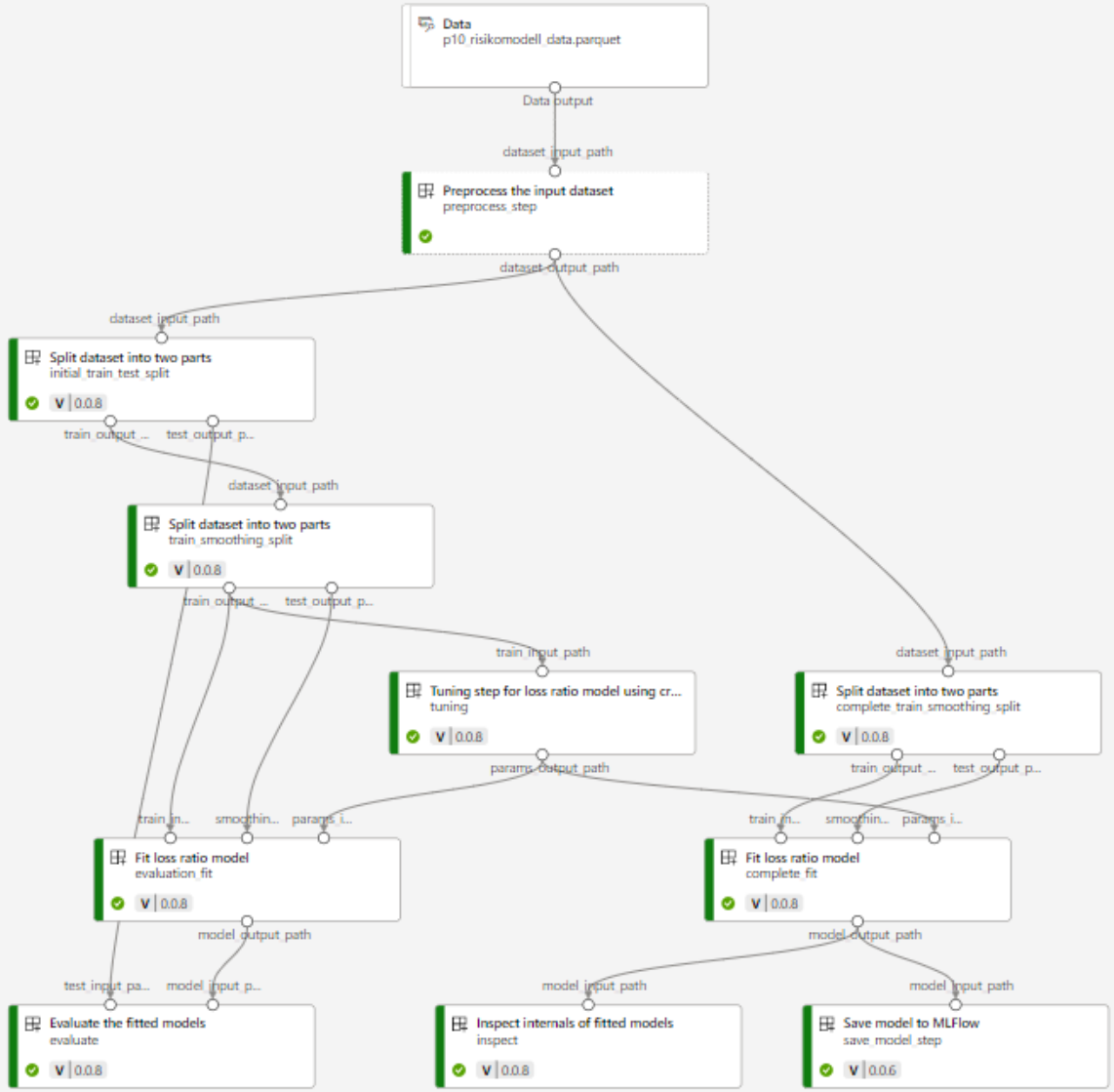
Custom Python library for interacting with the store

```
from effstools.feature_store import FeatureStore
feature_store = FeatureStore()

feature_store.select_columns(
  feature_store.get_level(2) == "Mean_claim_cost"
)

feature_store.to_sql(...)
```

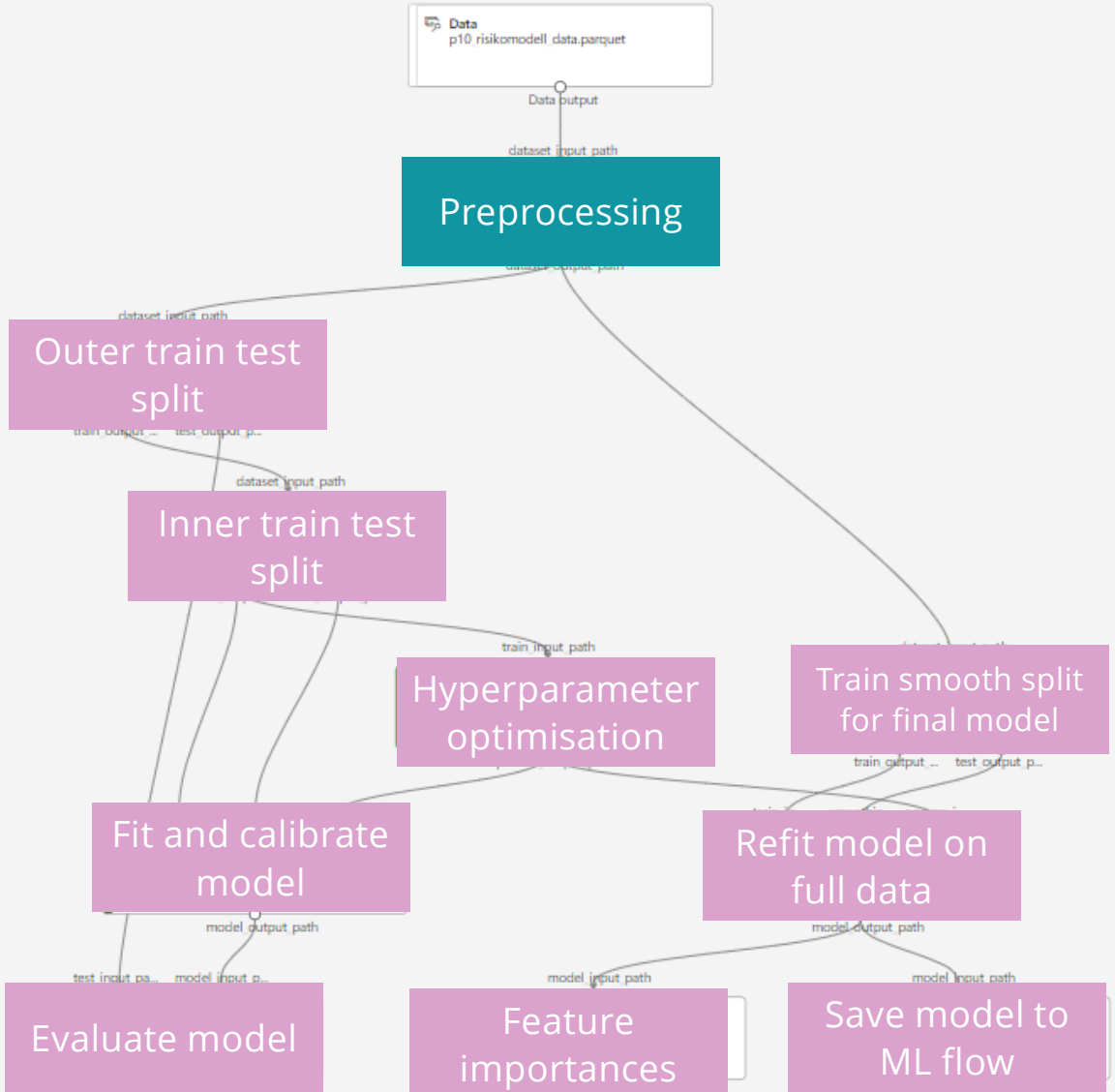
- Big mindset change: tailoring → manufacturing



Building a model factory

using reusable components

- Big mindset change: tailoring → manufacturing



Building a model factory

using reusable components

eika.

- Big mindset change: tailoring → manufacturing
- Reusable components in AzureML

```yaml
$schema: https://azuremlschemas.azureedge.net/latest/commandComponent.schema.json
type: command

name: fit_loss_ratio_model
display_name: Fit loss ratio model
description: Fit loss ratio model and calibration model on training set and smoothing set respectively, u
sing the supplied parameters
version: 0.0.8
inputs:
  train_input_path:
    type: uri_file
  smoothing_input_path:
    type: uri_file
  params_input_path:
    type: uri_file
outputs:
  model_output_path:
    type: uri_file
code: src/fit_loss_ratio_model
environment: azureml:risikomodell_r_env@latest
command: >-
  Rscript fit_loss_ratio_model.R
  --train_input_path ${{inputs.train_input_path}}
  --smoothing_input_path ${{inputs.smoothing_input_path}}
  --params_input_path ${{inputs.params_input_path}}
  --model_output_path ${{outputs.model_output_path}}
```

```python
group_split = ml_client.components.get(name="group_split", version="0.0.2")
tune_loss_ratio_model = ml_client.components.get(name="tune_loss_ratio_model", version="0.0.6")
fit_loss_ratio_model = ml_client.components.get(name="fit_loss_ratio_model", version="0.0.8")
evaluate_loss_ratio_model = ml_client.components.get(name="evaluate_loss_ratio_model", version="0.0.6")
inspect_loss_ratio_model = ml_client.components.get(name="inspect_loss_ratio_model", version="0.0.7")
save_model = ml_client.components.get(name="save_model", version="0.0.3")
```

Building a model factory

using reusable components

If you learned nothing else…

- A feature store dramatically increases development speed and reduces errors in development and production
- If you can, choose a commercial ML platform + feature store and use a thought through framework for structuring your features
- Consider approaching machine learning as a process of manufacturing: build models using quality-controlled, production-grade reusable components

Data scientists

Øyvind Klåpbakken

Sindre Henriksen

Fredrik Wollert Hansen

Actuaries

Maja Bratlien Larsen

Kim André Arntsen