VLAIO TETRA

Machine Learning Operations for Edge Condition Monitoring

(MLOps4ECM)

Tussentijdse vergadering 14/11/2024

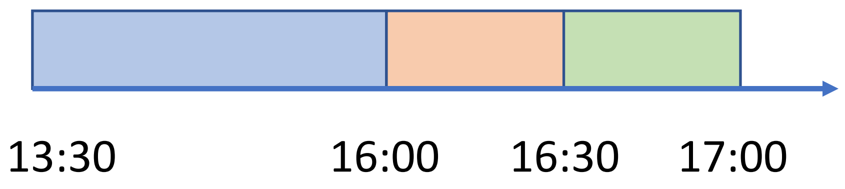
Locatie: Marelec Food Technologies

Met steun van

HBC.2023.0062

Agenda

- Introductie
- Monitoring & drift detection
- Data management within the MLOps cycle
- Auto deployment & energy monitoring
- Volgende stappen
- Rondleiding Marelec Food Technologies
- Receptie & netwerken



Update MLOps Team

- KU Leuven
  - Mathias Verbeke
  - Lara Luys

- VIVES
  - Jonas Lannoo
  - Alexander D’hoore
  - Sille Van Landschoot
  - Noah Debaere

Update begeleidingsgroep

- **Araani**
- Beckhoff
- Bekaert*
- CNHi
- CTRL Engineering
- Leap Technologies
- LET
- Marelec
- Superlinear (Radix)
- Siemens
- Summa nv
- Televic
- Vandewiele
- Vintecc
- Yazzoom
- Odisee (waarnemer)
- Thomas More (waarnemer)
- Howest (waarnemer)

***TODO: Reglement van Orde (RvO)**

Feedback vorige bevraging

Responsgraad: 72%

Is het project nog relevant voor de onderneming/organisatie:	4.43/5
Projectverloop en voorlopige resultaten voldoende:	4.21/5
Tevredenheid over ruimte voor overleg en sturing in project:	4.36/5
Tevredenheid van de behandelde punten in de vergadering:	4.43/5
Verwachting van de toepassing van de resultaten in bedrijf:	3.64/5

Feedback: positief (“Benieuwd naar het vervolg!”, “Uitermate interessant”)

Feedback vorige bevraging

Suggestie:

- Video’s gebruiken om tools te demonstreren

Populariteit hardware platformen:

1. NVIDIA Jetson
2. Beckhoff PLC & Raspberry Pi
3. “Andere” (bv. TDA4VM)
4. ARM Cortex-A & STM32

Feedback vorige bevraging

Workshop suggesties allemaal populair!

Seminarie evenement met sprekers uit industrie:

- Sprekers melden zich aan
- Content suggesties:
  - Praktische voorbeelden, concrete toepassingen van MLOps
  - Real-time integratie van ML in een machine
  - Pipeline in productie

Monitoring & drift detection

Lara Luys

Monitoring on the edge

- Started from use case: Vandewiele & Leap technologies
- ML model = Fault detection of a machine
- Trained with certain machine settings
- Model performs worse with other machine settings

How do you start setting up monitoring?

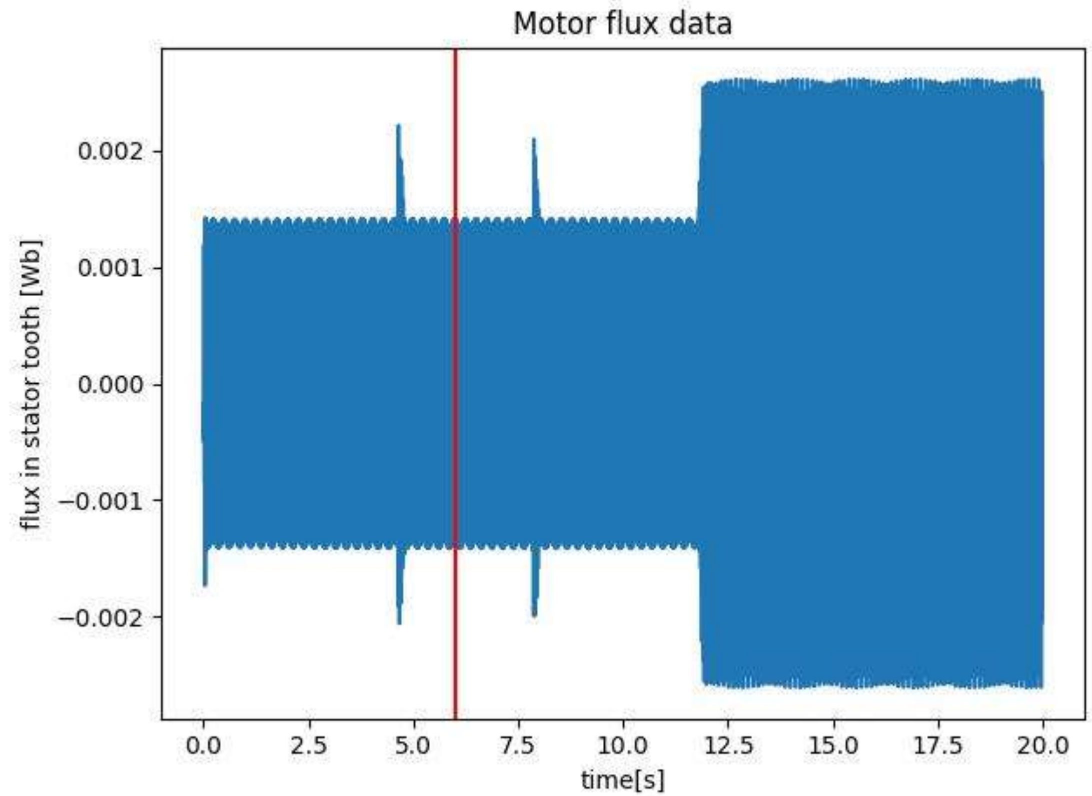
- **Explore data and model**
- Choose drift metric
- To Evidently or not to Evidently
- When to detect drift?
- How to retrain?

Explore data and model

- What (type of) data was used for training?
- Which machine conditions/metadata is known?
- Which type of model is used?
  - XGBoost
  - Neural Network
  - Support vector machine
  - …

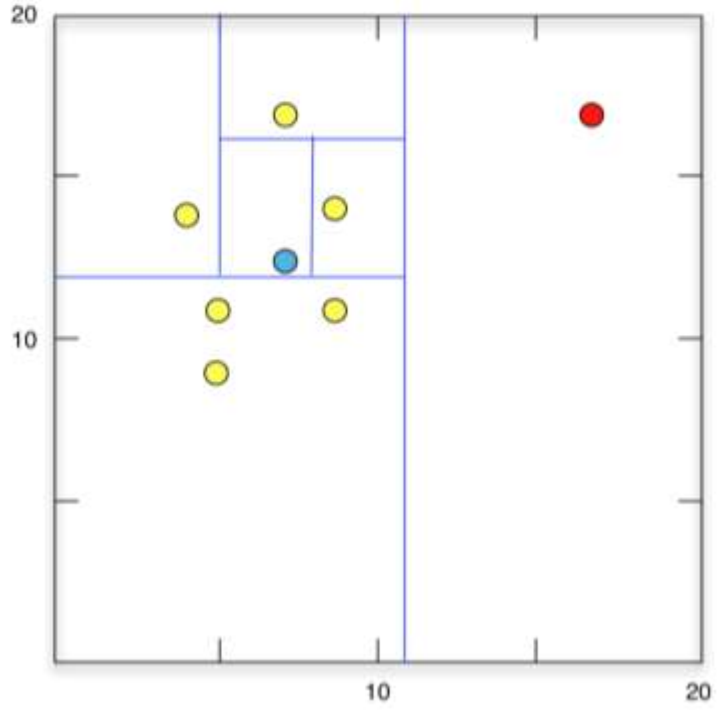
Dataset example

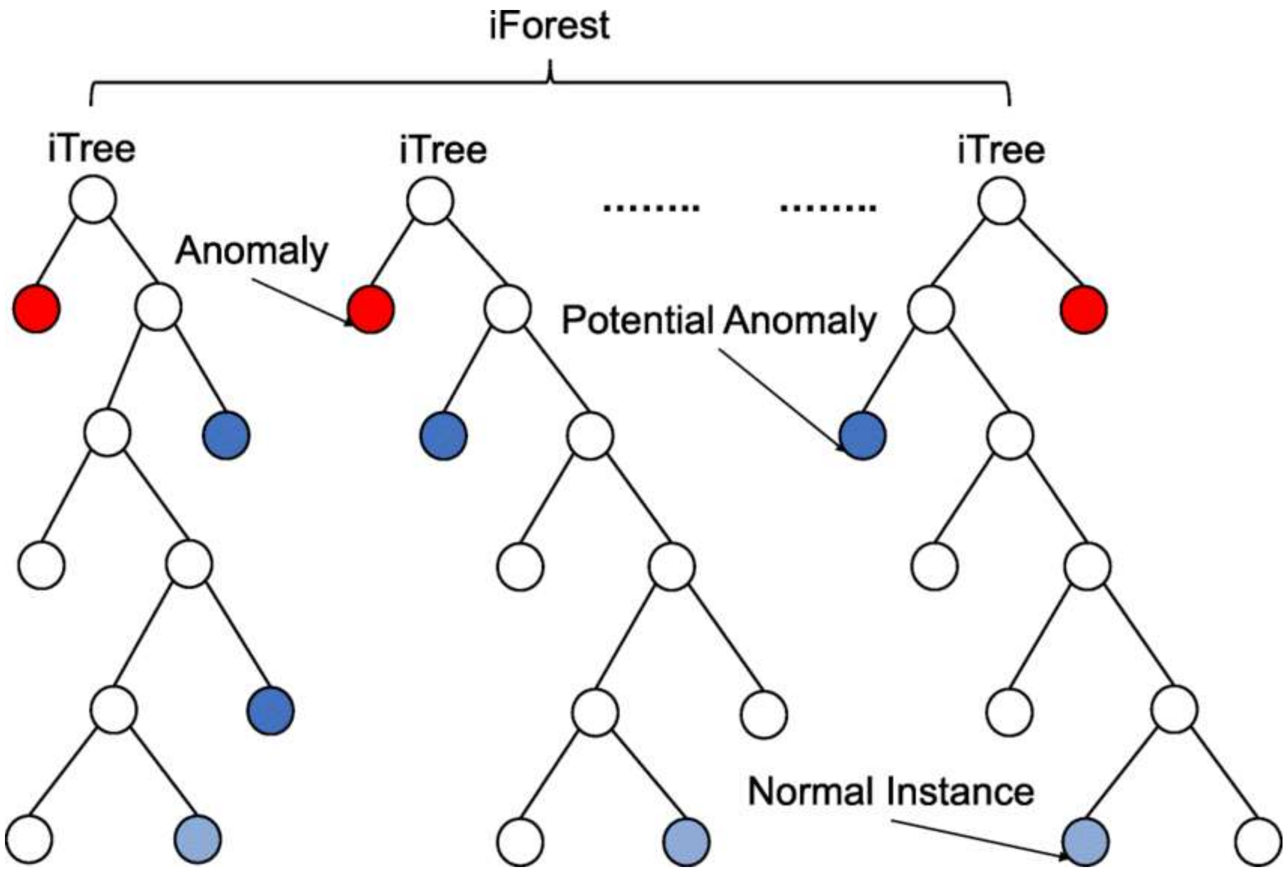
- Example: Flux measured in a stator tooth of electric motor
- Type of problem: Anomaly detection
- Trained model with voltage level 164V
- However, voltage level changes to 289V



Isolation forest

- Create decision trees
- Try to separate data from each other

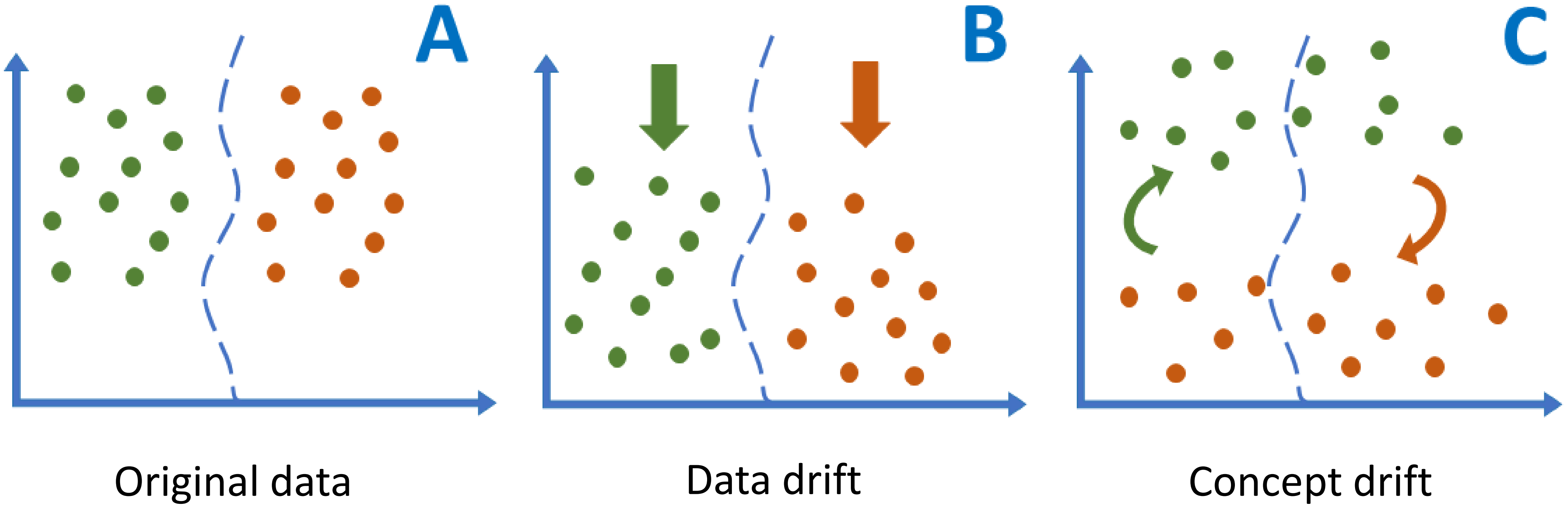




How do you start setting up monitoring?

- Explore data and model
- **Choose drift metric**
- To Evidently or not to Evidently
- When to detect drift?
- How to retrain?

Goal is drift detection



Different type of drift metrics

- Drift can be detected in several types of ways
- Common = distribution drift metrics
  - Compare the current distribution to a reference dataset distribution (e.g. training dataset)
  - Several different distribution metrics available
- Which metric you choose for comparison = dependent on data

Categorical data vs. Numerical data

- Categorical:
  - data is split up into several categories
  - e.g. our motor setup outputs: Anomaly or Not
- Numerical data:
  - data can be any value:
  - e.g. motor inputs: flux data
- Some metrics only work for one type of data, other metrics work for both

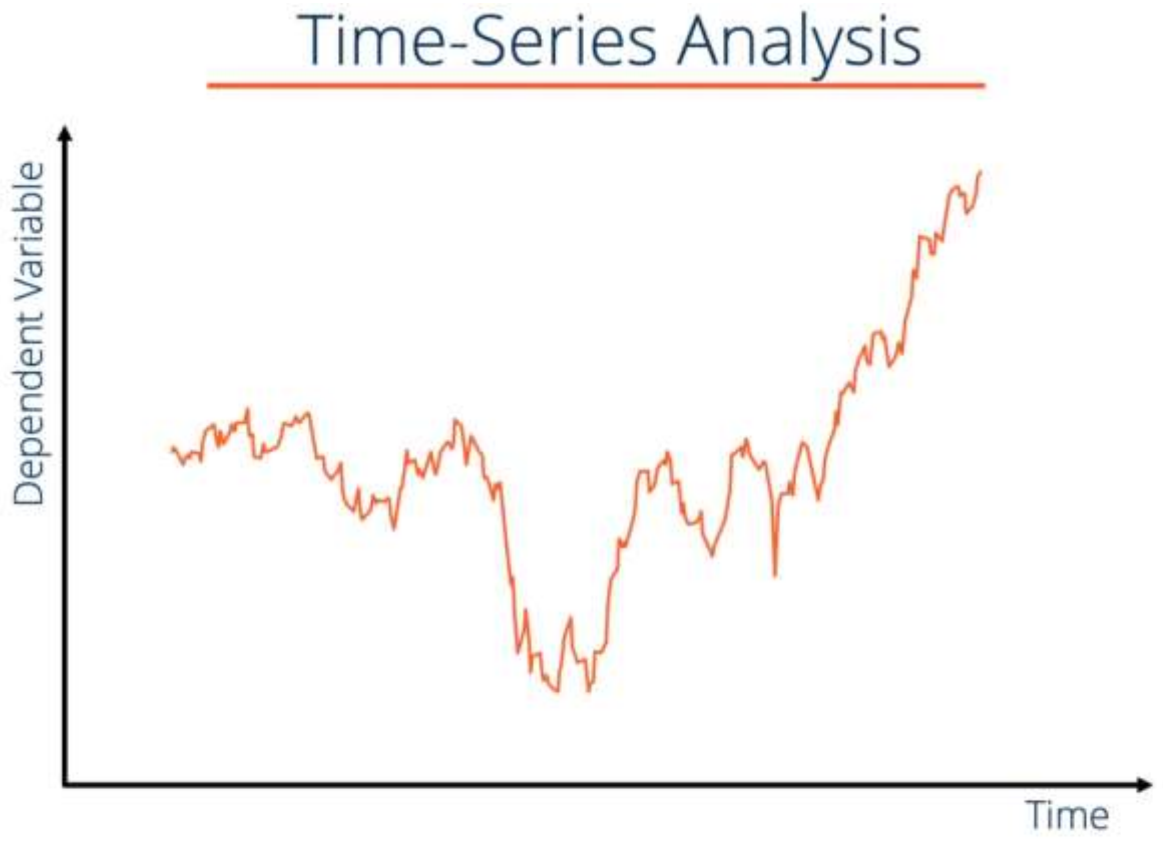
Categorical drift metrics

- Chi-squared
- Z-test
- Fisher’s Exact test
- G-test
- Total-Variation-Distance

Machine name	Machine location	User
Machine 1	Factory 1	Person 1
Machine 1	Factory 2	Person 2
Machine 2	Factory 1	Person 4
Machine 3	Factory 1	Person 1
Machine 2	Factory 2	Person 3
Machine 1	Factory 1	Person 2
Machine 3	Factory 2	Person 4
Machine 2	Factory 2	Person 3

Numerical drift metrics

- Kolmogorov-Smirnov test
- Wasserstein distance
- Anderson-Darling test
- Cramer-Von-Mises test
- Mann-Whitney U-rank test
- Energy distance
- Epps-Singleton test
- T-test
- Empirical-MMD

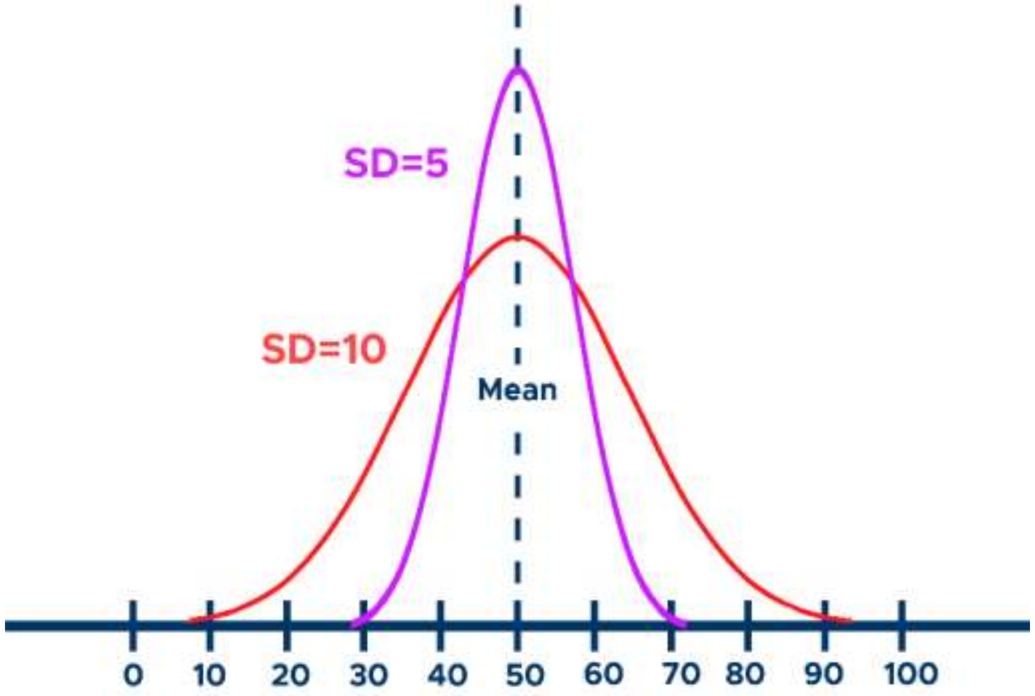


Metrics for both

- Kullback-Leibler divergence
- Population Stability Index
- Jensen-Shannon distance
- Hellinger distance

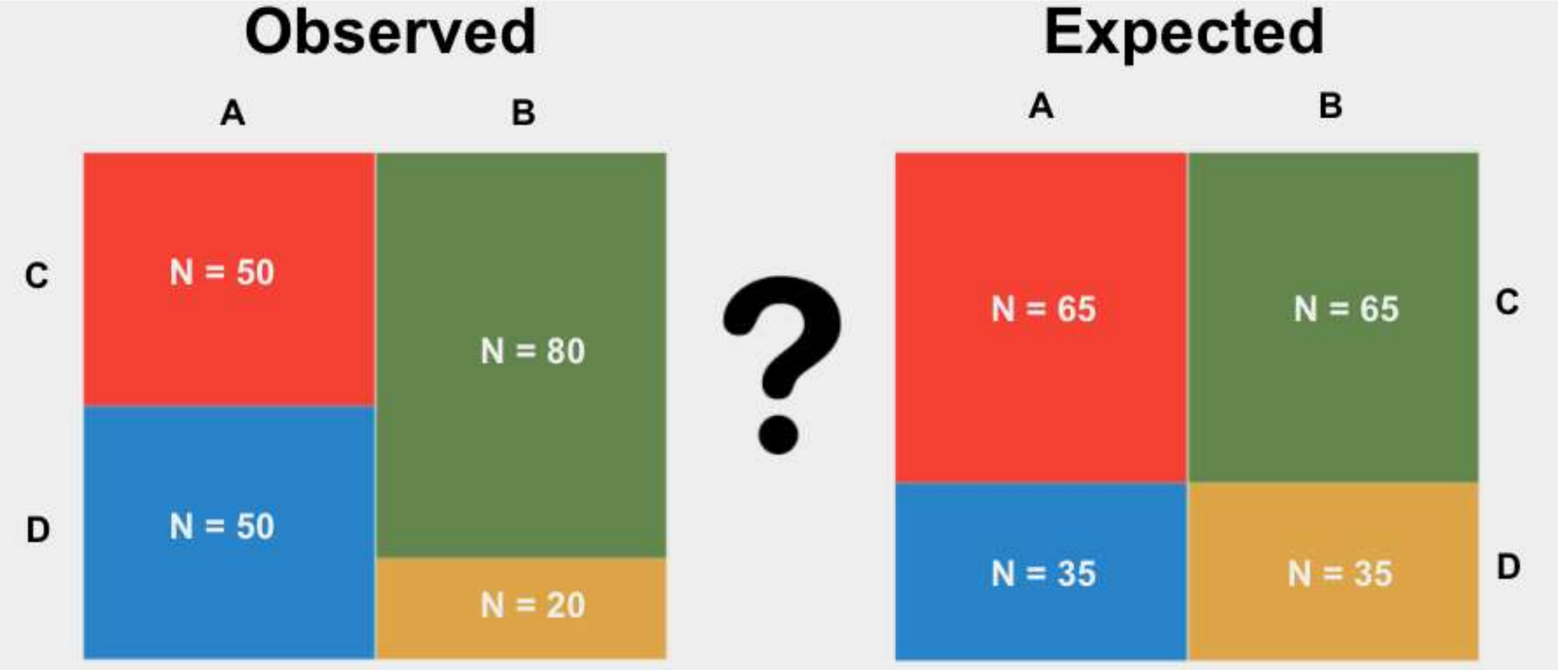
Z-test (Categorical)

- Compare two distributions using their **means** and **standard** deviations.
- Formula: Z = $\frac{(\bar{X}_1 - \bar{X}_2)}{\sqrt{\sigma^2_{X_1} + \sigma^2_{X_2}}}$
- if Z > 2.33 →distributions are different
- Good for binary class data



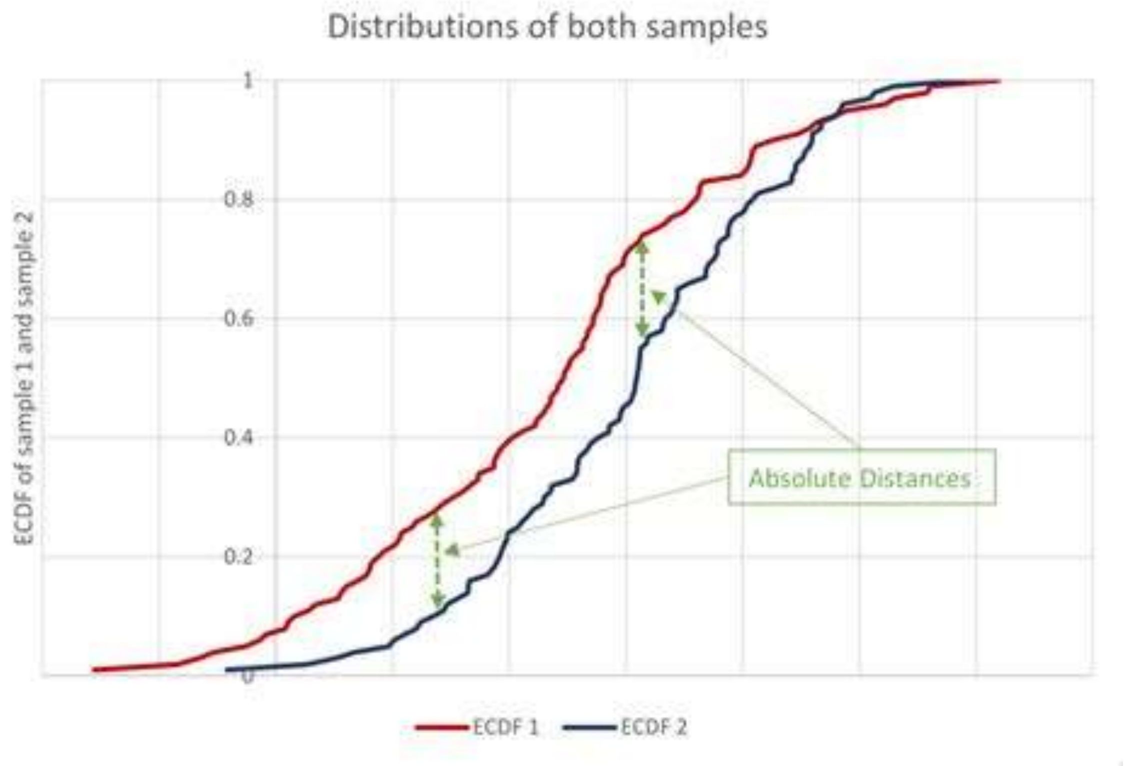
Chi-squared (Categorical)

- Compare the observed values of each class (current data) to their expected values (reference data).
- Good for multi-class data



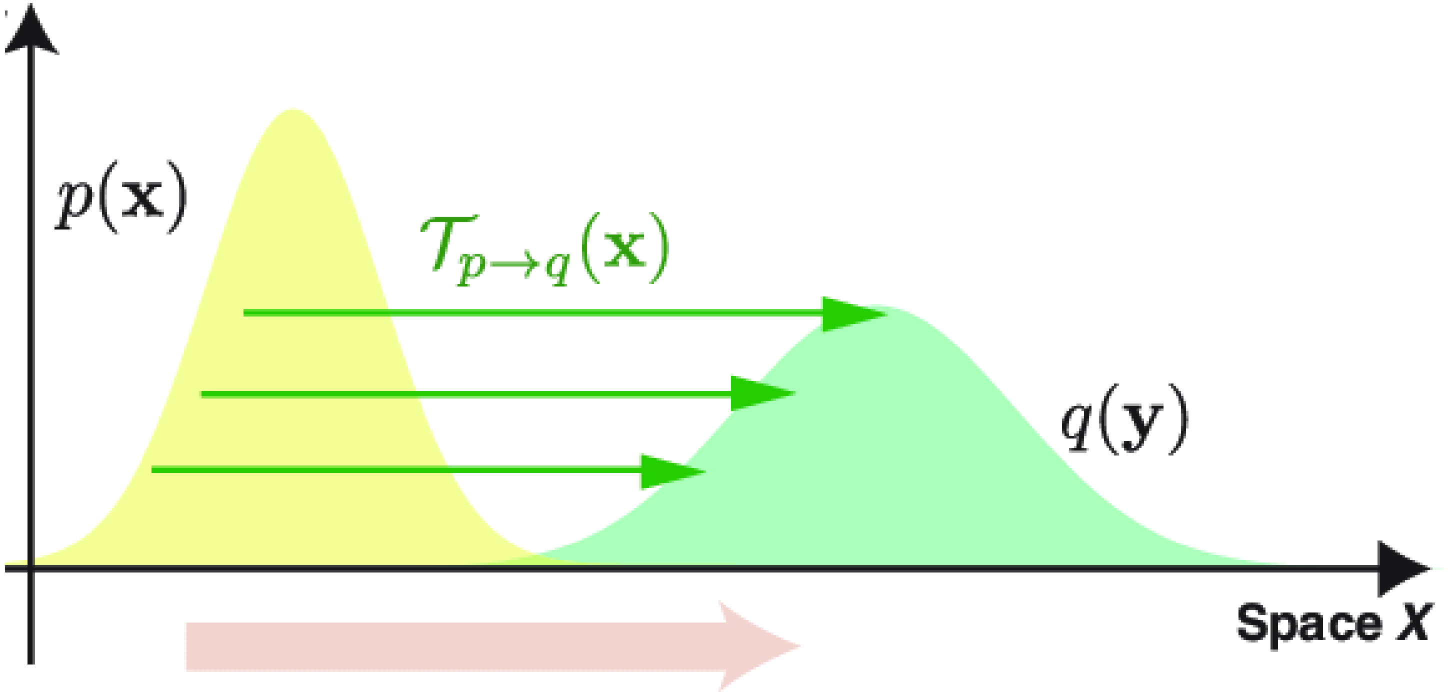
Kolmogorov-Smirnov test (Numerical)

- The biggest distance between accumulative distributions
- Bigger distance = bigger drift



Wasserstein distance (Numerical)

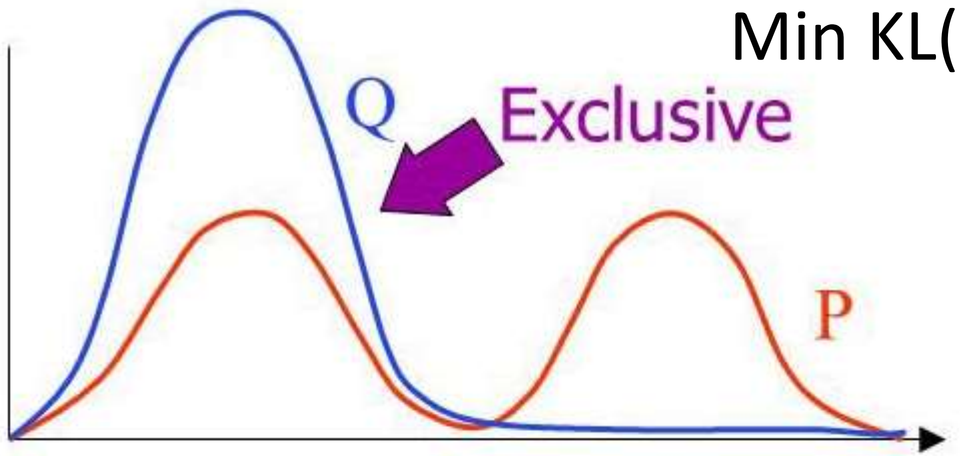
- How much effort does it take to change one distribution into the other



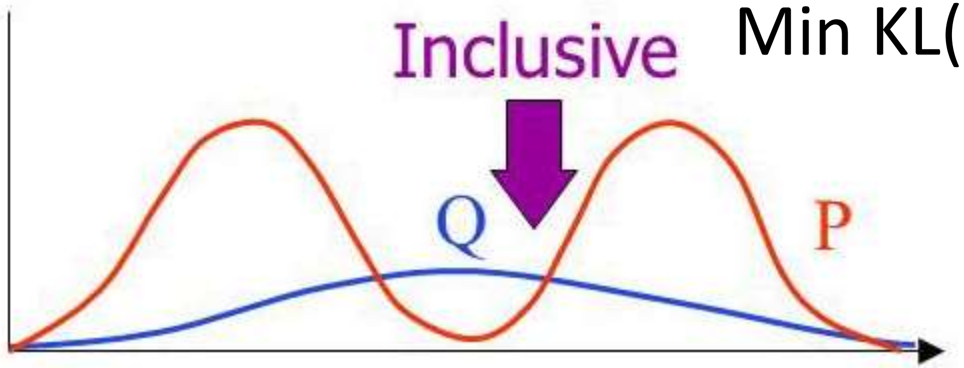
Kullback-Liebler , Population stability index, and Jensen-Shannon (Both)

- KL-divergence: Compares relative entropy (= difference) between two distributions
- Is not symmetrical : P,Q ≠ Q,P
- 2 solutions:
  - **PSI**: Cannot deal with empty bins
  - **Jensen-Shannon**: Can deal with empty bins

Min KL(Q||P)



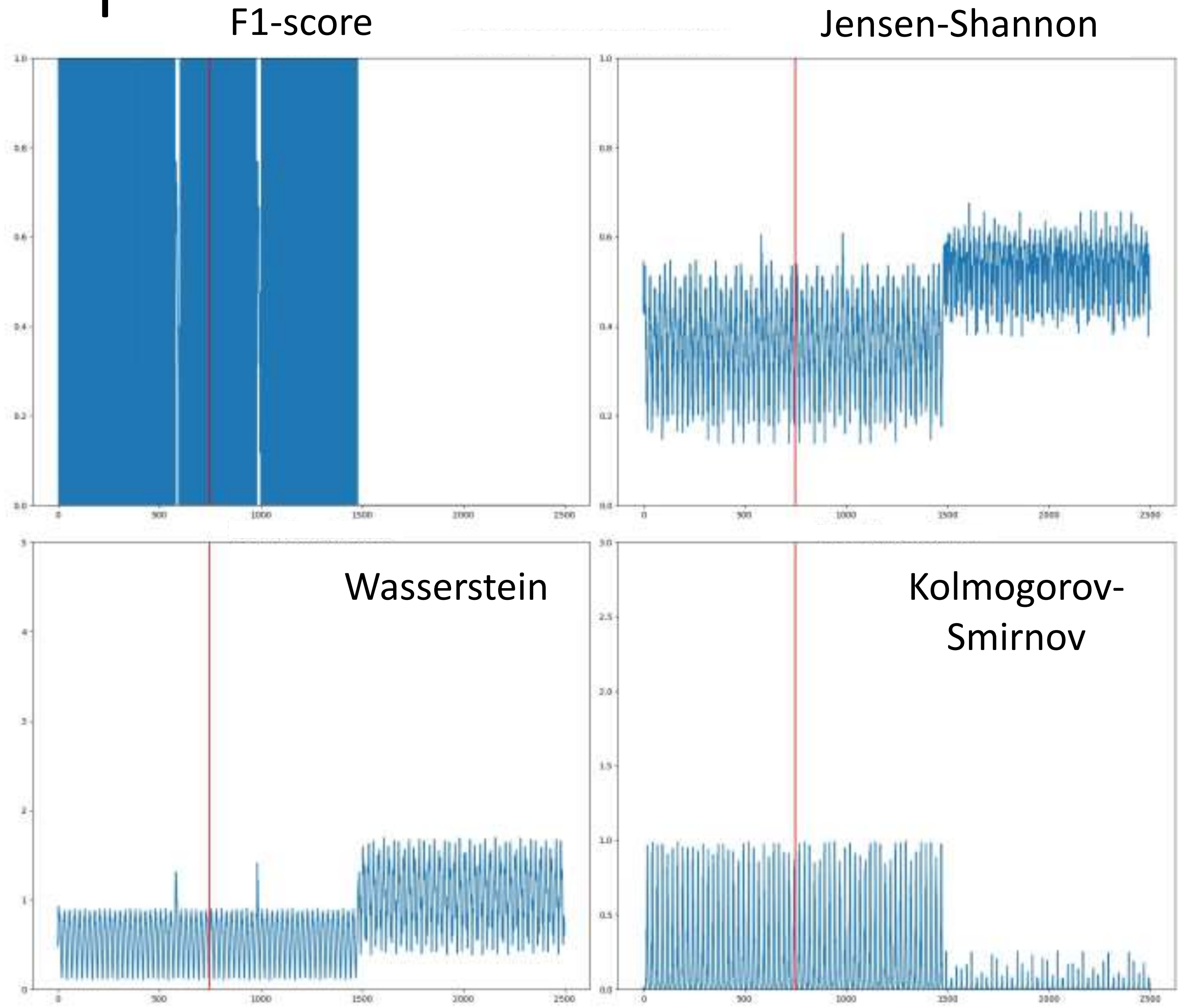
Min KL(P||Q)



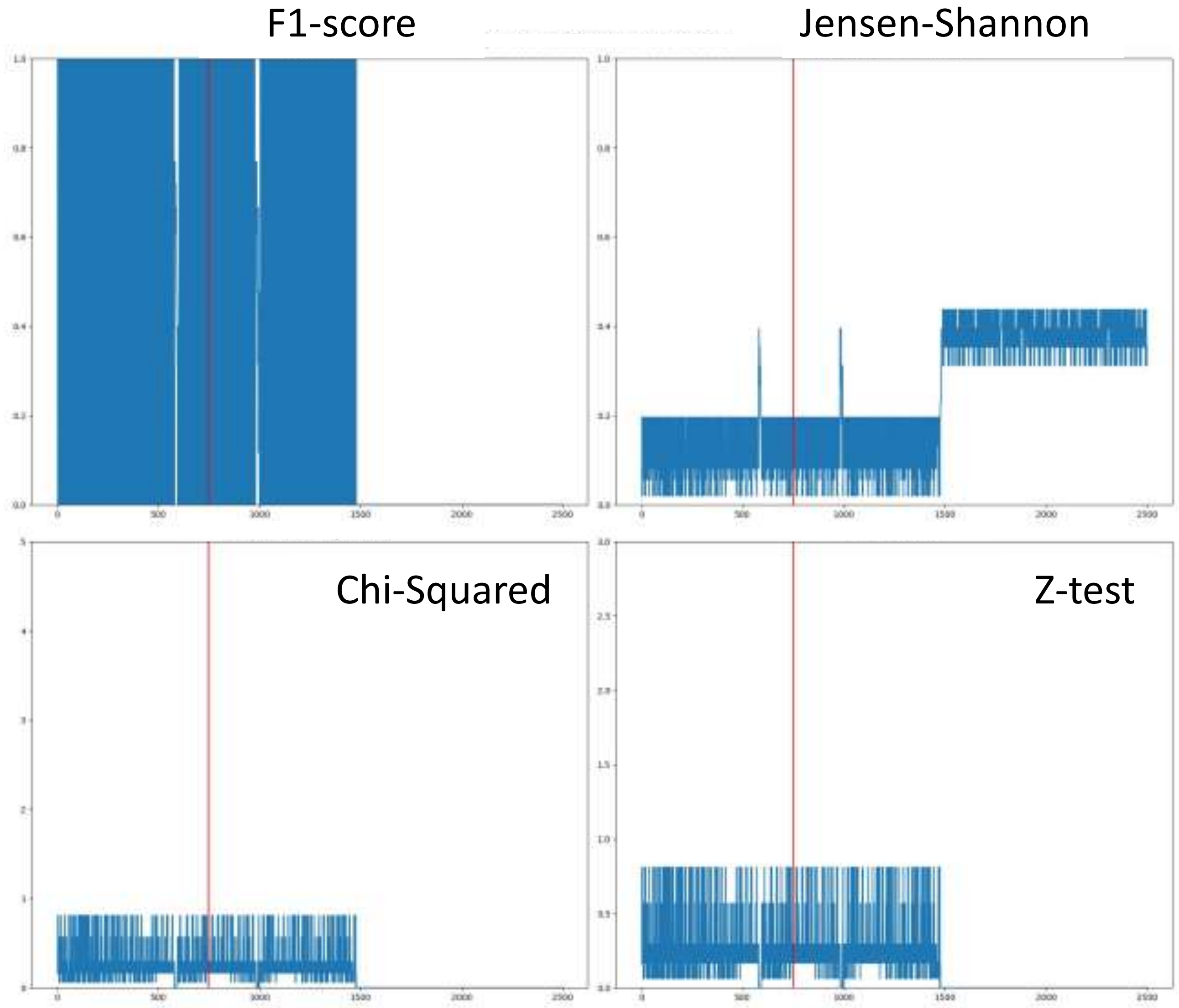
How to choose a metric?

- Do you have normal and drifted data available?
  - Yes? Test different metrics → Which creates better drift detection
  - No? Depending on data size and type data different defaults
- Categorical
  - Data <= 1000 objects: Z-test for binary data & Chi-square test otherwise
  - Data > 1000 objects: Jensen-Shannon
- Numerical
  - Data <= 1000 objects: Kolmogorov-Smirnov test
  - Data > 1000 objects: Wasserstein distance

Motor example: Data drift



Motor example: Prediction drift



How do you start setting up monitoring?

- Explore data and model
- Choose drift metric
- **To Evidently or not to Evidently**
- When to detect drift?
- How to retrain?

Evidently

- Open-source ML observability platform
- Python library → creates (pre-made) test suits and reports
- Able to save as html, json, dictionary → fully offline possible
- Only tabular and text data → working on other unstructured data

Evidently Pro’s

- Easy to create thorough, customizable reports and test suites
- Can be used offline (not possible for most other monitoring tools)
- Can be integrated with other visualisation tools if necessary
- Automatically creates distributions from data for tests

Evidently Con’s

- Depending on number of tests & metrics → can be a lot slower than custom implementation in python
- Size of reference data can make this even slower
- Sometimes limitations in implementation

When to use Evidently (or other tools)

- When you want more accurate monitoring metric
- When you want clear reports and test suites without a lot of work
- When time does not matter too much

- E.g. Use self-implemented metric for real-time usage and Evidently for every second (or longer)

How do you start setting up monitoring?

- Explore data and model
- Choose drift metric
- To Evidently or not to Evidently
- **When to detect drift?**
- How to retrain?

When to detect drift?

- Set threshold for drift detection
  - Use data and predictions if possible
  - Use different metrics if it helps
  - Only detect drift if all thresholds are exceeded
- Watch out for anomalies → This is not drift
  - Thresholds should be exceeded N times

How do you start setting up monitoring?

- Explore data and model
- Choose drift metric
- To Evidently or not to Evidently
- When to detect drift?
- **How to retrain?**

How to retrain?

- Retraining on edge or on cloud?
  - Type model (neural network, K-means, …)
  - Size edge device (memory, CPU power)
- Data collection?
  - Also takes resources
  - More data = better retraining
  - Different strategies
    - Always keep a window of data
    - Start collecting when first threshold exceeded, send data when both are exceeded
    - …

Vandewiele – Leap Technologies

- Example:
  - 1 drift at a time
  - Change in data & prediction means model is worse
- Real life :
  - Multiple drifts at the same time
  - Not all changes in data means a worse model
  - Not all changes in predictions means a worse model
- Next steps:
  - Use a different technique on detecting drift finding a "true" label instead.
  - Focus on 1 bigger change

Televic: Data Management within the MLOps Cycle

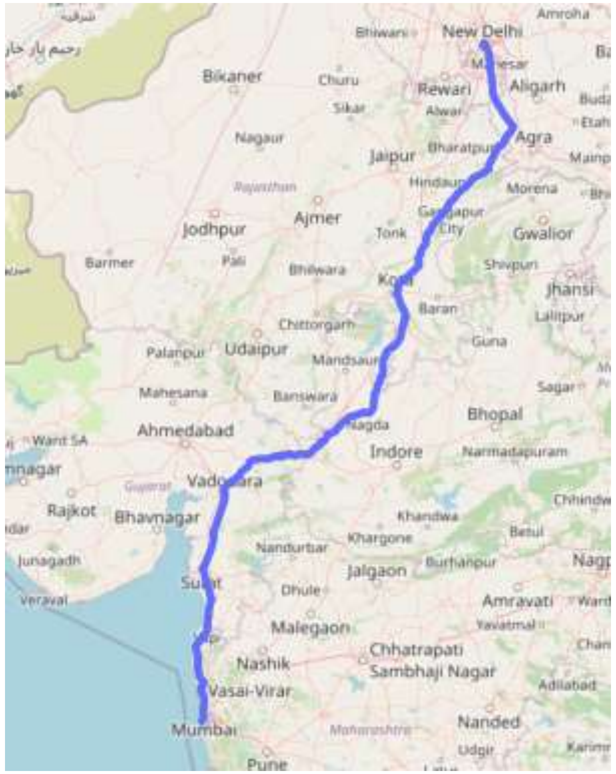
Alexander D’hoore

Introduction: Televic Dataset

- Received a **vibration sensor** dataset from Televic Rail
  - Train journeys between New Delhi and Mumbai
  - Includes GPS location, datetime, temperature, etc

- **Dataset size:**
  - 16 million rows for GNSS data at 1 Hz
  - 325 million rows for IMU data at 20 Hz

- **Time span:** Nearly a full year



shape: (16_162_030, 19)

gnss_lat	gnss_lon	gnss_heading	gnss_speed	dr_lat	dr_lon	dr_heading	dr_speed	temperature
f64	f64	f64	f64	f64	f64	f64	f64	f64
21.453674	72.950279	22.429083	35.872547	21.454035	72.950439	22.532396	36.0	65.0
21.453974	72.950417	22.365637	35.91634	21.454334	72.950577	22.55385	36.0	65.0
21.454275	72.950546	22.267731	35.825092	21.454634	72.950706	22.519489	36.0	65.0
21.454575	72.950676	22.285122	35.81258	21.454935	72.950836	22.509026	35.0	65.0
21.454872	72.950813	22.229118	35.817322	21.455231	72.950966	22.477715	35.0	65.0
…	…	…	…	…	…	…	…	…
26.021254	76.360023	47.898663	7.812088	26.021282	76.360062	55.534973	8.0	57.0
26.021305	76.360077	46.056339	7.968601	26.02133	76.360123	56.72504	7.0	57.0
26.021358	76.360138	45.945461	7.977659	26.021383	76.360176	57.779652	7.0	57.0
26.021408	76.360191	44.731991	8.112914	26.021427	76.360214	58.659435	7.0	57.0
26.02146	76.360252	44.87648	8.111129	26.021479	76.360275	59.045689	8.0	57.0

shape: (325_513_093, 7)

accel_x	accel_y	accel_z	gyro_roll	gyro_yaw	gyro_pitch	time_ts
f64	f64	f64	f64	f64	f64	datetime[ns]
370.642857	4.035714	12304.428571	58.785714	-269.928571	27.25	2019-01-31 14:48:35.800
72.122449	179.163265	12573.693878	180.612245	-266.040816	34.653061	2019-01-31 14:48:35.850
125.372549	12.764706	13015.72549	247.960784	-269.372549	50.764706	2019-01-31 14:48:35.900
187.408163	-237.693878	13091.408163	298.959184	-263.918367	89.244898	2019-01-31 14:48:35.950
183.0	-661.058824	13394.627451	231.45098	-255.568627	102.509804	2019-01-31 14:48:36
…	…	…	…	…	…	…
289.607843	-258.27451	12393.411765	118.568627	-271.27451	89.490196	2019-11-14 00:59:56.600
297.102041	-316.285714	12419.673469	136.653061	-273.979592	94.142857	2019-11-14 00:59:56.650
249.882353	-311.352941	12445.843137	141.294118	-279.372549	81.117647	2019-11-14 00:59:56.700
315.142857	-339.959184	12546.326531	130.489796	-274.857143	87.571429	2019-11-14 00:59:56.750
303.911765	-308.647059	12541.147059	114.705882	-274.588235	82.735294	2019-11-14 00:59:56.800

Introduction: Goals

- Goal: **Effective Data Management** within the MLOps cycle
  - How should we efficiently store, process, and manage our data?
  - What tools are available for streamlined data management?

- Goal: **Exploratory Data Analysis** (EDA)
  - How can we gain a better understanding of our dataset?
  - To improve train diagnostics and enable predictive maintenance
  - “Finding insights within unknown data”

DevOps is about Process and Tooling

- **DevOps**: Standardize processes and tools, between development and operations teams
  - In some cases, these teams are merged into one
- Goal: Quickly deploy R&D work to **production** with minimal changes
- **MLOps**: Extends DevOps to data science and machine learning
  - Keeps data science aligned with production needs

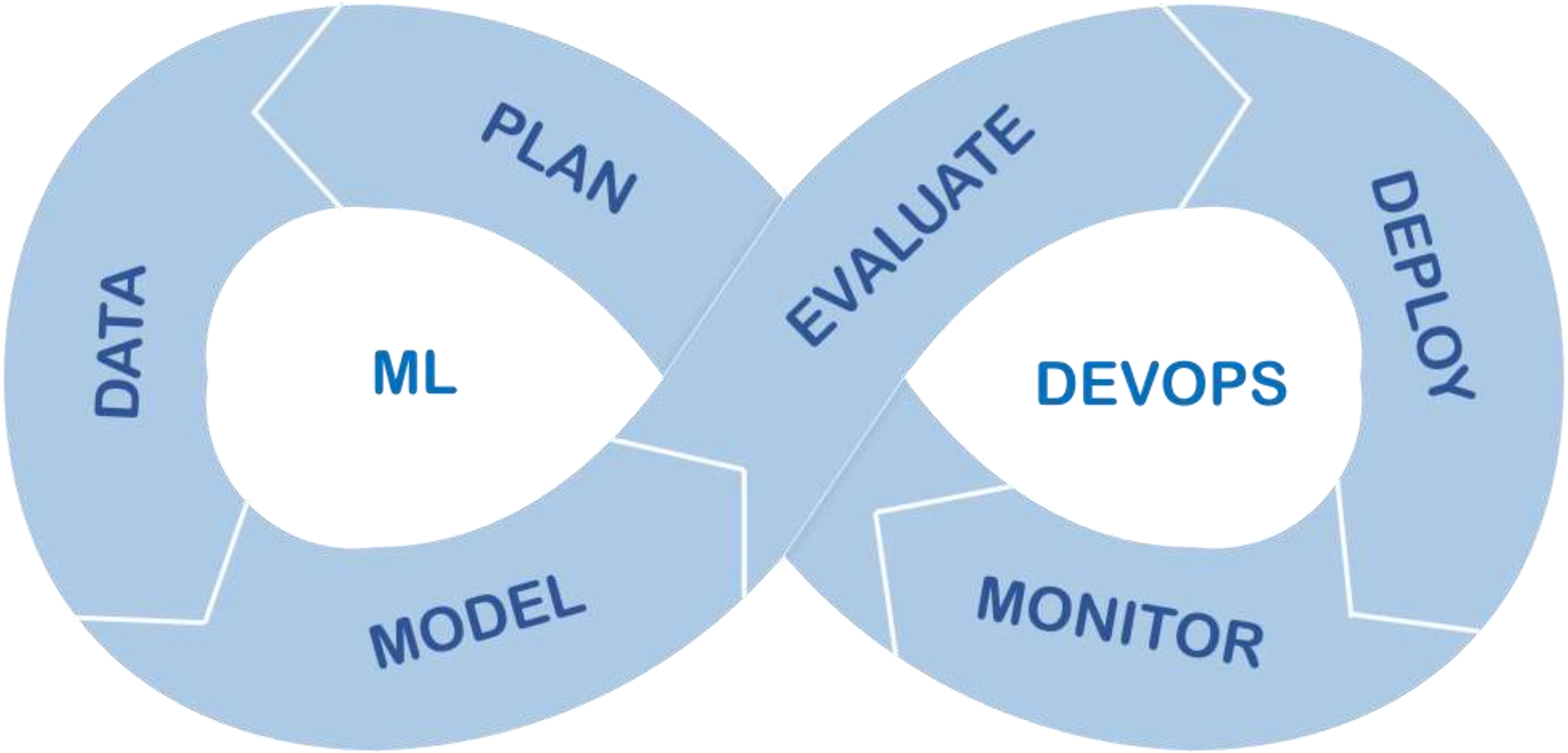
Data Management in the MLOps Cycle

- **Data Management**: Essential in every stage of the MLOps cycle

- For example:
  - **Data Ingestion**: Collect data from various sources
  - **Data Storage**: Use efficient formats like Parquet
  - **Data Processing**: Clean and prepare data for analysis
  - **Data Versioning**: Track changes for reproducibility

- **Goal**: Ensure data is reliable, accessible, and optimized for ML tasks

Data Management in the MLOps Cycle



So You Want to Do Data Science?

- Where will you **store** your data?
- How will you **process** it efficiently?
- Can you extract **insights** from the data?
- How will you **monitor** data in production?

- Remember: Real-world data is **messy**!

Which Programming Language to Use?

- Requirements for data exploration:
  - Easy to write and experiment with code
  - Widely supported by tools and libraries
- Possible candidates:
  - SQL: Popular, ideal for database processing
  - Python: Popular, extensive library support
  - R: Common in academia, focus on statistics
  - Julia: Modern, efficient, but less widely used
- Our choice: **Python**

On-Disk Data Storage

- **How to store Python datasets on disk?**

- **Pickle**: Quick, but unsafe and non-portable, avoid in production
- **CSV, JSON, XML**: Widely used, uncompressed text formats
  - Inefficient for large datasets (maybe .zip it?)
- **Apache Parquet**: Popular, compressed, ideal for large datasets
  - Optimizes storage space, lowering storage costs
  - Smaller files allow larger datasets to fit on disk
  - Supports big data exploration with simple hardware

01-pickle	
Type:	Bestandsmap
Locatie:	\\wsl.localhost\Ubuntu\home\alexander\Code\d
Grootte:	21,7 GB (23.371.598.444 bytes)

televic_data_190201.pkl	134.909 kB
televic_data_190202.pkl	91.449 kB
televic_data_190203.pkl	112.364 kB
televic_data_190204.pkl	43.536 kB

02-parquet	
Type:	Bestandsmap
Locatie:	\\wsl.localhost\Ubuntu\home\alexander\Code\d
Grootte:	7,68 GB (8.254.847.273 bytes)

gnss-190201.parquet	2.614 kB
gnss-190202.parquet	1.817 kB
gnss-190203.parquet	4.203 kB
gnss-190204.parquet	855 kB
imu-190221.parquet	14.724 kB
imu-190222.parquet	26.915 kB
imu-190223.parquet	29.603 kB
imu-190224.parquet	31.259 kB

In-Memory Data Handling

- **How to load Python datasets into memory?**
  - Python data structures are **slow** and **memory-inefficient**
- **Numpy**: Popular for numerical operations in Python
  - Less optimal for complex data types (e.g., string, datetime, dict)
- **Pandas**: Compact representation of tabular data
  - Very popular, but **Python-only**
- **Apache Arrow**: Modern, efficient in-memory format
  - Strong tool/language **interoperability**
  - Efficient memory layout, large datasets in memory

Efficient Data Processing

- **How to process datasets in Python?**

- **Pandas**: Popular but lacks parallelism
  - In-memory only, so can’t handle larger-than-RAM datasets
  - Best-known example of a **DataFrame** library (not using SQL)

- **Polars**: High-performance, **parallel** operations (lazy)
  - Supports **streaming** queries for larger-than-RAM data
  - Built on Rust, allows custom Rust extensions
  - DataFrame interface similar to Pandas

Efficient Data Processing

- Pandas / Polars: Python
- **DuckDB**: In-memory SQL analytics engine
  - Supports on-disk queries for larger-than-RAM data
  - SQL interface, like traditional databases
  - Ideal for large analytical queries (S3)

- All three support the **Arrow and Parquet** formats
  - Zero-copy Arrow between Polars and DuckDB

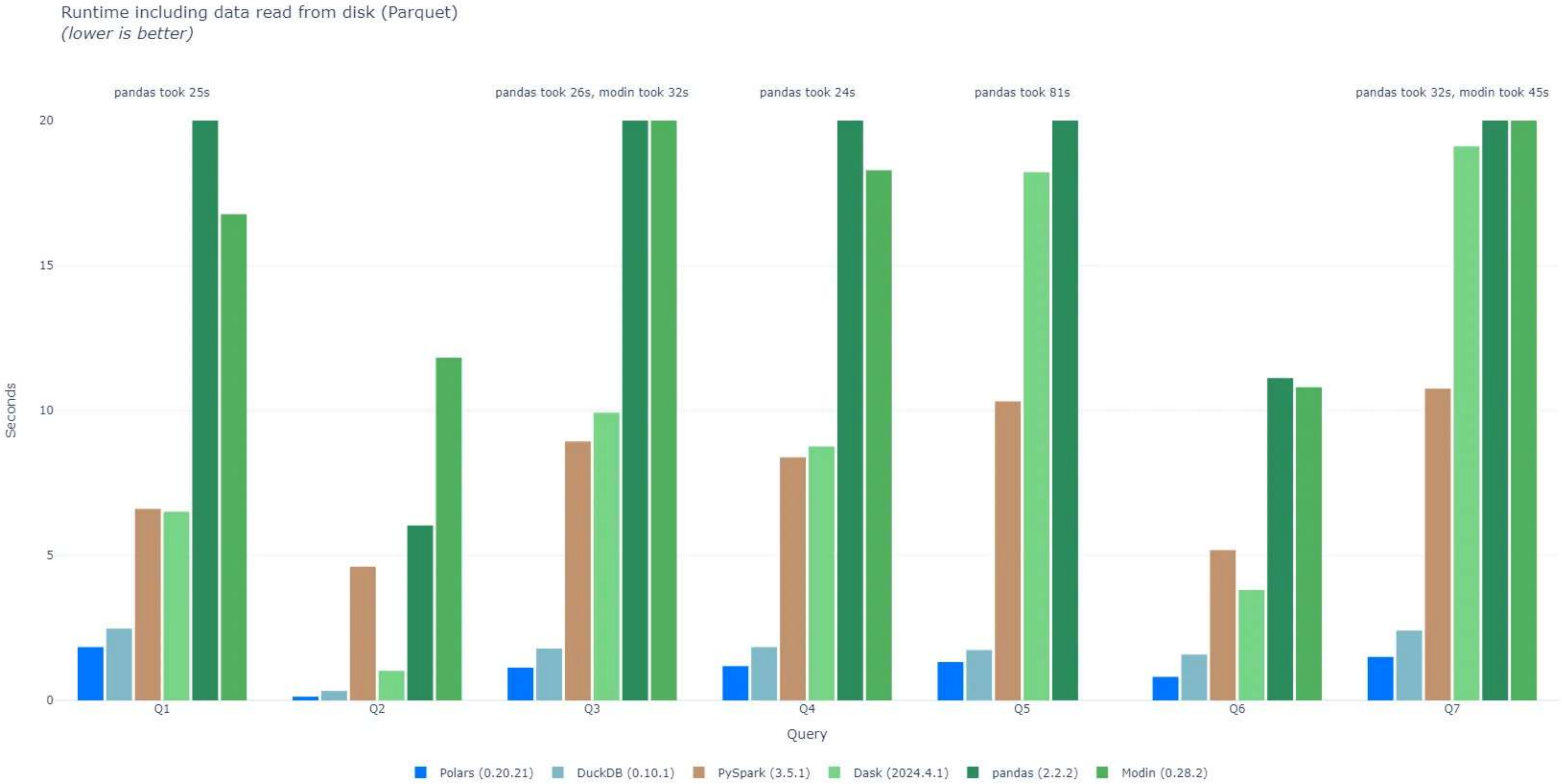
- **Conclusion**: Arrow in-memory, Parquet on-disk

Extremely Large Datasets

- **Dask, PySpark**: Distributed compute engines for Python
- **Ray**: Distributed machine learning in Python
- **ClickHouse**: Distributed SQL-based data warehouse

- With great power comes… complexity and overhead
  - Polars and DuckDB are faster on a single node

If you can perform the task on a single machine, then perhaps you should



Runtime including data read from disk (Parquet)
(lower is better)
pandas took 25s
pandas took 26s, modin took 32s
pandas took 24s
pandas took 81s
pandas took 32s, modin took 45s
20
15
10
5
0
Seconds
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Query
Polars (0.20.21)
DuckDB (0.10.1)
PySpark (3.5.1)
Dask (2024.4.1)
pandas (2.2.2)
Modin (0.28.2)

```python
import polars as pl

data = pl.scan_parquet("../data/02-parquet/gnss-*.parquet")
data = data.with_columns(
    (pl.col("gnss_lat") * 100).round().alias("round_lat"),
    (pl.col("gnss_lon") * 100).round().alias("round_lon"),
)
data = data.group_by("round_lat", "round_lon").agg(
    pl.col("gnss_lat").mean().alias("lat"),
    pl.col("gnss_lon").mean().alias("lon"),
    pl.col("gnss_speed").mean().alias("speed"),
    pl.len().alias("count").cast(pl.Float32),
)
data = data.select("lat", "lon", "speed", "count")
data = data.collect(streaming=True)
```

✓ 0.8s

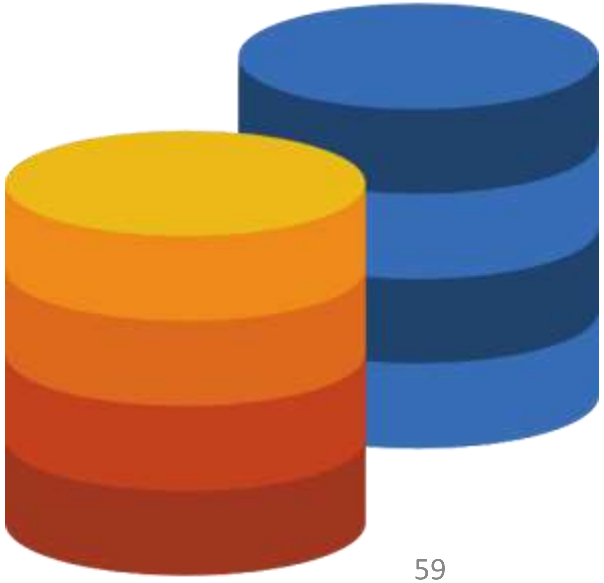
✓ 0.8s

Understanding OLTP vs OLAP

- Two types of **databases**: OLTP and OLAP

- **OLTP (Online Transaction Processing)**
  - Real-time, transactional operations, not analytics
  - Efficient at frequent inserts, updates, and deletes
  - Examples: booking tickets, processing payments

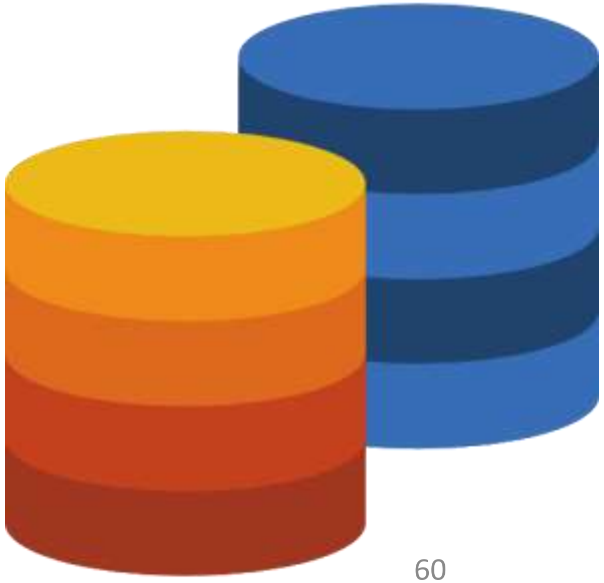
  - **SQL**: Relational database systems, well-known OLTP
  - Key-value, NoSQL: pros and cons, but all are OLTP



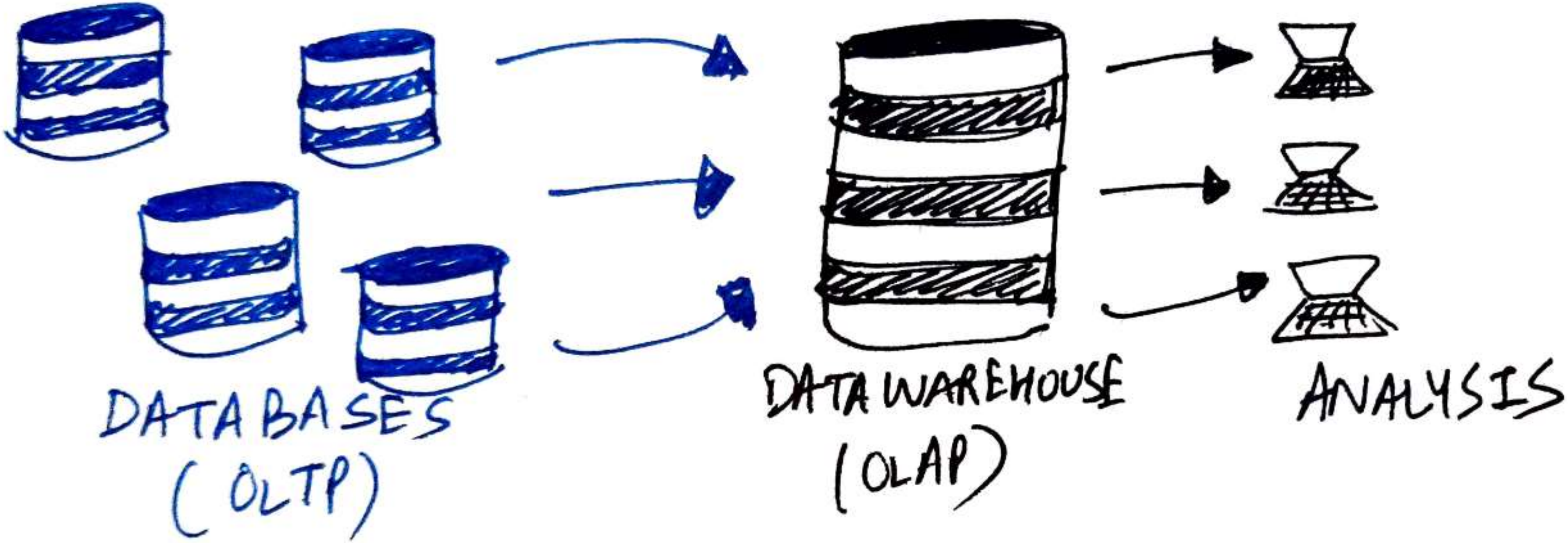
Understanding OLTP vs OLAP

- OLTP (Online Transaction Processing)
- **OLAP (Online Analytical Processing)**
  - Analyzes large datasets and reads data quickly
  - Data is written infrequently but **read extensively**
  - Commonly used in data warehouses or data lakes
  - Tools: Pandas, Polars, DuckDB, ClickHouse

- **Conclusion**: Use **OLAP** for pipelines and exploration
  - Avoid overloading OLTP systems with analytics queries
  - OLTP’s constantly changing data complicates analysis



Understanding OLTP vs OLAP

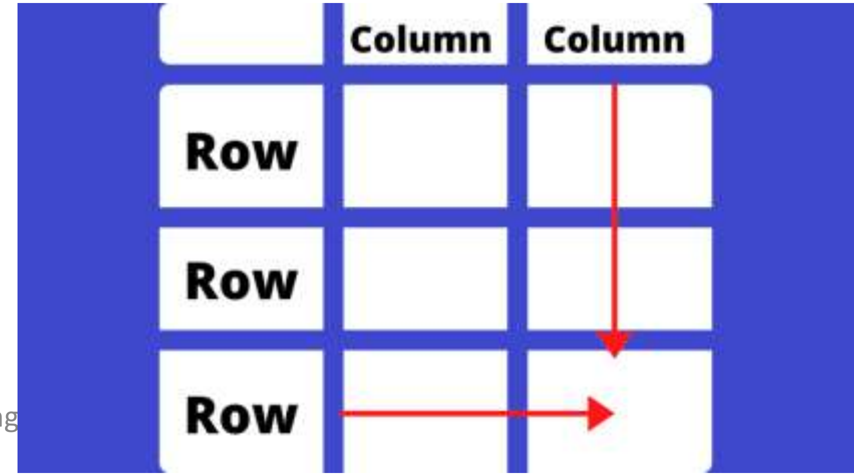


Row-Based vs Columnar Formats

- Two types of **data formats**: row-based and columnar

- **Row-Based Formats**
  - Examples: CSV, Pandas, traditional databases
  - Data stored row-by-row, fields for a single record are kept together
  - **Ideal for OLTP**: Fast for inserting/updating rows, data is localized
  - **Bad for OLAP**: Reading single column loads all rows

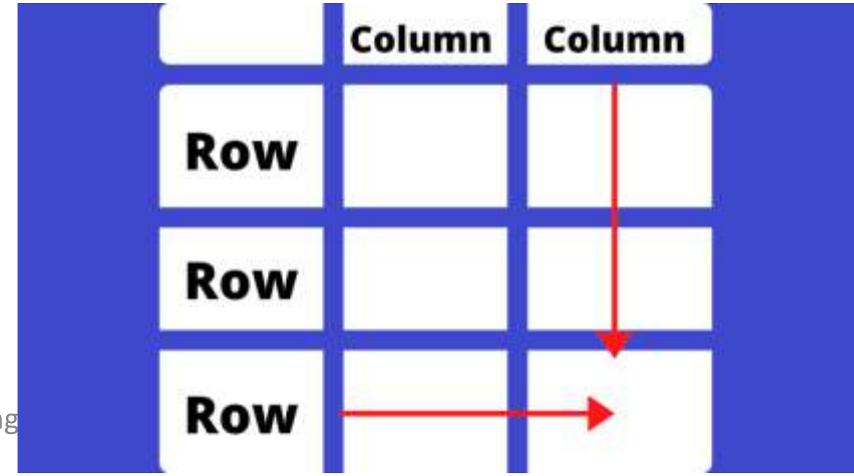
  - **Example**: Drawing a map of GPS locations
    - Loads timestamps and vibration data
    - Even when only location data is needed



Row-Based vs Columnar Formats

- **Columnar Formats** (e.g., Arrow, Parquet)
  - Data stored column-by-column
  - **Ideal for OLAP**: Load only needed columns for analysis
  - **Example**: Analyze vibrations without loading GPS locations
  - **Compression**: Similar data types stored together
    - Reduced storage costs and improved query speeds

- **Conclusion**: Prefer **columnar** file formats
  - For data pipelines and exploration
  - Modern OLAP systems use columnar storage



Where to Store Data?

- **Local Files / Network File Systems**
  - No built-in versioning (e.g., ext4, zfs, nfs)
  - Require external solutions (backups)

- **Object Storage** (e.g., Amazon S3, GCS, Azure)
  - Cost-effective, scalable, cloud or on-premise (MinIO)
  - Easy versioning: use hash of content as filename
    - Content-addressable storage (CAS)
    - Files remain unchanged unless deleted

Where to Store Data?

- **“Git for Large Files” Solutions**
  - **Git**: Built for text, struggles with large files
  - **Git-LFS**: Suitable for code assets, limited scalability
- **DVC**
  - Integrates with Git for data versioning
  - Supports datasets with S3-compatible storage
- **LakeFS**
  - Git-like version control for data lakes (S3)
  - Tracks data changes, supports branching at scale

Storing Tabular Data in S3

- **S3 as a Data Lake**
  - Scalable, cost-effective storage
  - DIY approach: write data directly to S3 using Python scripts

- **Parquet Files**
  - Columnar format, supports partial column reading
  - Tools like **Polars** and **DuckDB** can read directly from S3
  - Enables data exploration without full downloads

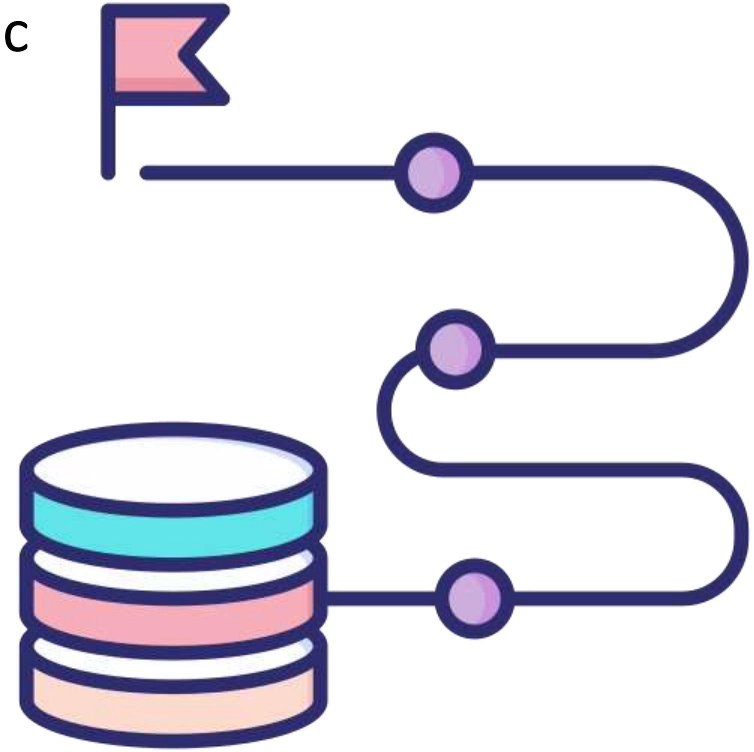
Storing Tabular Data in S3

- **Delta Tables & Apache Iceberg**
  - Update data without re-uploading entire files
  - Maintain versioning and support ACID transactions

- **Data Lake Platforms**
  - Examples: **Delta Lake** (Databricks), **Apache Hudi**
  - Handle schema evolution and incremental processing
  - Reliability and performance for **large-scale operations**
  - Simplify complex data management

How to Schedule Data Processing?

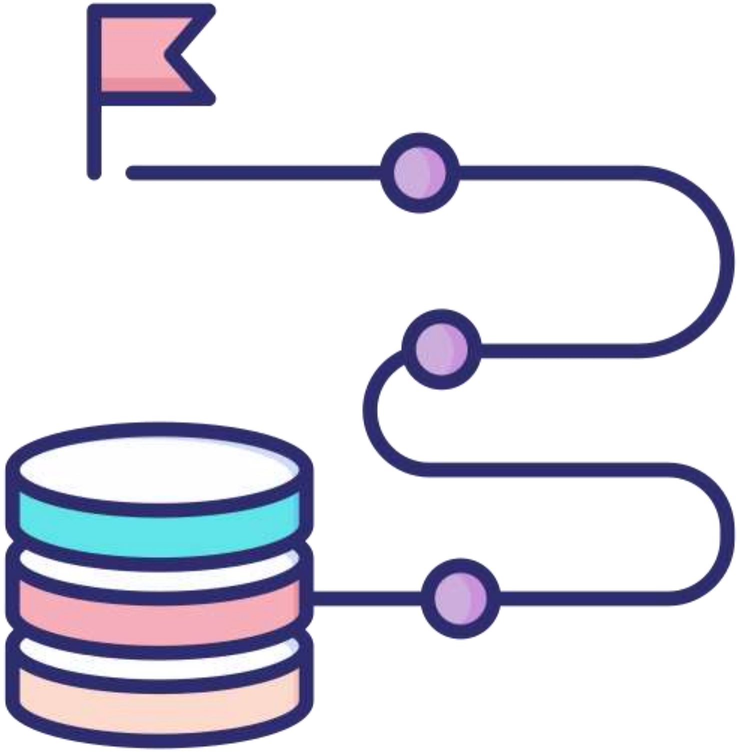
- **Automating Data Science**
  - Systems must run without manual intervention
  - Processes should be reproducible and deterministic
  - Ensure continuity if people are unavailable



- **Job Scheduling & Pipelines**
  - **Airflow**: Popular but older
  - **Prefect / Dagster**: Python-focused
  - **Argo / Flyte**: Kubernetes-based
  - **dbt**: SQL-based

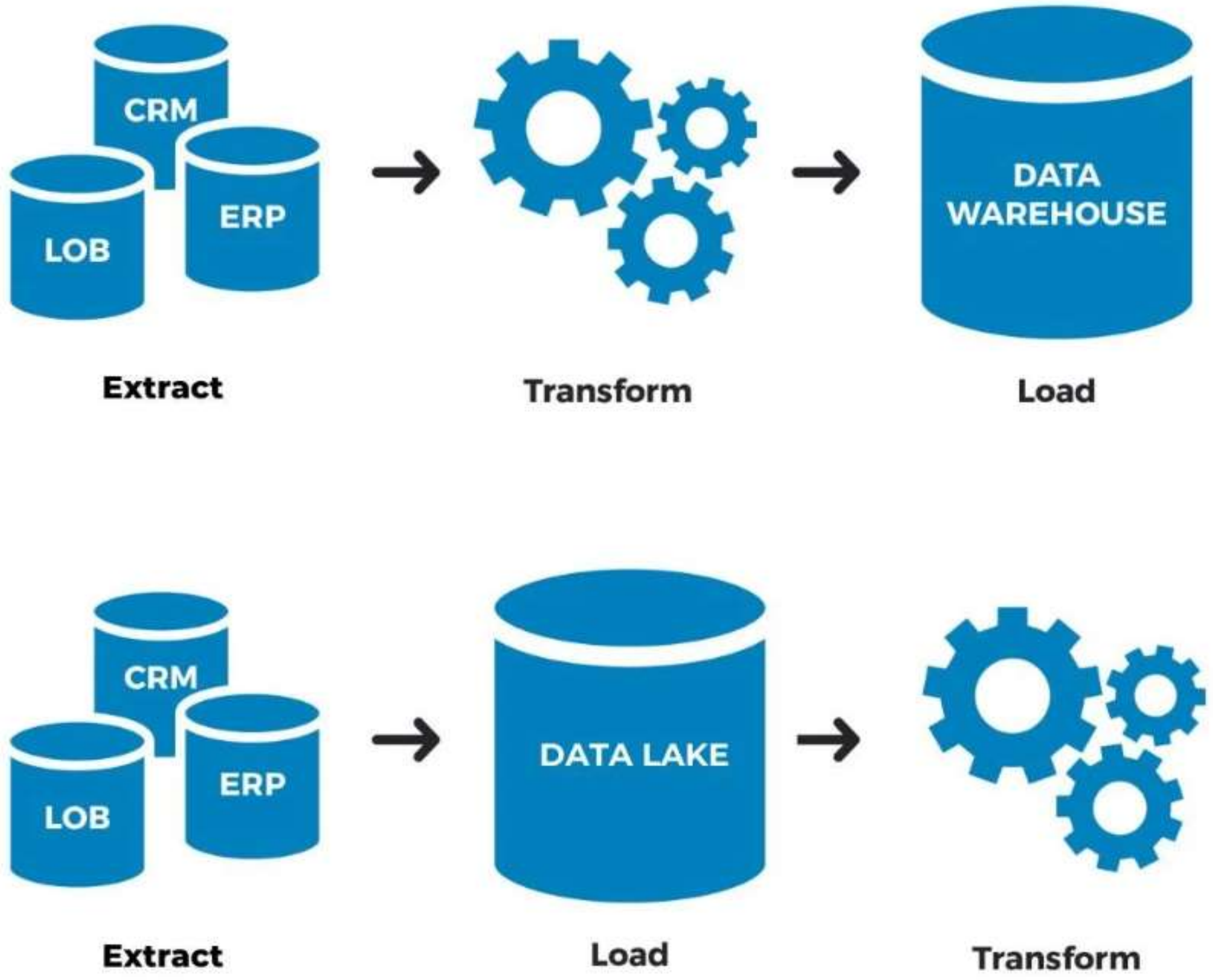
How to Schedule Data Processing?

- **Features of Pipelines**
  - Scheduling (daily, hourly)
  - Task dependencies (order, events)
  - Parallel task execution
  - Monitoring and alerts
- **Common Use Cases**
  - ETL/ELT: Load data from OLTP to OLAP
  - Poll external sources (e.g. REST APIs)
  - Process data queues (e.g. Redis)



Getting Data into OLAP

- **Batch vs Stream**
  - Batch (ETL/ELT): Loads data on a schedule
  - Stream: Continuously updates data in real-time
- **ETL**: Extract, Transform, Load
  - Data transforms outside OLAP, then loads in
  - **Pros**: Saves OLAP storage (only aggregates)
  - **Cons**: Partial data history, mixed processing
- **ELT**: Extract, Load, Transform
  - Data loads first, then transforms inside OLAP
  - **Pros**: Simplified loading, consistent processing


CRM
LOB
ERP
DATA WAREHOUSE
Extract
Transform
Load
CRM
LOB
ERP
DATA LAKE
Extract
Load
Transform

What We Learned So Far

- We've covered:
  - **File formats** for data storage (Parquet)
  - **Tools** for data processing (Pandas, Polars, PySpark)
  - **Scheduling** data processing (Airflow, Prefect)
  - **Storage/versioning** options (S3, DVC, data lakes)

- What's missing:
  - Data remains abstract, a "black box"
  - Millions of data rows aren't easily understood
  - How can we explore data? Gain real insights?

Exploratory Data Analysis (EDA)

- We need powerful tools to **explore data**
- Preferably open-source, supporting Python/Linux

- Don't waste time on difficult tools
- Focus on **understanding and insight**

Basic Software Tools

- **Jupyter Notebooks (.ipynb)**
  - Great for exploration, less ideal for production
  - Use sparingly, operations teams prefer Python .py
  - Code in Python modules, import in notebooks

- **VSCode with Remote Connection**
  - Better than JupyterLab for development
  - Port forwarding built into VSCode
  - Connect to a powerful VM on server
  - GPU passthrough for ML (if needed)

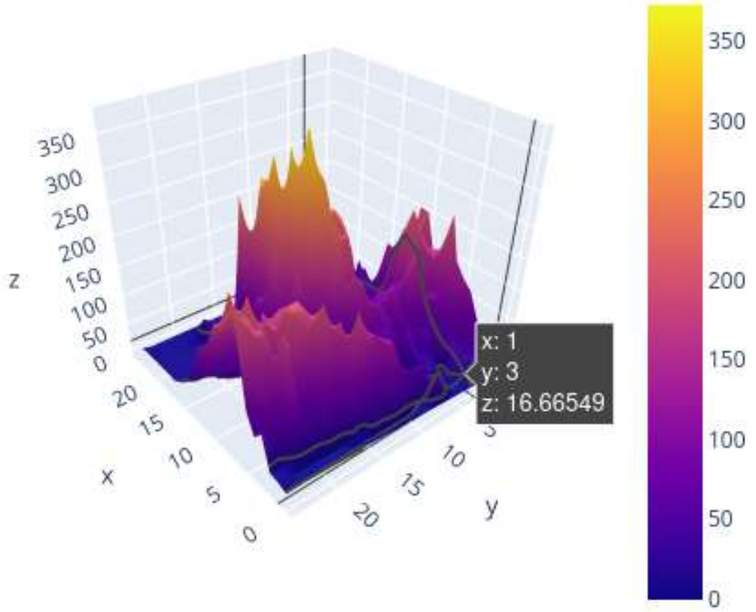
Basic Software Tools

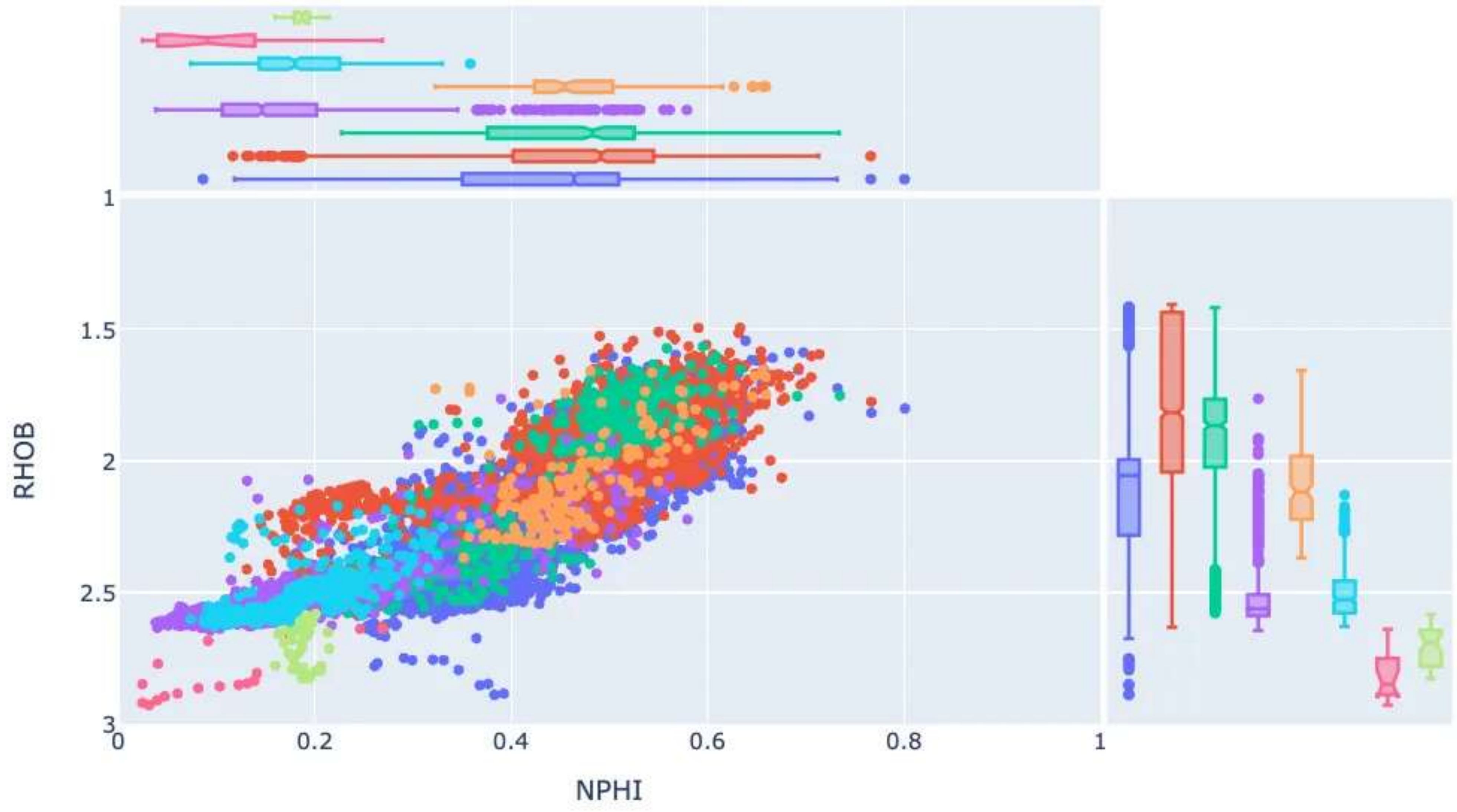
- **Always work in Docker containers**
  - The best for dependency management
  - Improves code deployment to production
  - Colleagues can reproduce your environment
  - Data pipelines run without surprises

- **Version control code with Git**
  - Awkward with Jupyter notebooks (JSON, output)
  - Use tools like `nbstripout` and `nbdime`
  - Use a separate tool (e.g., DVC) to version data

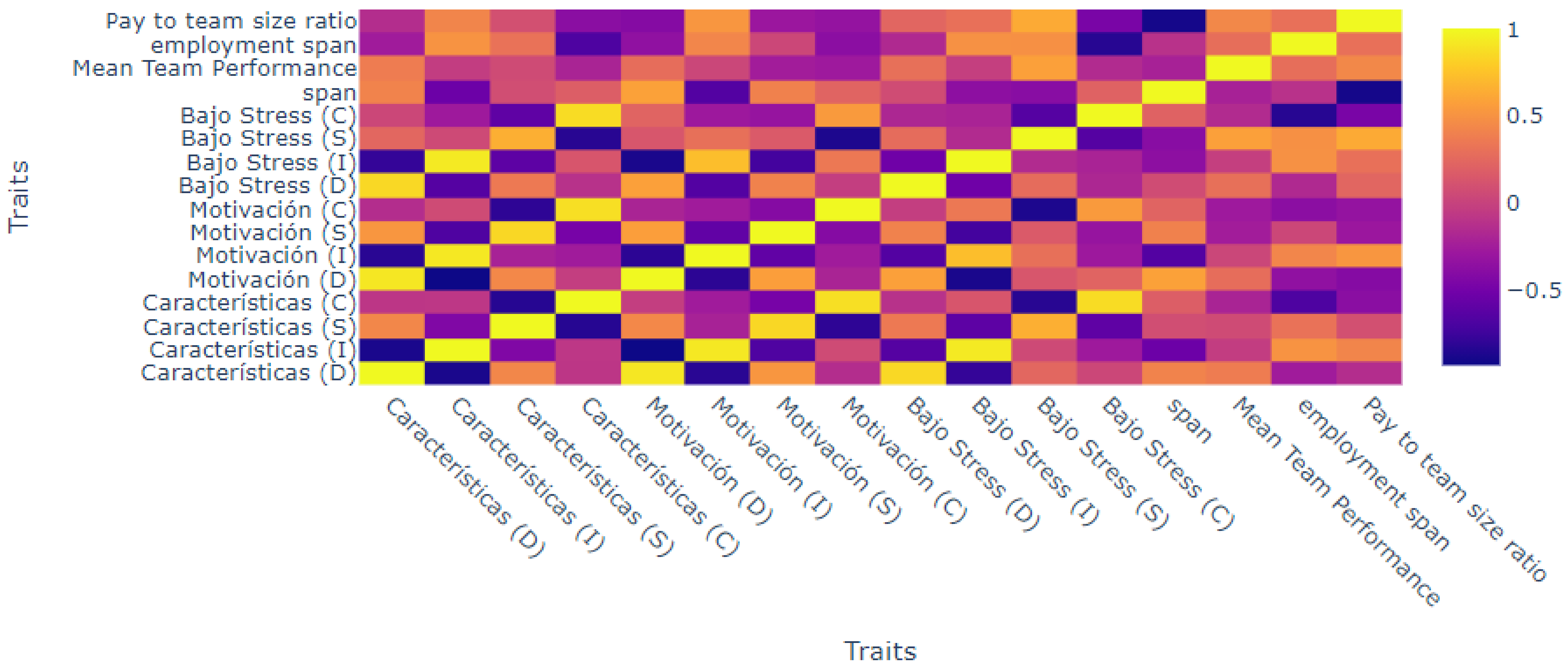
Visualizing Data

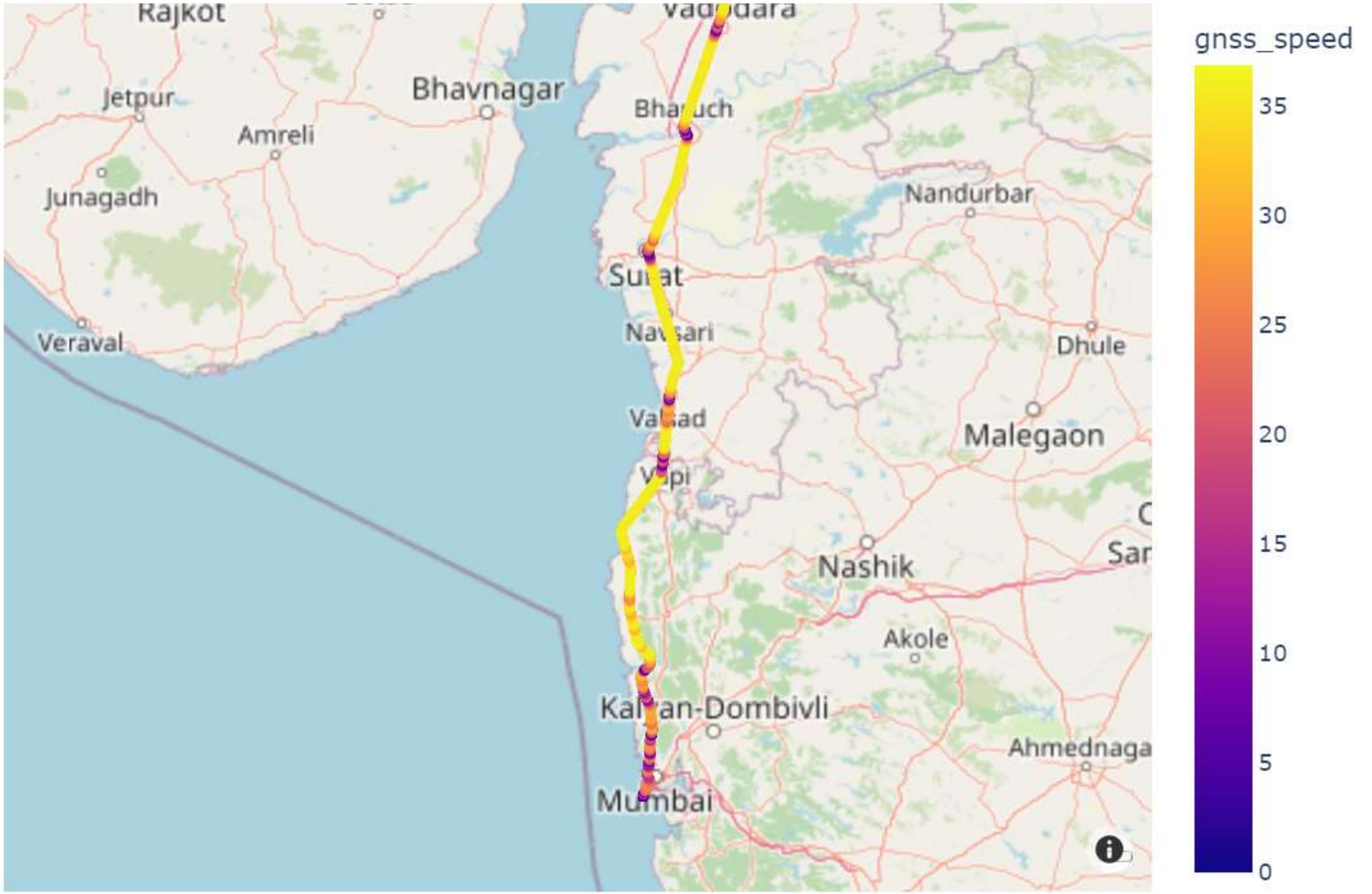
- **Why Visualize?**
  - To gain insight and reveal patterns
  - Makes complex data easier to understand
- **Interactive Visuals**
  - Graphs, maps, heatmaps, histograms
  - Interactivity supports deeper exploration
- **Matplotlib**: Widely used, basic interactivity
- **Plotly**: Modern, interactive, good-looking



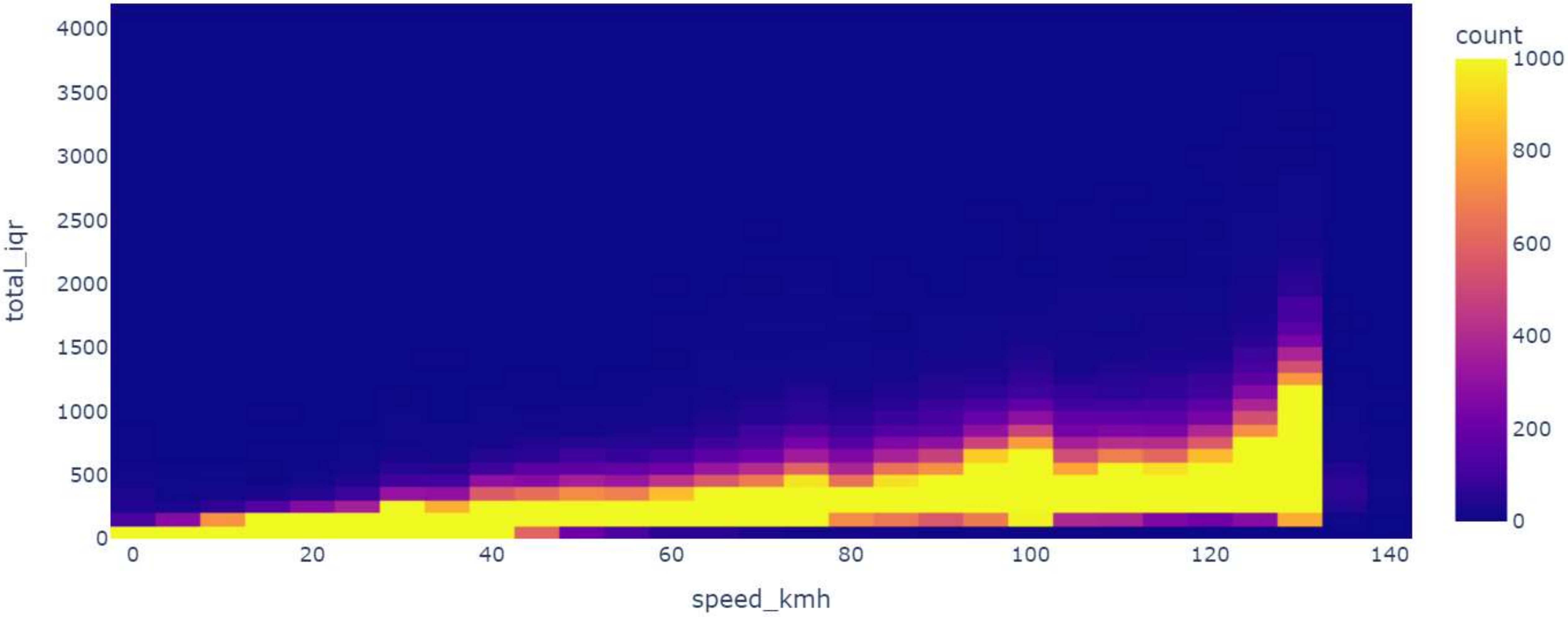

RHOB
NPHI
1
1.5
2
2.5
3
0
0.2
0.4
0.6
0.8
1

Correlation heatmap





Rajkot
Vadodara
Jetpur
Bhavnagar
Bharuch
Amreli
Junagadh
Nandurbar
Surat
Veraval
Navsari
Dhule
Valsad
Malegaon
Vapi
Nashik
Akole
Kalyan-Dombivli
Ahmednaga
Mumbai
gnss_speed
35
30
25
20
15
10
5
0

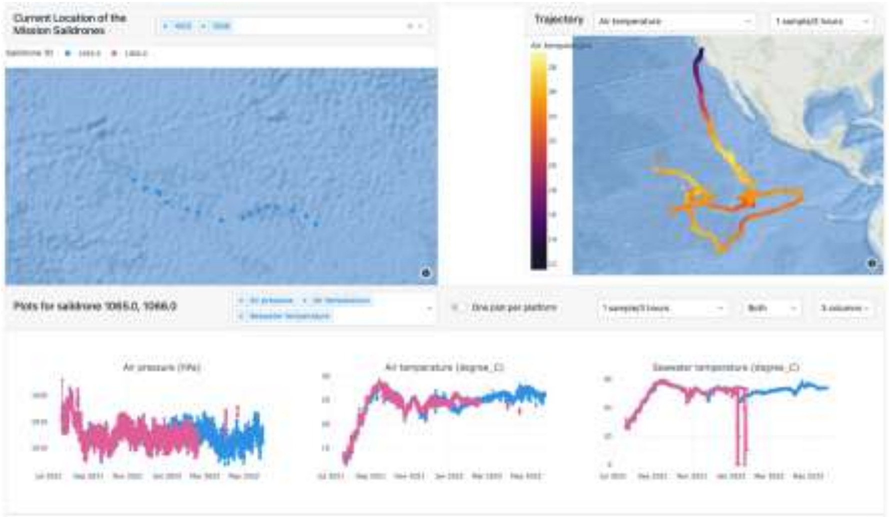

count
1000
800
600
400
200
0
4000
3500
3000
2500
2000
1500
1000
500
0
total_iqr
0
20
40
60
80
100
120
140
speed_kmh

Building Dashboards

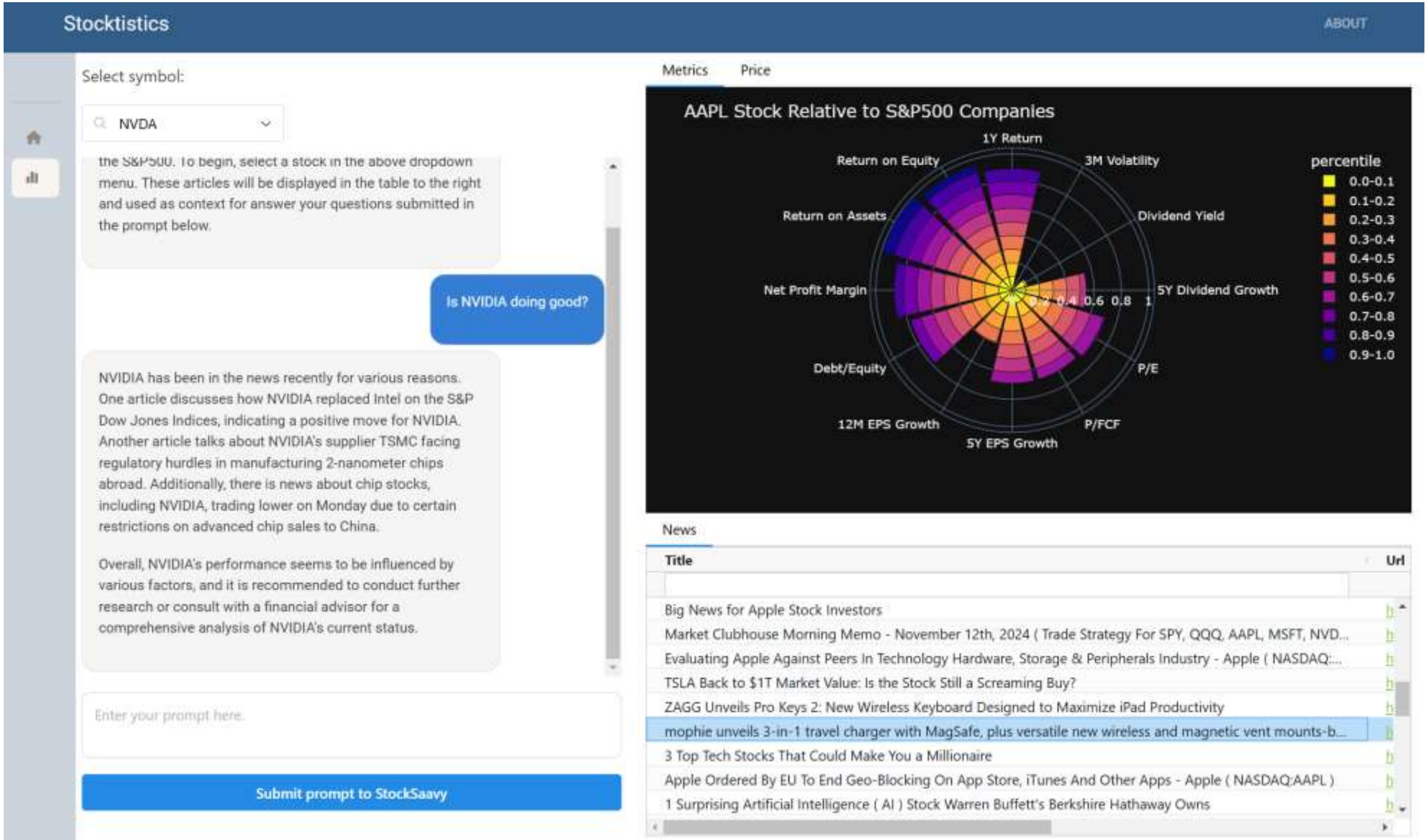
- **Why Create Dashboards?**
- **Exploratory Development**
  - Enables quick, iterative data exploration
  - Interactive, complex visualizations
- **Monitoring and Deployment**
  - Real-time monitoring post-deployment
  - Useful for communication and insights

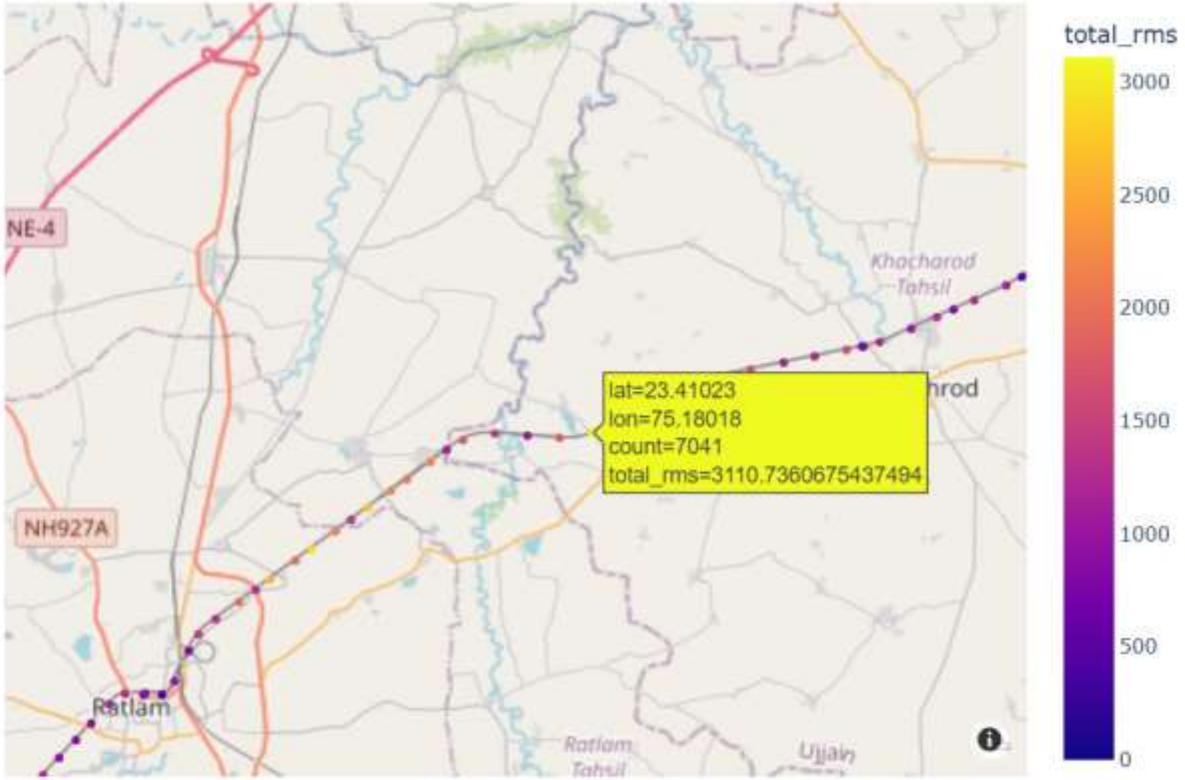
Building Dashboards

- **Streamlit, Gradio**
  - Very easy for simple, fast dashboards
  - Gradio works inside notebooks
- **Plotly Dash**
  - Framework for interactive dashboards
  - Scales to complex interfaces (React)
- **Grafana / Prometheus**
  - Ideal for monitoring, less for exploration

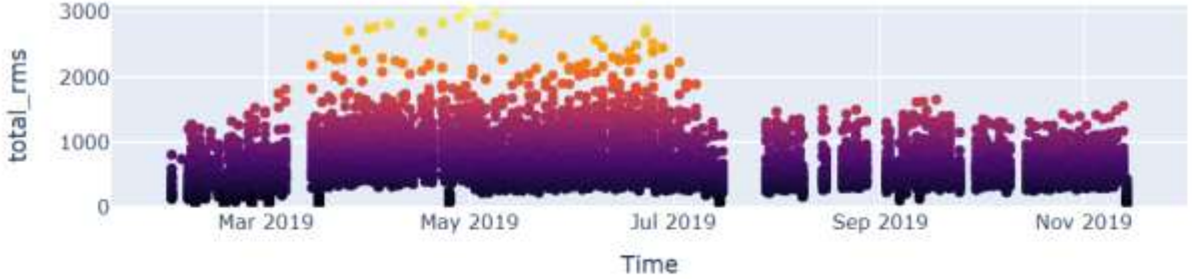



Hide path
Show path
Select A Property To Display
Elevation
651
Temperature
286
Speed
28.0
Time
08:29
Latitude
00595902
Longitude
00067000
Solar-Panel-0
Solar-Panel-1
Camera
Thrusters
Motor
Signal
Fuel Level
Battery-Level


Stocktistics
ABOUT
Select symbol:
NVDA
the S&P500. To begin, select a stock in the above dropdown menu. These articles will be displayed in the table to the right and used as context for answer your questions submitted in the prompt below.
Is NVIDIA doing good?
NVIDIA has been in the news recently for various reasons. One article discusses how NVIDIA replaced Intel on the S&P Dow Jones Indices, indicating a positive move for NVIDIA. Another article talks about NVIDIA's supplier TSMC facing regulatory hurdles in manufacturing 2-nanometer chips abroad. Additionally, there is news about chip stocks, including NVIDIA, trading lower on Monday due to certain restrictions on advanced chip sales to China.
Overall, NVIDIA's performance seems to be influenced by various factors, and it is recommended to conduct further research or consult with a financial advisor for a comprehensive analysis of NVIDIA's current status.
Enter your prompt here.
Submit prompt to StockSaavy
Metrics
Price
AAPL Stock Relative to S&P500 Companies
1Y Return
3M Volatility
Dividend Yield
5Y Dividend Growth
P/E
P/FCF
5Y EPS Growth
12M EPS Growth
Debt/Equity
Net Profit Margin
Return on Assets
Return on Equity
0.6
0.8
1
percentile
0.0-0.1
0.1-0.2
0.2-0.3
0.3-0.4
0.4-0.5
0.5-0.6
0.6-0.7
0.7-0.8
0.8-0.9
0.9-1.0
News
Title
Url
Big News for Apple Stock Investors
Market Clubhouse Morning Memo - November 12th, 2024 (Trade Strategy For SPY, QQQ, AAPL, MSFT, NVD...
Evaluating Apple Against Peers In Technology Hardware, Storage & Peripherals Industry - Apple (NASDAQ:...
TSLA Back to $1T Market Value: Is the Stock Still a Screaming Buy?
ZAGG Unveils Pro Keys 2: New Wireless Keyboard Designed to Maximize iPad Productivity
mophie unveils 3-in-1 travel charger with MagSafe, plus versatile new wireless and magnetic vent mounts-b...
3 Top Tech Stocks That Could Make You a Millionaire
Apple Ordered By EU To End Geo-Blocking On App Store, iTunes And Other Apps - Apple (NASDAQ:AAPL)
1 Surprising Artificial Intelligence (AI) Stock Warren Buffett's Berkshire Hathaway Owns



NE-4
NH927A
Khacharod
Tahsil
hrod
Ratlam
Ratlam
Tahsil
Ujjain
lat=23.41023
lon=75.18018
count=7041
total_rms=3110.7360675437494
total_rms
3000
2500
2000
1500
1000
500
0

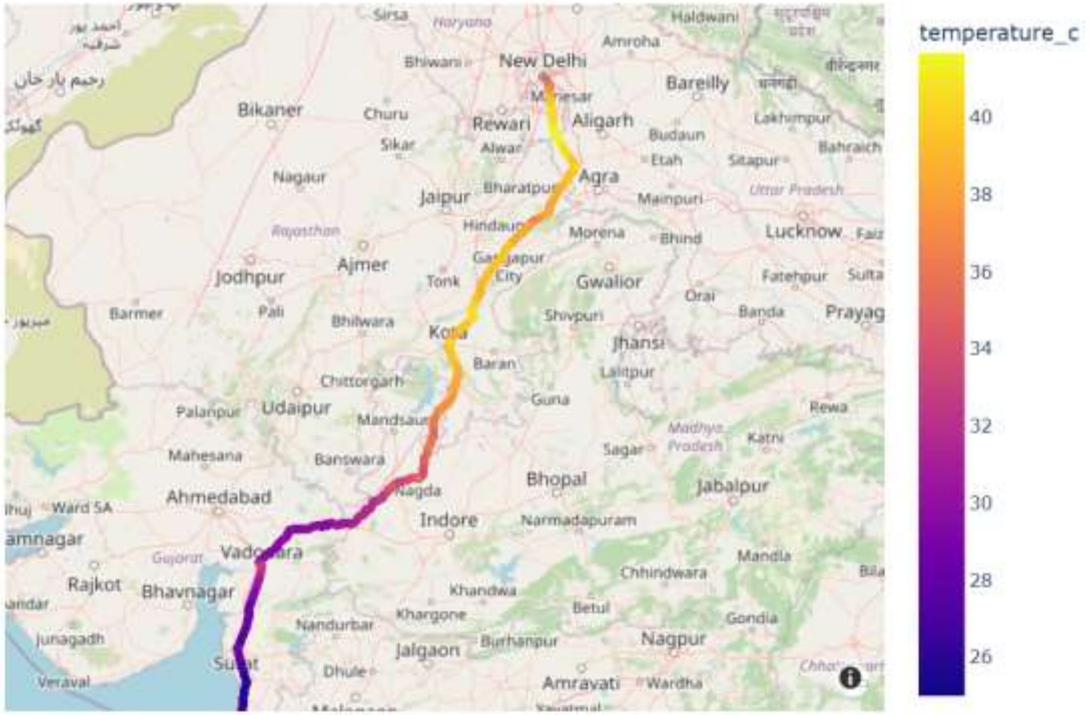


total_rms
3000
2000
1000
0
Mar 2019
May 2019
Jul 2019
Sep 2019
Nov 2019
Time

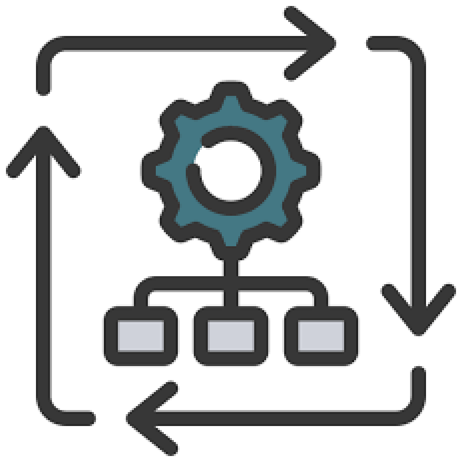
Televic: Demo Time!

- **Explore it yourself:**
  - [https://**train1.devbitapp.be**](https://train1.devbitapp.be)
  - [https://**train2.devbitapp.be**](https://train2.devbitapp.be)
    - **User**: mlops
    - **Password**: 4ecm

- **Technologies used:**
  - Parquet files
  - Polars processing
  - Plotly Dashboard
  - Git / DVC versioning
  - Proxmox server cluster



Auto deployment & energy monitoring



Sille Van Landschoot

Auto deployment & energy monitoring

- **Energy monitoring of AI models on embedded devices**
  - 150+ models
  - 6+ targets
- Multiple firmware implementations and revisions
- New targets and new models at any time

  #tests = #models x #targets x #version
- Solution: Automate!
  - How? **Prefect**?

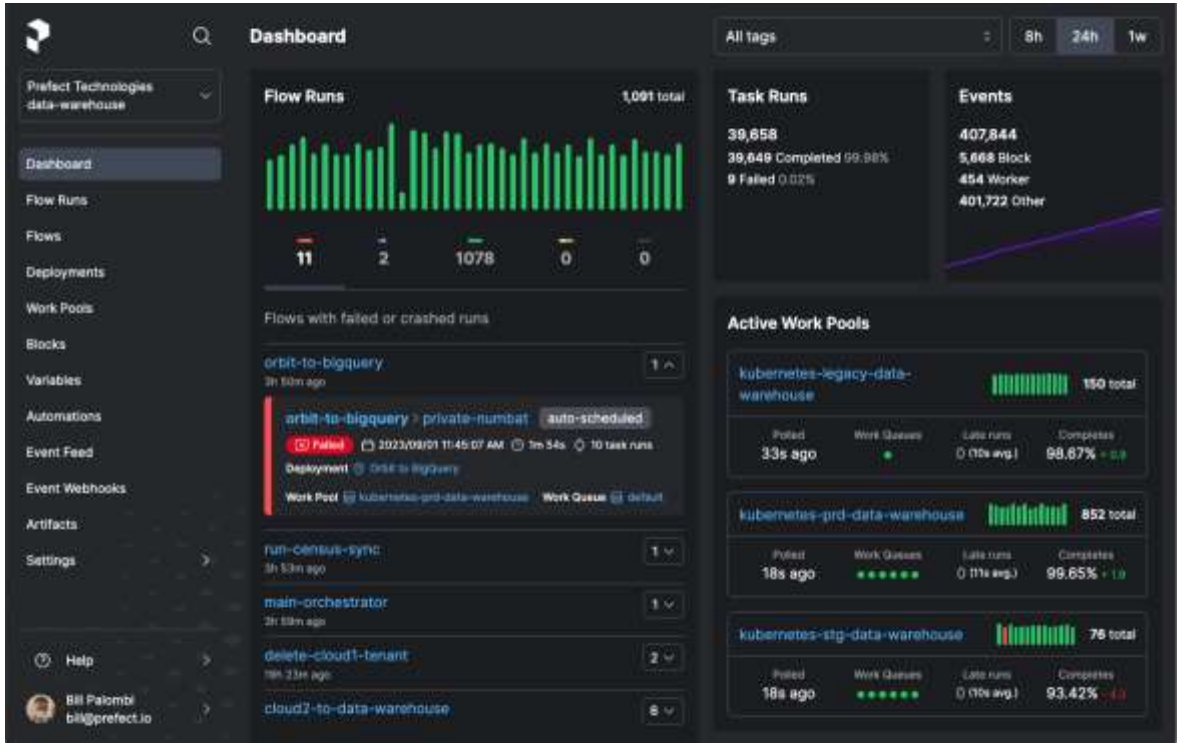
PREFECT

What is Prefect?

Quick introduction

Prefect

- “Modern **workflow orchestration** for data and ML engineers”
  - Open source
  - Manage, schedule, and monitor data workflows
  - Flexibility and scalability for various automation tasks
  - Useful in data engineering, MLOps, and machine learning pipelines
- **Python-First** and Code-Driven
- **Integrations** with ML libraries, data sources and platforms

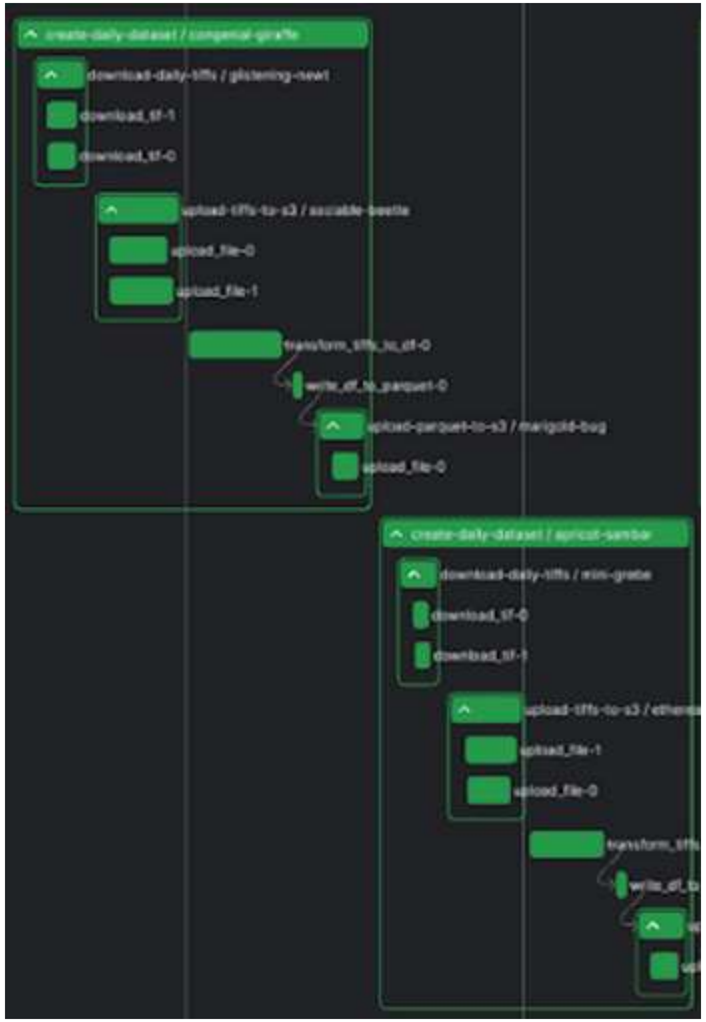


Prefect

- **Flow & Task Abstraction**
  - Organize tasks into flows to structure workflows easily
  - Task dependencies defined intuitively in Python
- **Resilience & Fault Tolerance**
  - Automatic retries, timeouts, and error handling
  - Options to resume, skip, or rollback tasks upon failure
- **Observability & Real-Time Monitoring**
  - Track task status, view logs, and monitor performance
  - Real-time alerts for task failures or anomalies

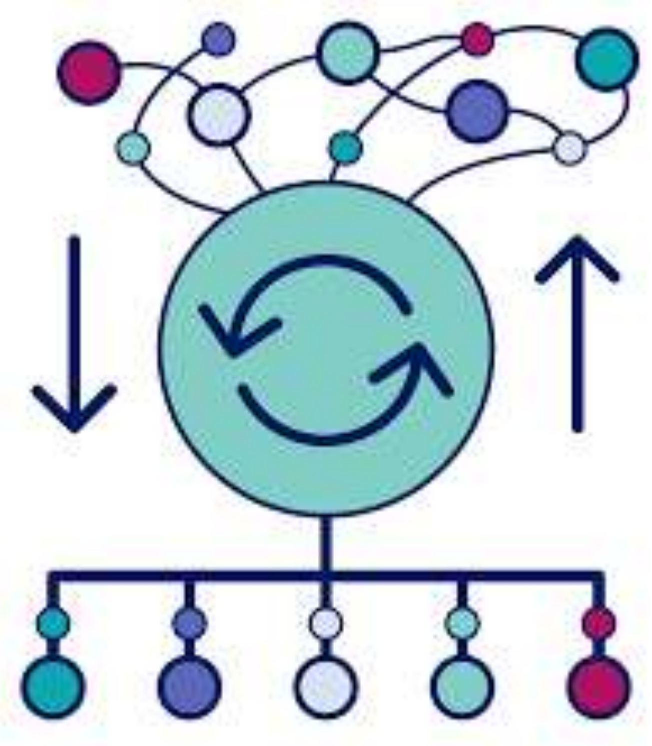
Prefect

- **Seamless Scheduling & Triggering**
  - Schedule model training, data pipelines, and more
  - Supports both cron and event-based triggers (e.g., based on file uploads)
- **Dynamic Workflow Management**
  - Easily adjust workflows as models evolve over time
  - Version control for tracking and iterating on flows
- **Integration with ML Tools**
  - Prefect works with ML libraries, cloud storage, databases, and API services for a complete pipeline



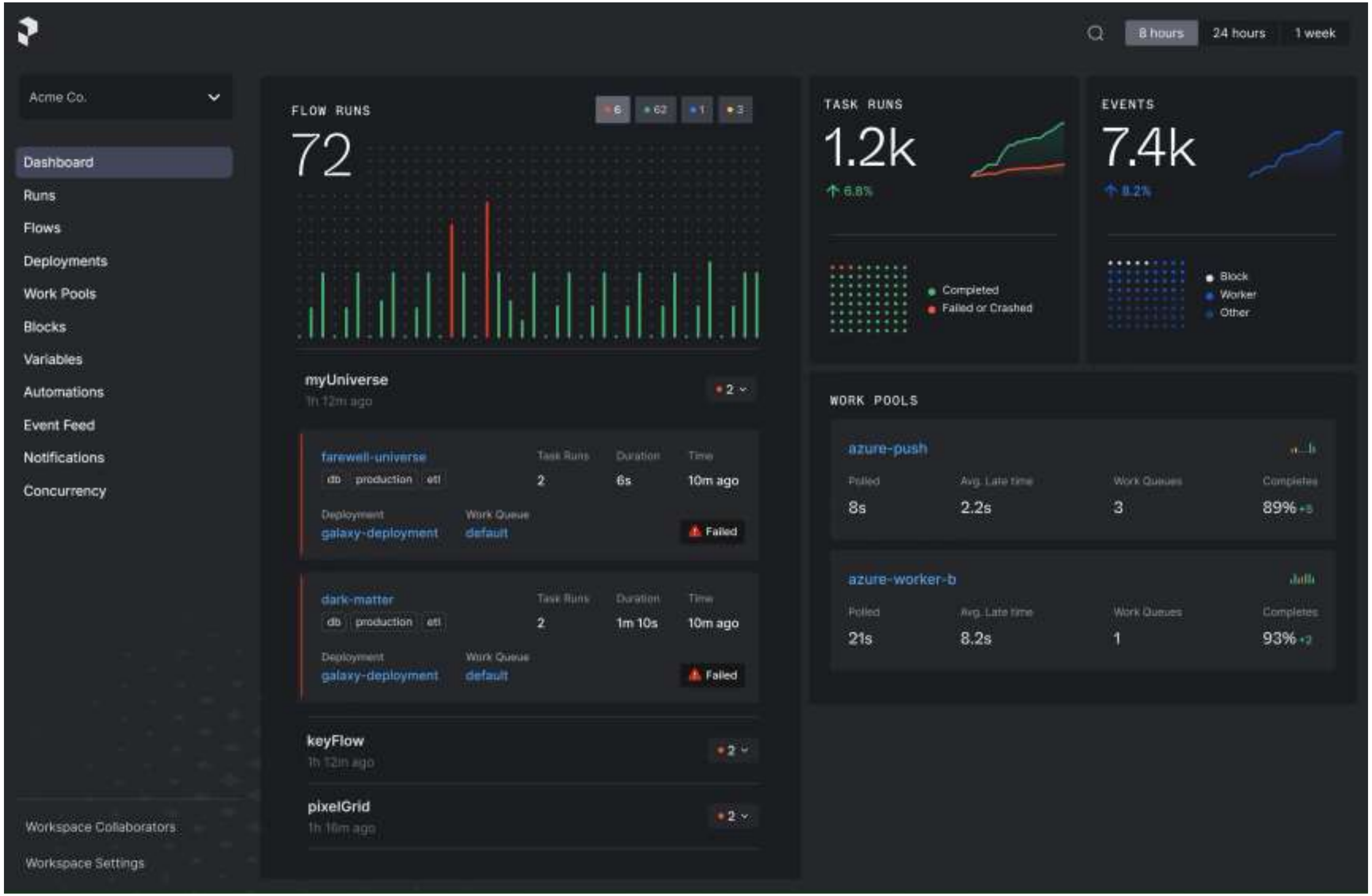
Why use Prefect ?

Task orchestration



Prefect Dashboard

- Server overview
- Runs
- Flows
- Deployments
- Work Pools
- Blocks
- Variables
- Automations
- Event Feed
- Notifications
- Concurrency



Prefect Tasks and Flows

- **Tasks**: Small, reusable functions representing individual steps in a workflow (e.g., data extraction, processing).
- **Flows**: Organized sequences of tasks forming a complete workflow or pipeline.
- **Modular Design**: Build, reuse, and manage tasks within flows for efficient development.
- **Dependency Management**: Control task order and relationships for accurate, predictable execution.

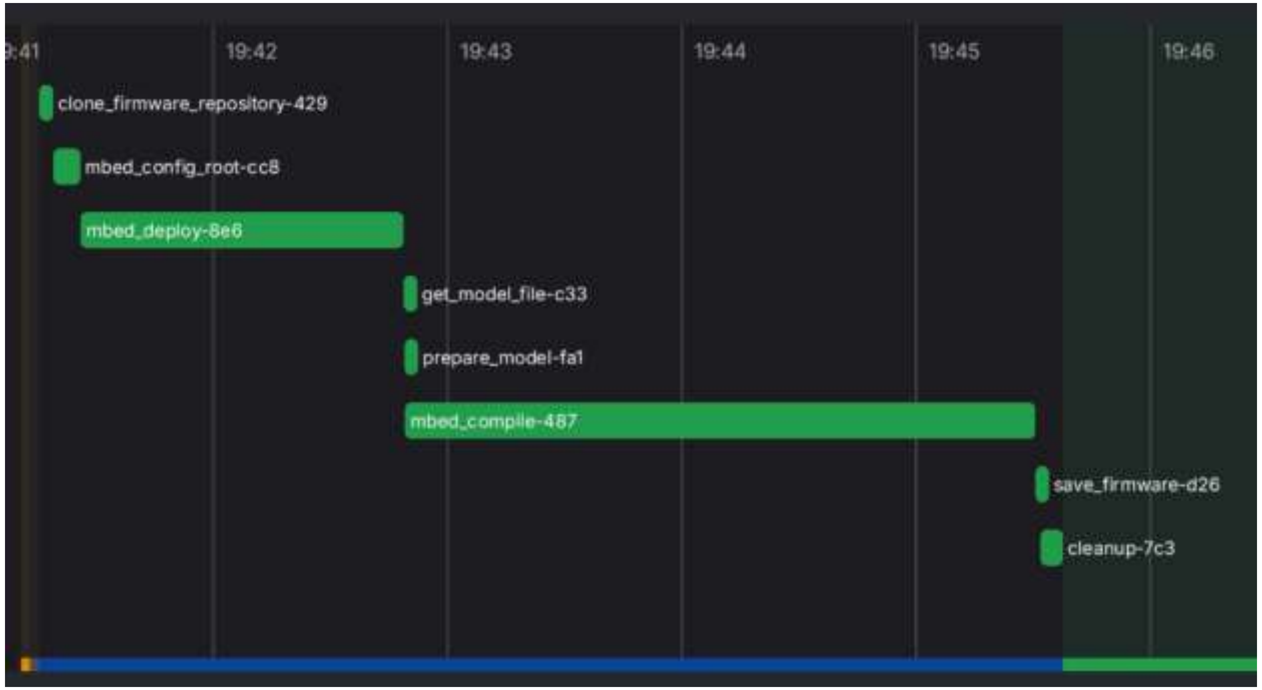
```python
@task
def mbed_compile(target, compiler) -> None:
    run_mbed_container(f"mbed compile -m {target} -t {compiler}")

@task
def prepare_model(tflite_file: str) -> None:
    model_file_name = "model.h"
    shell_run_command(command=f"xxd -i {tflite_file_name} > {model_file_name}")
    replace_text = tflite_file_name.replace('/', '_').replace('.', '_')
    shell_run_command(command=f"sed -i 's/'{replace_text}'/g_model/g' {model_file_name}")
```

```python
@flow
def firmware_build(mbed_target: str, tflite_file = None) -> None:
    clone_firmware_repository(repo=firmware_repository)
    mbed_config_root()
    mbed_deploy()
    get_model_file(tflite_file)
    prepare_model(tflite_file)
    mbed_compile(target=mbed_target, compiler=mbed_compiler)
    save_firmware(s3_bucket_name=bucket_name, s3_folder=s3_folder, file_path=full_bin_path,
                  mbed_target=mbed_target)
    cleanup()
```

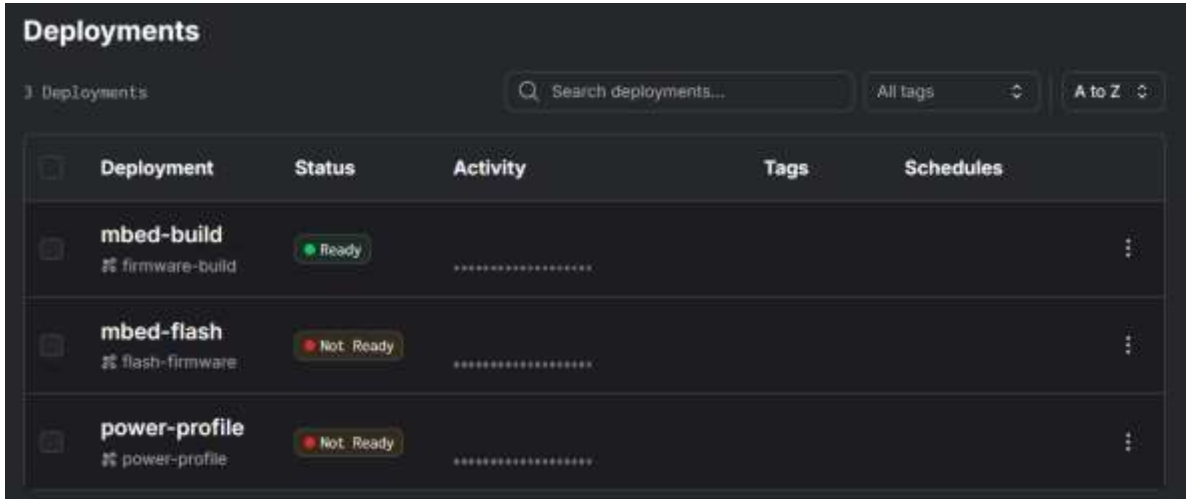
Prefect Runs

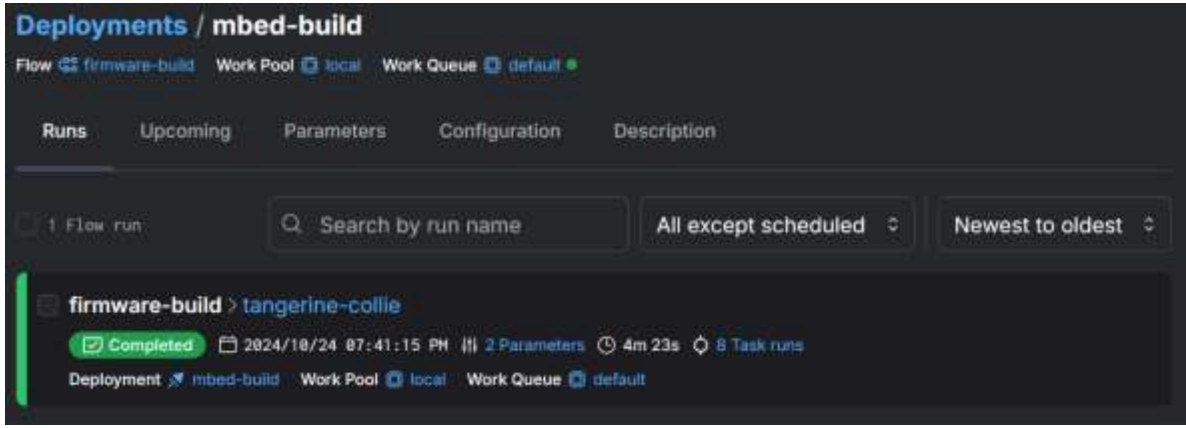
- Represents each **instance of a workflow execution**.
- Tracks **status**, **logs**, and **outputs** for real-time monitoring.
- Enables **issue identification** with success, failure, and retry info.



Prefect Deployments

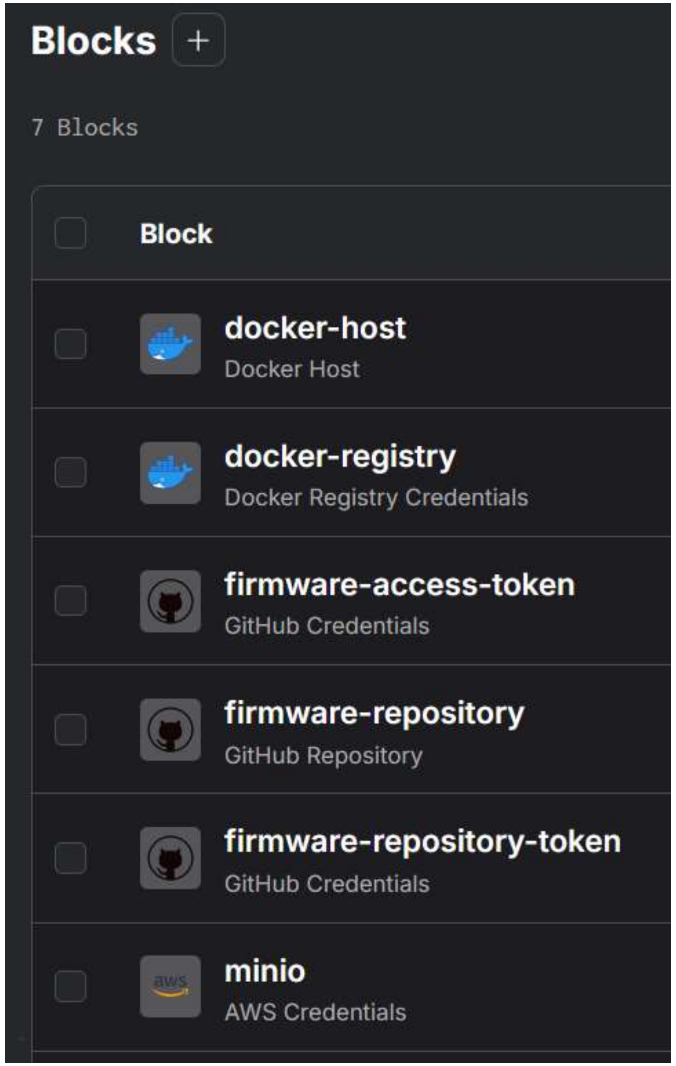
- ***Automated, scalable workflow deployments*** designed for production-ready pipelines.
- **Schedule & Trigger Workflows**: Set up automatic or event-based triggers for reliable execution.
- **Parameterization**: Customize workflows with dynamic inputs to adapt to changing data and conditions.
- **Version Control**: Track and roll back to specific versions for easy updates and maintenance.





Prefect Blocks

- ***Reusable, modular building blocks*** that simplify the development, deployment, and management of workflows.
- **Pre-built integrations**: Connect to popular tools (databases, cloud providers, messaging services) without custom code.
- **Customizable**: Create and share custom blocks tailored to unique needs.
- **Easy configuration**: Define and reuse configurations across tasks to reduce setup time.

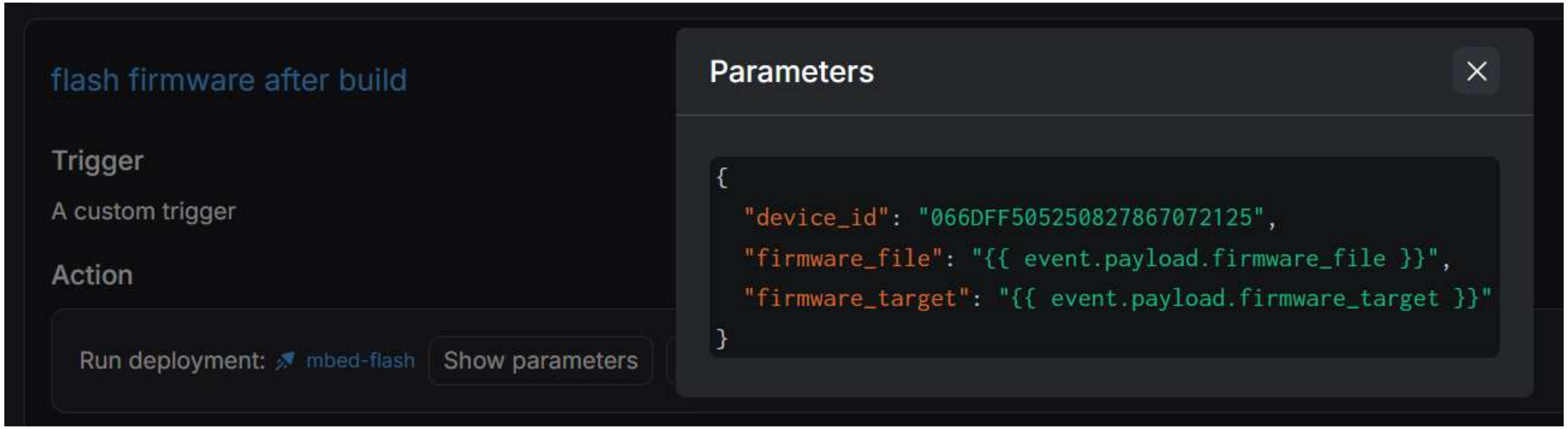


Prefect Variables

- ***Dynamic configuration elements*** used to streamline and adapt workflows.
- **Centralized Configuration**: Manage settings and constants across flows from a single place.
- **Environment-Aware**: Adjust workflow behavior based on environments (dev, staging, production).
- **Reusable & Secure**: Define once, reuse everywhere; securely stored to protect sensitive information.

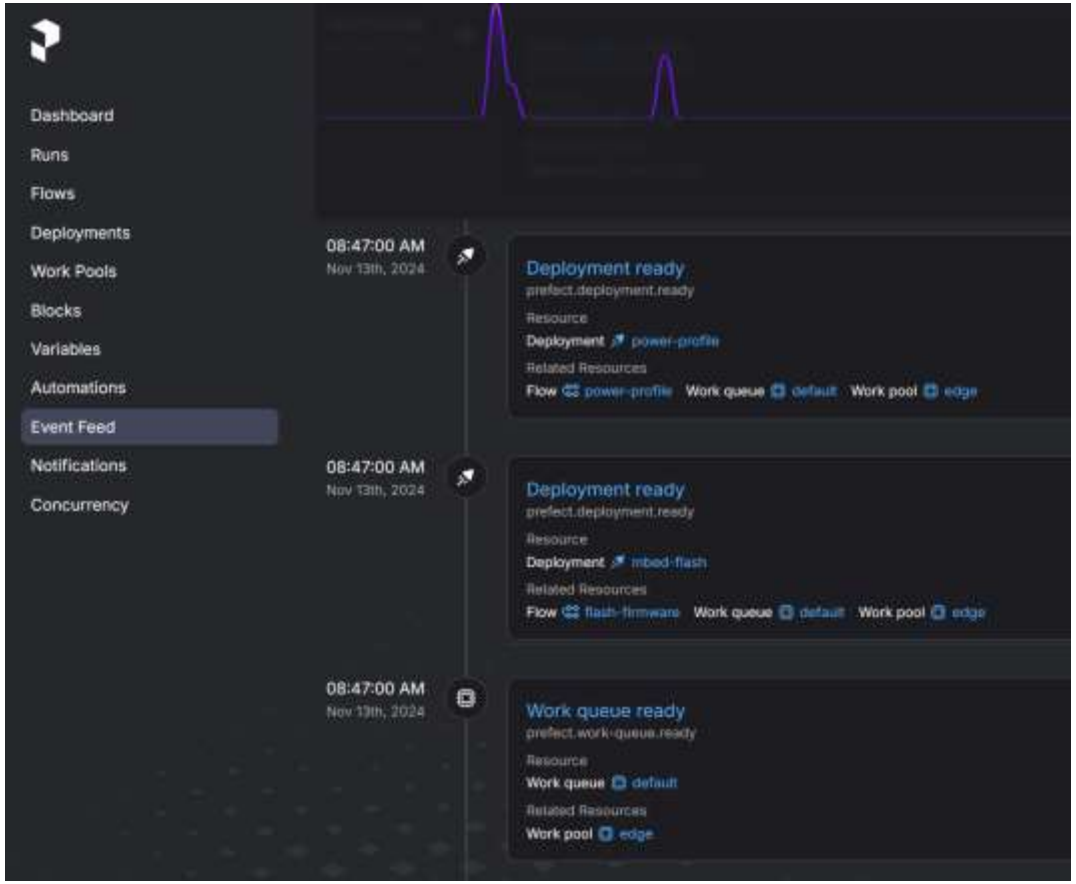
Prefect Automations

- **Automated Actions**: Trigger responses based on events (e.g., failures, successes).
- No-Code Setup: Easily configure alerts, retries, or escalations without custom code.
- **Improves Reliability**: Reduces manual intervention and ensures timely responses.
- **Customizable**: Tailor actions to specific workflows and business needs.



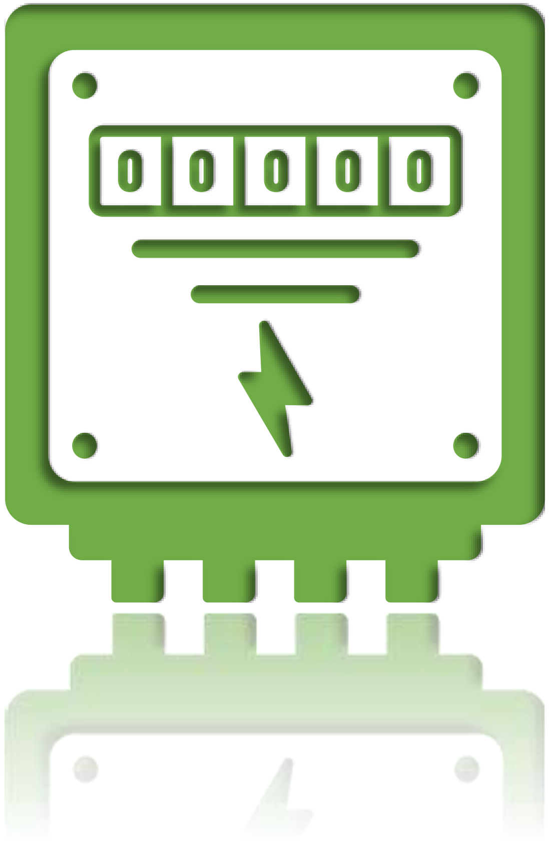
Prefect Event feed & Notifications

- **Real-time updates** on workflow events (starts, completions, failures).
- **Centralized view** for monitoring all activity.
- Quick **identification of issues**.
- **Alerts** for key events (failures, retries) via email, Slack, etc.
- Customizable **triggers** to focus on relevant updates.
- Keeps teams **informed** for proactive responses.



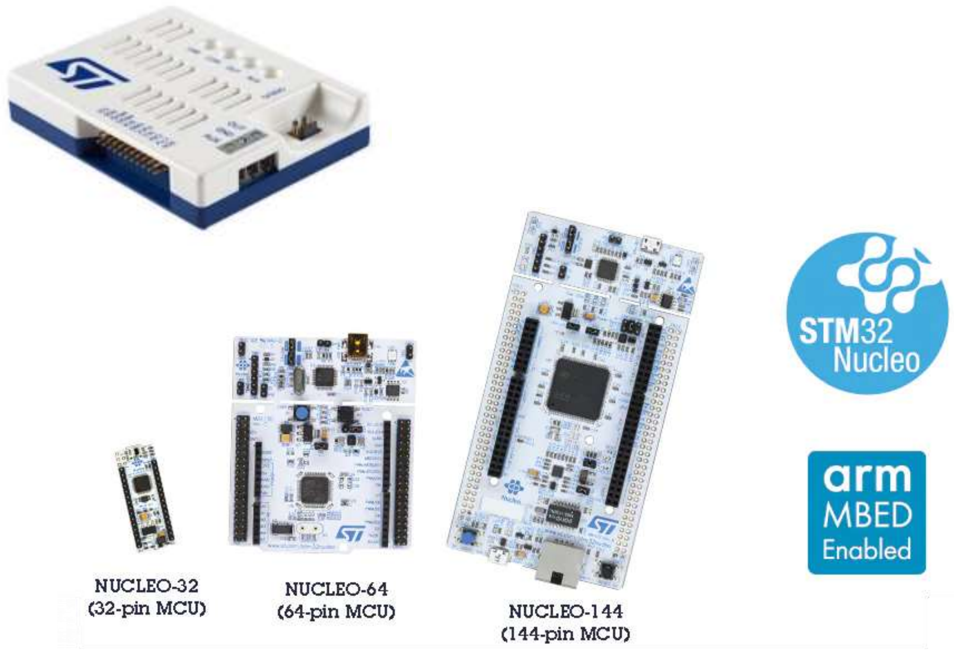
How to use Prefect?

For energy monitoring of AI models on embedded devices

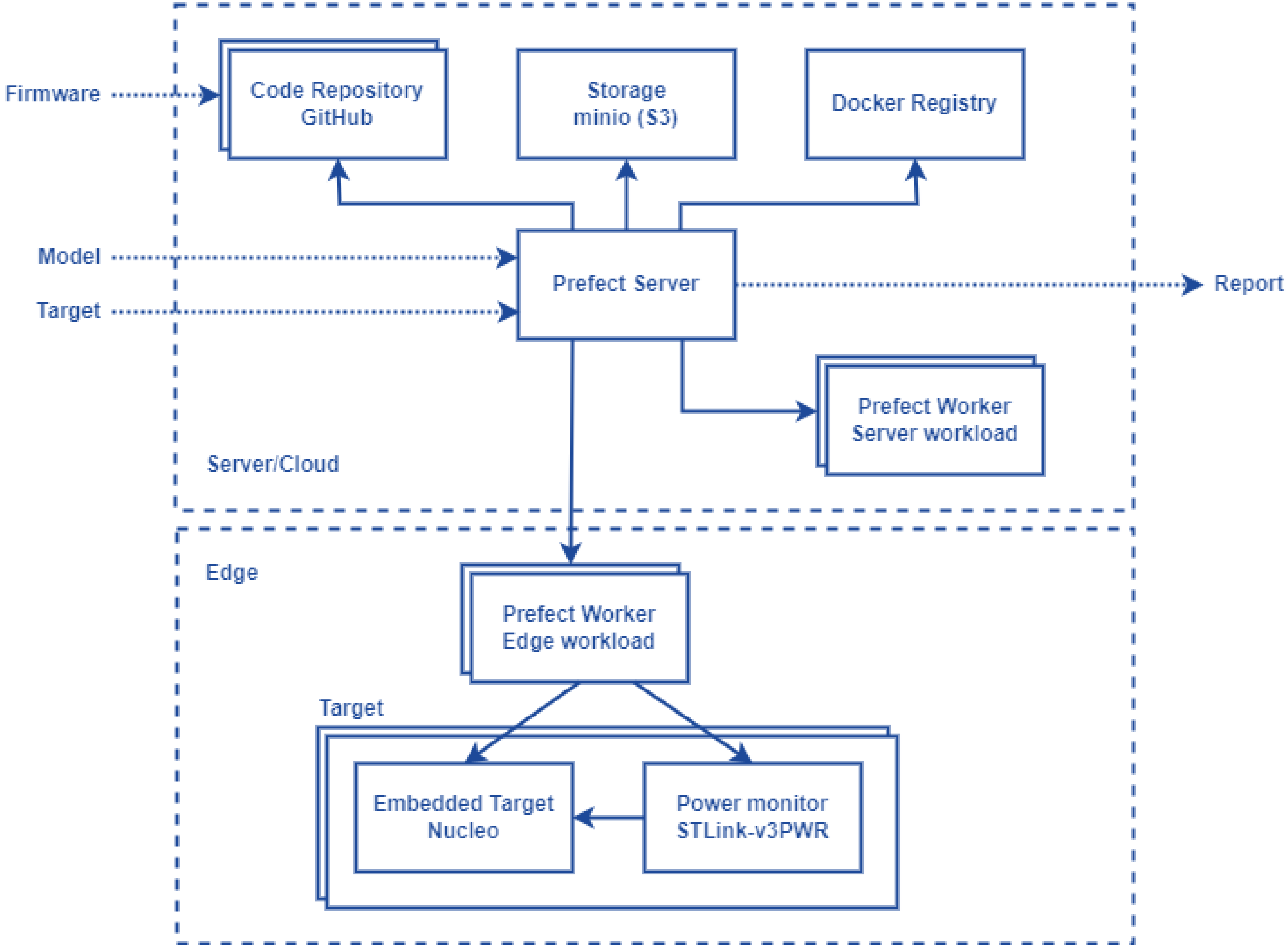


Auto deployment & energy monitoring

- **Version control**: Git(Hub)
- **Model storage**: Minio, S3 compatible
- **Docker image repository**: Docker Registry
- **Task orchestration**: Prefect server
- **Edge nodes**: Prefect workers
- **Targets**: STM32 Nucleo development boards
- **Energy monitoring**: STLink-v3PWR



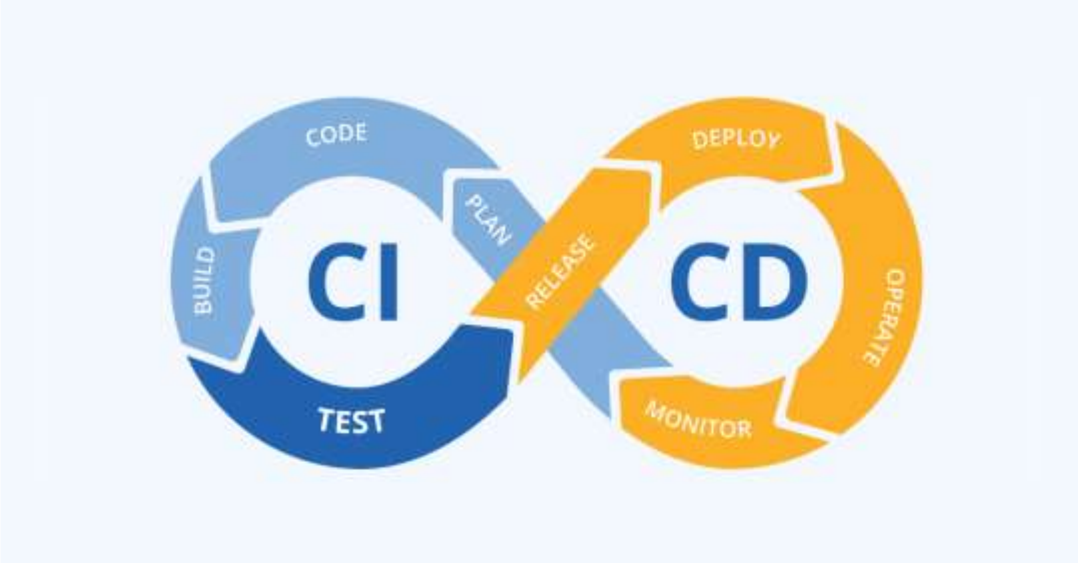
Get target → Build firmware → Get model → Deploy to target → Monitor energy → Make report

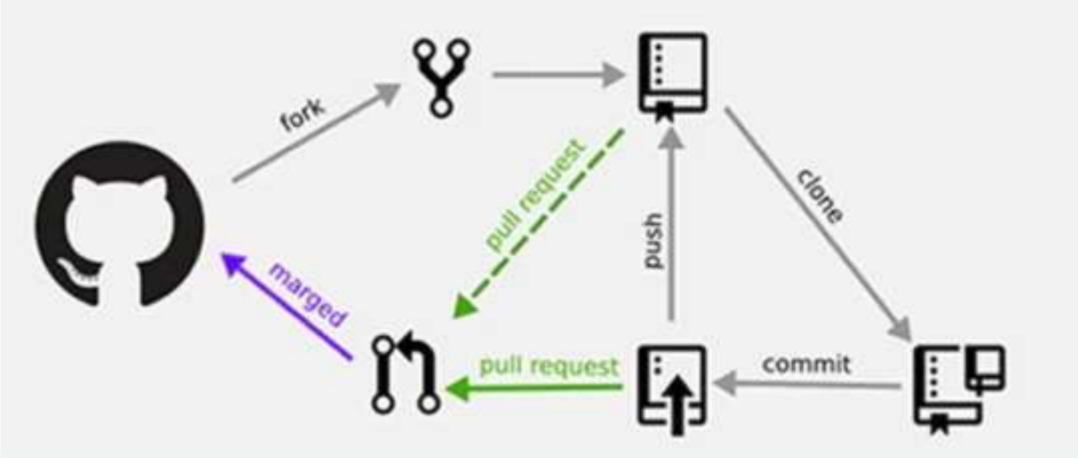


- End-user
  - **Blackbox**
- **Inputs**
  - Firmware
  - Model
  - Target
- **Output**
  - Power report

Version control

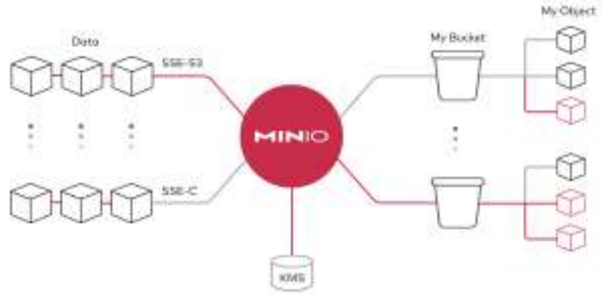
- **Essential in MLOps** for version control, collaboration, and CI/CD, ensuring reproducibility and streamlined deployment of machine learning models.
- **Prefect repostory**
  - Docker compose project configuration
  - Prefect flows and prefect.yaml configuration
- **Firmware repository**
  - Target independent implementation
  - Minimal model to validate the build phase
  - Implementation can be adjusted according to the preferred test functionalities

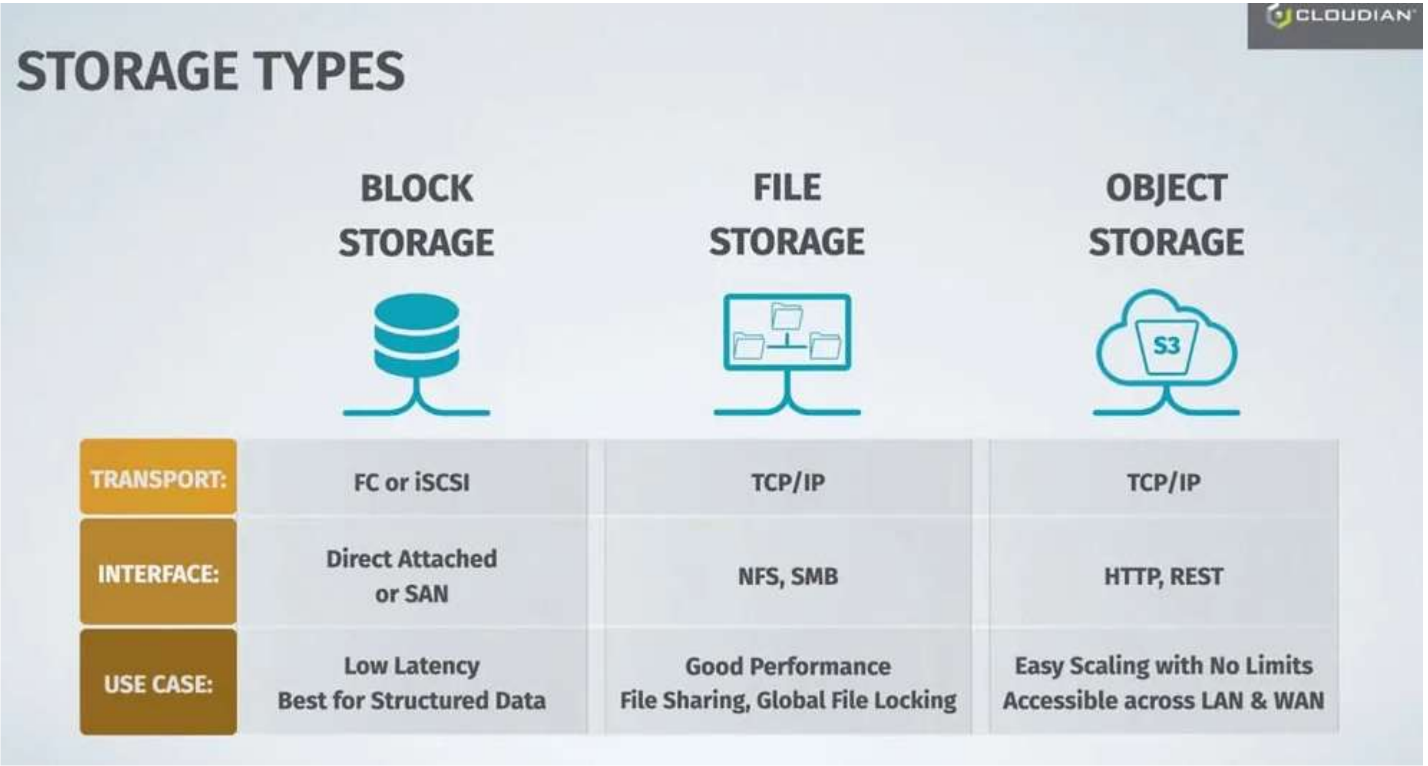




Model storage

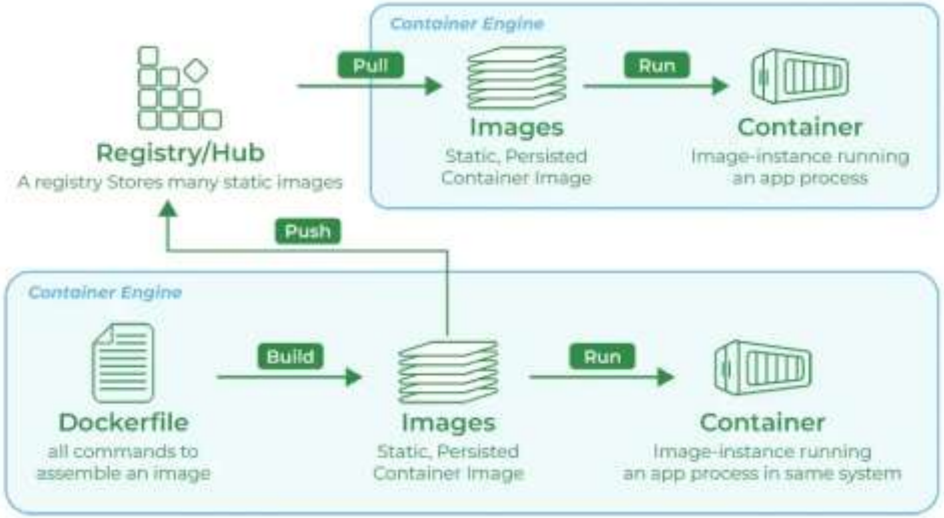
- Minio
- S3 compatibele object storage
- Open source en self-hosted
- **Storage**
  - Models (.tflite files)
  - Firmware builds (.bin files)
  - Reports en results
  - Prefect Deployments (Flows & Tasks)





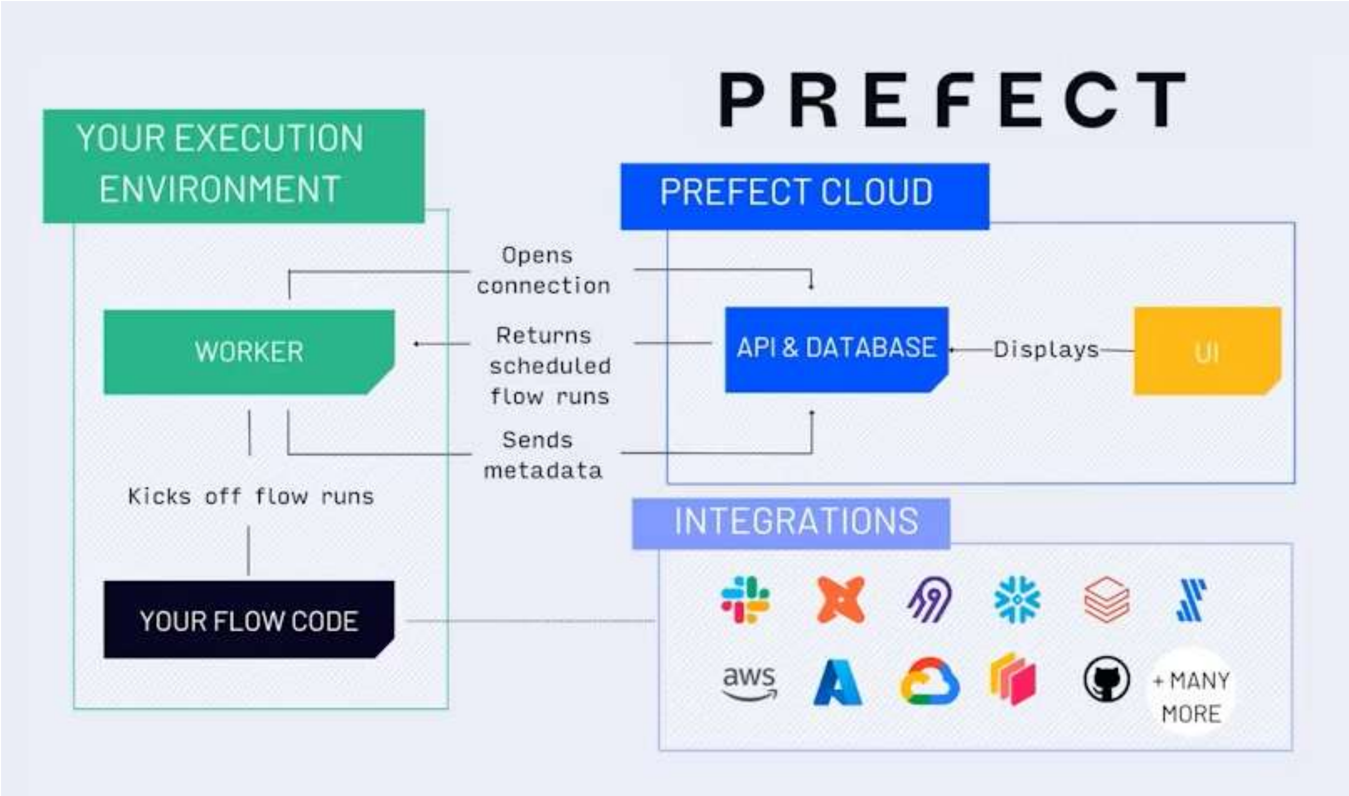
Docker Image management

- Docker Registry
  - **Local Docker Registry** for the management of Docker Images
  - Efficient **(re)use of Docker images**, regardless of whether they need to be executed on the server or edge
  - Docker Image per firmware and target combination
  - Docker Images for complex Prefect Tasks
    - Example: energy monitoring script that can communicate with the target ánd the energy monitoring device



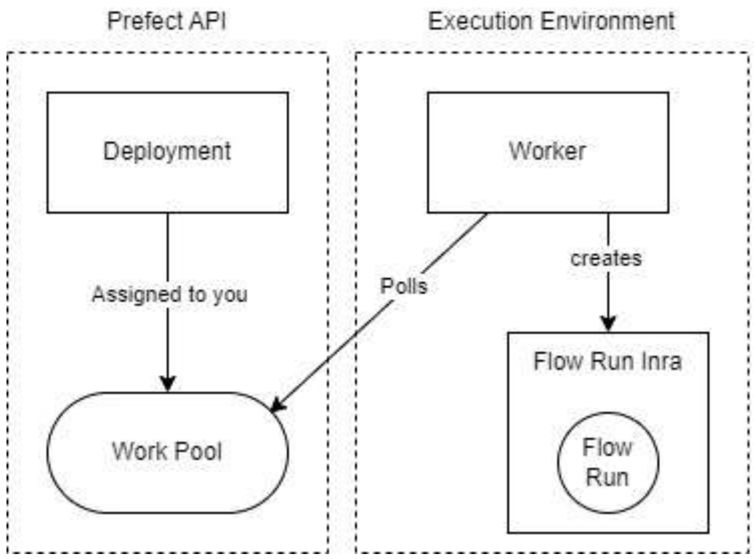
Prefect Server

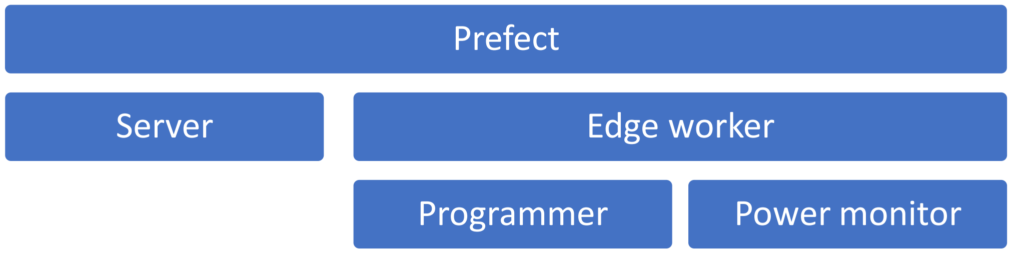
- **Web UI**
- Flow and task manager
  - Flows and tasks are Python scripts
- Review and monitor 'runs'
- **Deployments**
- **Workers** and work pools
- Blocks: Shared services
- Credentials to third party services
- Variables and environments
- Automations



Prefect Worker

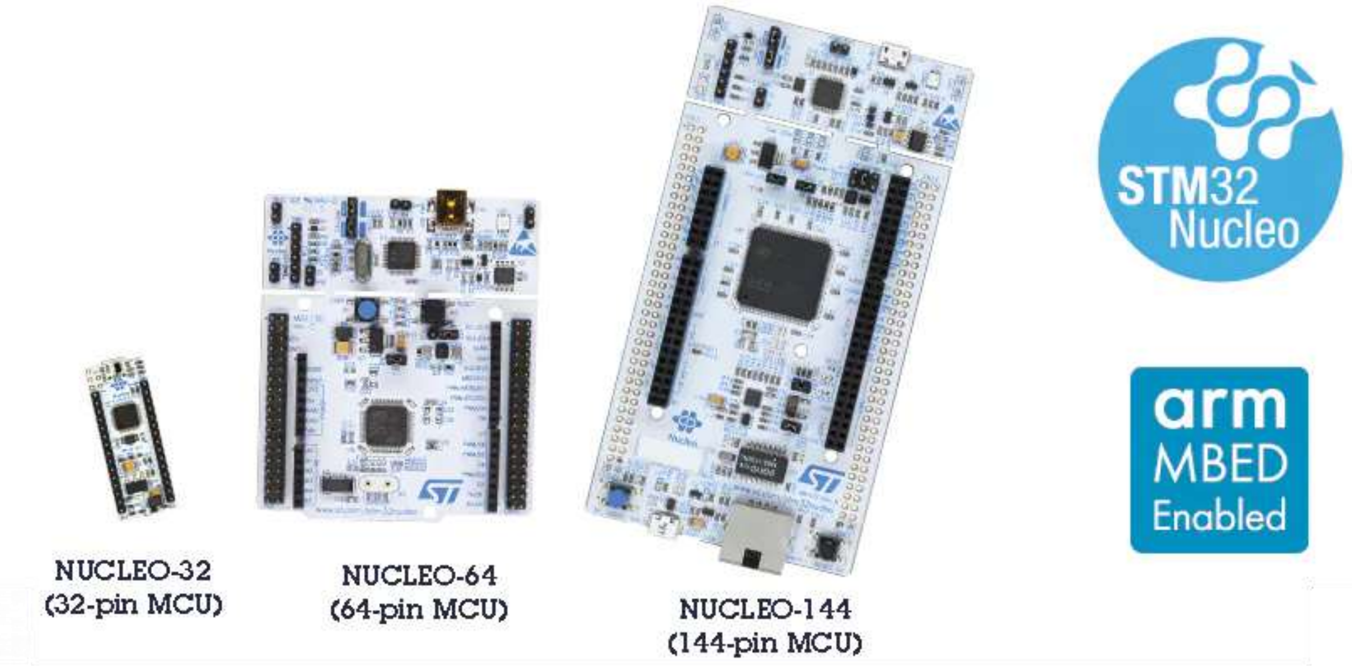
- **Execution enrvironment** for Prefect Flows
- Runs the Python Scripts
  - AWS, Azure and Google Coud containers
  - Kubernetes
  - Process
  - **Docker**
- **Worker pools**
  - Groups of worker
  - Task or **resource oriented**





Targets

- Nucleo target development boards
  - **Microcontrollers**
  - Microcontroller architectures Cortex-M
  - Memory **constraints**
    - RAM
    - Flash
  - **Peripherals**
- USB Connections to edge computer
  - UART comunication

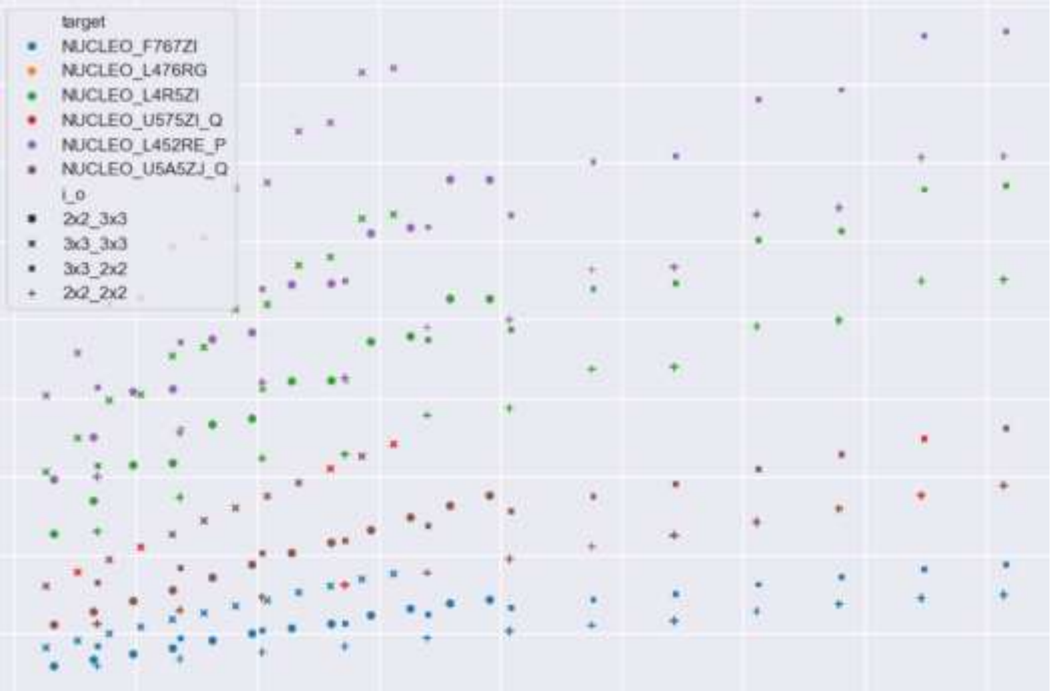


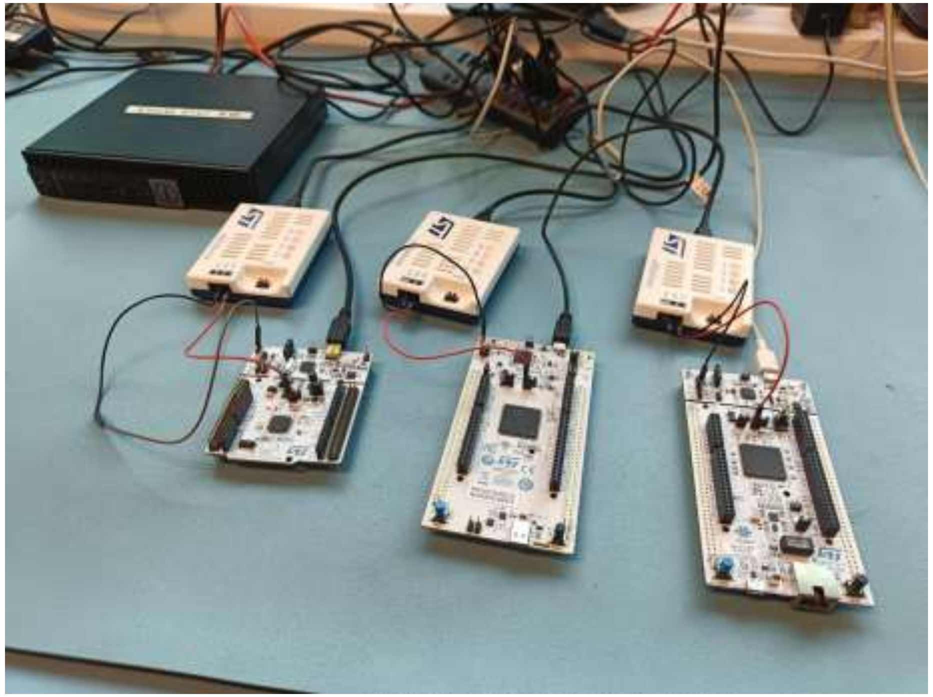
Power monitor

- STLink v3PWR
- Managed over UART
  - **Python script**
  - Start/stop measurements
  - Set power and voltages
  - Reset targets
  - UART for target commands

Reports and logs

- Plain old **Python scripts or Jupyter Notebook** with matplotlib, pandas,… decorated with Prefect Tasks and Flows





Volgende stappen

Planning volgende periode

1. Huidige resultaten verder documenteren en rapporteren (website)
2. Opstart en uitwerken **nieuwe** cases
3. Uitwerking content workshops
4. Communicatie volgende vergadering van de Begeleidingsgroep

Case studies – Call to action!

Belangrijke overwegingen:

- Beschikbaarheid van <u>geannoteerde data en modellen</u>
- Bereidheid tot <u>actieve participatie vanuit het bedrijf</u>
- Weinig tot geen beperkingen inzake <u>confidentialiteit</u>

Gelieve in het feedbackformulier voorstellen te plaatsen.

Of spreek ons aan tijdens de receptie!

Workshops – Save the dates!

- Introductie tot management van AI in operations
  - Leren werken met de tools (vooraf geïnstalleerd)
  - Data versioning, model tracking, monitoring

4 – 11 – 18 februari 2025

- Hands-on workshop met MLOps tools
  - De tools opzetten & configureren
  - DevOps for ML: CI/CD, docker, kubernetes

- Implementatie en monitoring op edge devices
  - Hardware platformen
  - Deployment, upgrade & monitoring

Seminariedag MLOps4ECM

- Publiek event, najaar 2025
  - Gratis voor leden
  - Betalen voor externen
- Sprekers mogen zich aanmelden!
  - Beckhoff
  - Siemens
  - CTRL Engineering
  - Vintecc
  - Superlinear
  - Yazzoom

Volgende gebruikersgroep vergadering

- Timing
  - April – May 2025
- Locatie
  - In jouw onderneming?
  - Andere interessante locaties?
  - Suggesties?

Praktische toepassingen van IoT, AI, Robotica en Digital Twin voor de KMO

- Interreg Art-IE: AI Robotica lab
- Datum: Dinsdag 26 november
- Locatie: VIVES Campus Kortrijk, The Cube
- Deadline inschrijven: 20 november

Programma

16:30 - 17:40: AI en Robotica voor de KMO

17:40 - 18:00: Pauze met Koffie en Thee

18:00 - 19:00: Digital Twin en IOT

19:00 - 21:00: Netwerkmoment, Open Lab en Diner

Inspiratiesessie ‘Succesverhalen digitalisering voor kmo’s’

FABRIEKEN VOOR DE TOEKOMST

- Datum: Donderdag 28 november
- Locatie: House of Manufacturing, Kortrijk
- Inschrijven

Programma

16.00u.	Onthaal	
16.30u.	Verwelkoming & voorstelling inspirerende cases	
18.00u.	Surveillance Art, Dying Phones and Fake Likes	*Dries Depoorter, AI-trendwatcher*
18.30u.	Netwerkreceptie & demo’s	

Medegefinancierd door de Europese Unie

Feedback - vragenlijst

Beschikbaar via

https://forms.office.com/e/CncV1zFdc0

Ook via https://mlops4ecm.be

Login: mlops

Password: 4ecm

VLAIO TETRA

Machine Learning Operations for Edge Condition Monitoring (MLOps4ECM)

Tussentijdse vergadering 14/11/2024

Locatie: Marelec Food Technologies

Met steun van

HBC.2023.0062