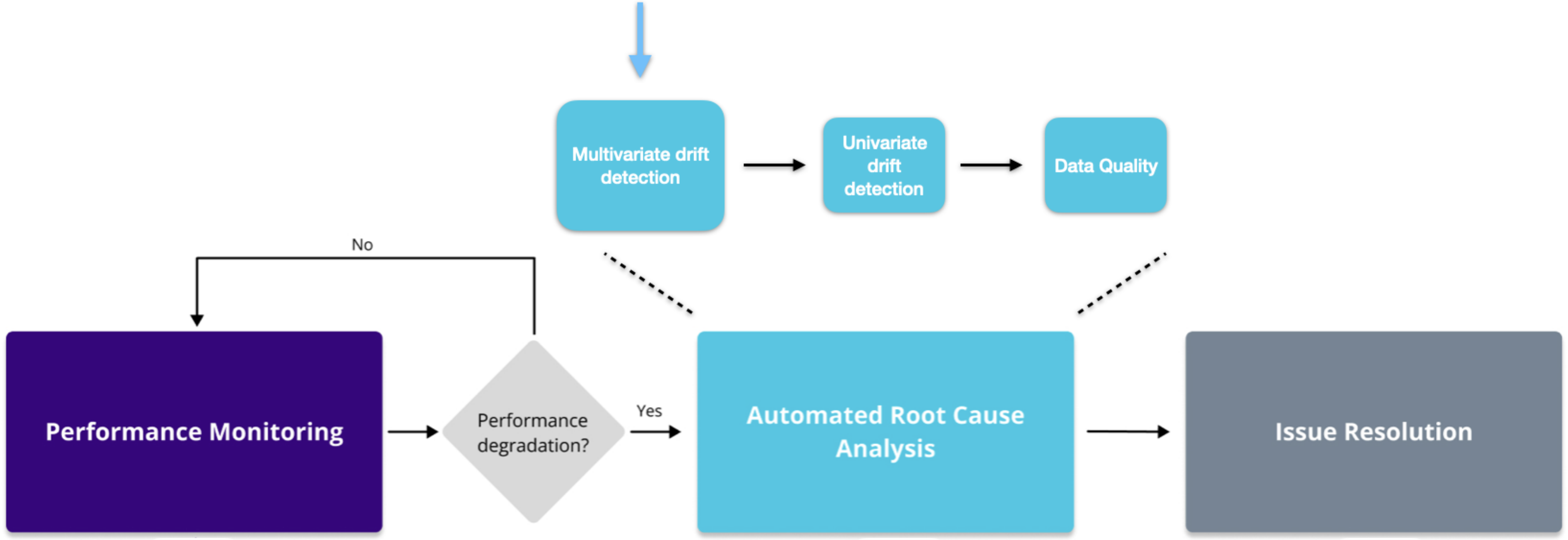
Multivariate drift detection

MONITORING MACHINE LEARNING IN PYTHON

Hakim Elakhrass
CEO and co-founder

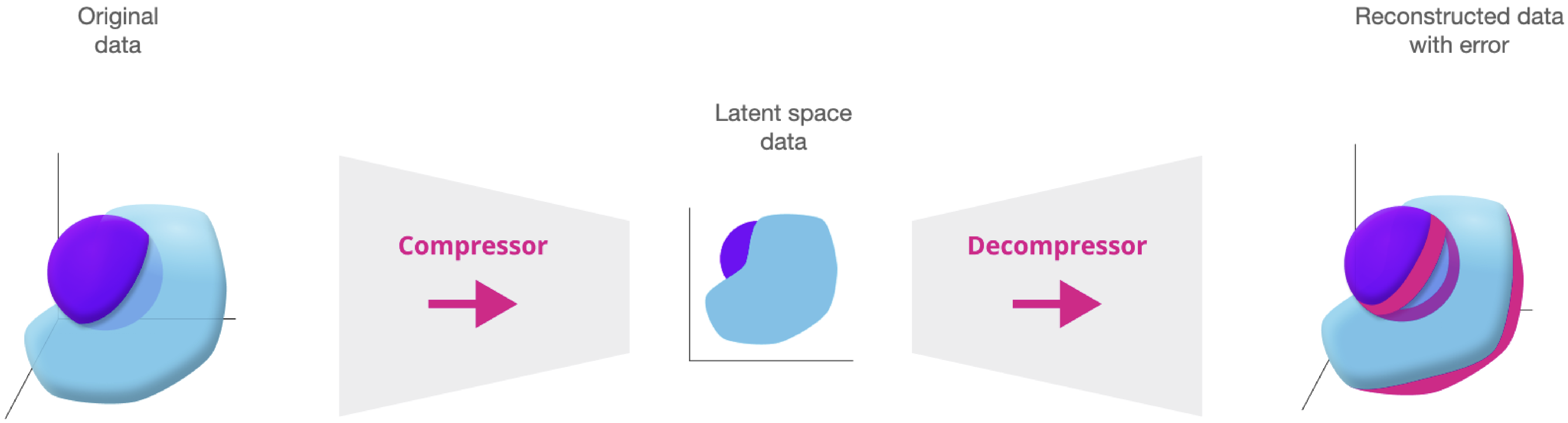
What is multivariate drift detection?

- First step of root cause analysis
- The result is a single number for all features
- Detects subtle data changes



How it works?

1. Compressing the data using PCA algorithm
2. Decompressing the data to initial shape using inverse PCA algorithm
3. Measure the reconstruction error, which increase indicates the data drift



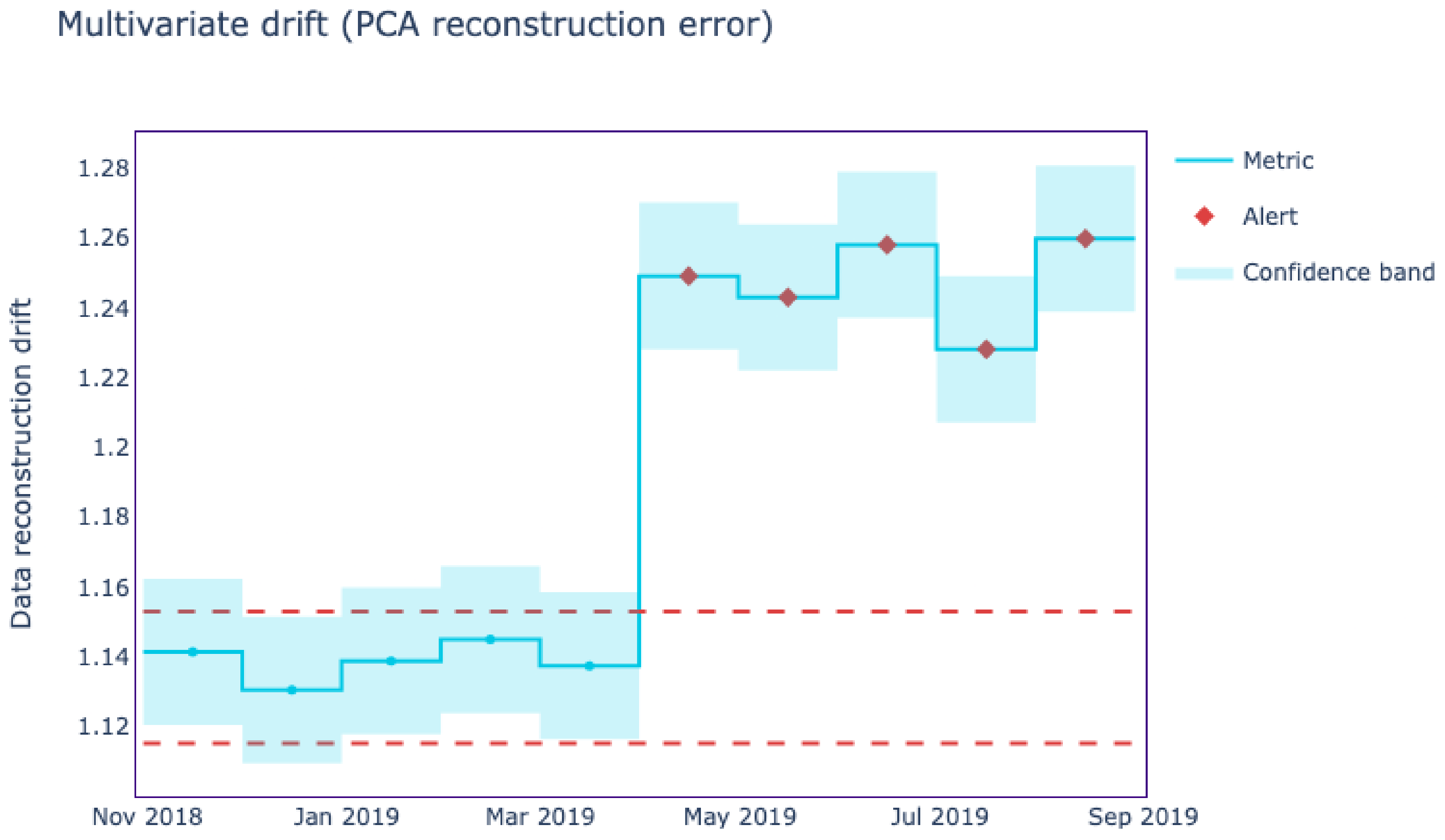
Code implementation

```
# Initialize multivariate drift detection calculator
mv_calc = nannyml.DataReconstructionDriftCalculator(
    column_names=features_column_names,
    timestamp_column_name='timestamp',
    chunk_period='m'
    )
```

```
# Fit and calculate the results
mv_calc.fit(reference)
mv_results = mv_calc.calculate(analysis)
```

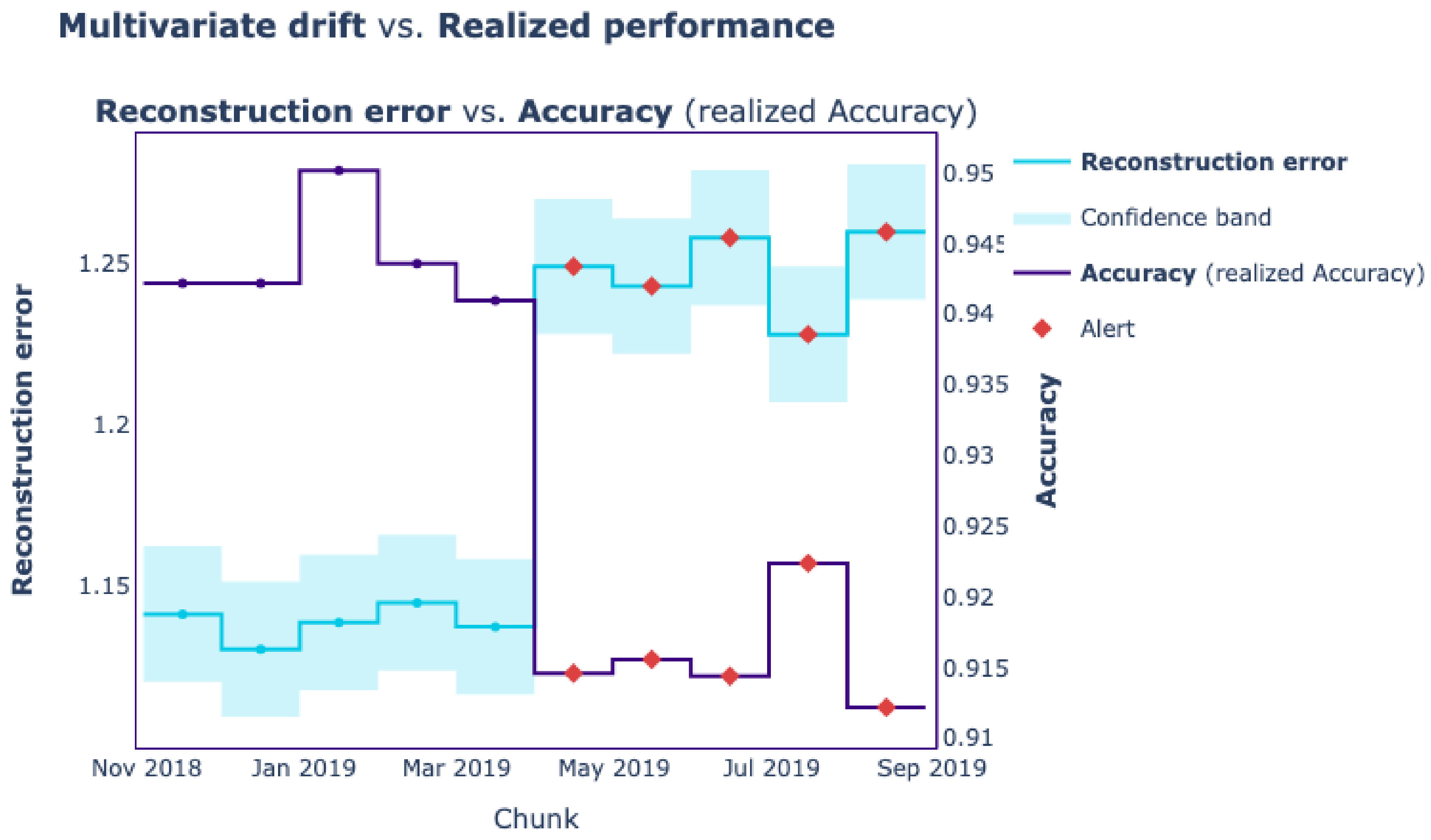
Plotting the results

```
mv_figure = mv_results.filter(period='analysis').plot()
mv_figure.show()
```



Multivariate drift vs. realized performance

```
figure = mv_results.filter(period='analysis').compare(perf_results).plot()
figure.show()
```



Let's practice!

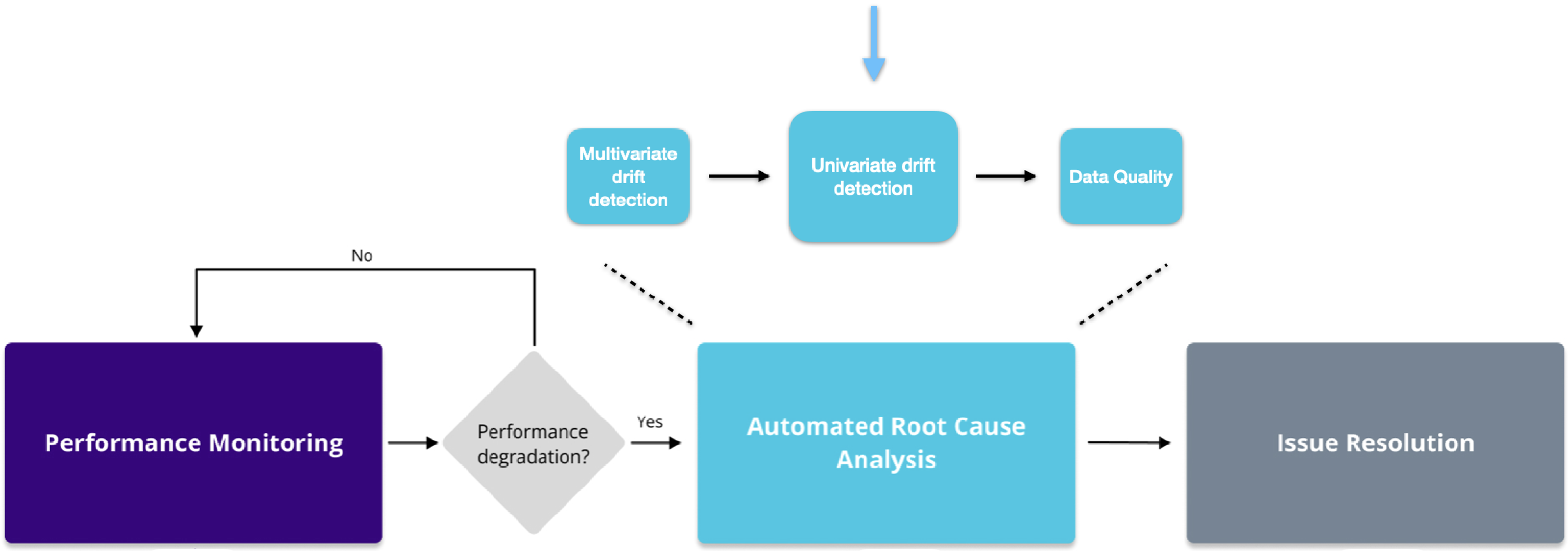
MONITORING MACHINE LEARNING IN PYTHON

Univariate drift detection

MONITORING MACHINE LEARNING IN PYTHON

Hakim Elakhrass
CEO and co-founder

What is univariate drift detection?



Univariate methods

- Jensen-Shannen distance - both categorical and continuous
- Hellinger - categorical and continuous
- Wasserstein - only continuous
- Kolgomorov-Smirnov - only continuous
- L-infinity - only categorical
- Chi2 - only categorical

[1] https://nannyml.readthedocs.io/en/stable/how_it_works/univariate_drift_comparison.html

Code implementation

```
# Intialize the univariate drift calculator
uv_calc = nannyml.UnivariateDriftCalculator(
    continuous_methods=['wasserstein', 'hellinger'],
    categorical_methods=['jensen_shannon', 'l_infinity', 'chi2'],
    column_names=feature_column_names,
    timestamp_column_name='timestamp',
    chunk_period='d'
    )
```

```
# Fit, calculate and plot the results
uv_calc.fit(reference)
uv_results = uv_calc.calculate(analysis)
uv_results.plot().show()
```

Filtering

- Based on the column names
- Based on the univariate methods

```
# Filter the univariate results
filtered_figure = uv_results.filter(column_names=['trip_distance', 'fare_amount'],
            methods=['jensen_shannon'])

# Plot the filtered results
filtered_figure.show().plot()
```

Alert count ranker

- Rank features based on the number of alerts

```
# Initialize the alert count ranker
alert_count_ranker = nannyml.AlertCountRanker()
alert_count_ranked_results = alert_count_ranker.rank(
    uv_results,
    only_drifting=False)
# Display the results
display(alert_count_ranked_results)
```

	number_of_alerts	column_name	rank
0	4	DOLocationID	1
1	3	fare_amount	2
2	1	trip_distance	3
3	1	PULocationID	4

Correlation ranker

- Ranks features based on how much they correlate to absolute changes in performance

```
# Initialize the correlation ranker
correlation_ranker = nannyml.CorrelationRanker()
correlation_ranker.fit(perf_results.filter(period='reference'))
correlation_ranked_results = correlation_ranker.rank(uv_results, perf_results)

# Display the results
display(correlation_ranked_results)
```

	column_name	pearsonr_correlation	pearsonr_pvalue	has_drifted	rank
0	trip_distance	0.736320	0.000041	True	1
1	DOLocationID	0.257138	0.225134	True	2
2	fare_amount	0.193746	0.364340	True	3
3	PULocationID	-0.071132	0.741181	True	4

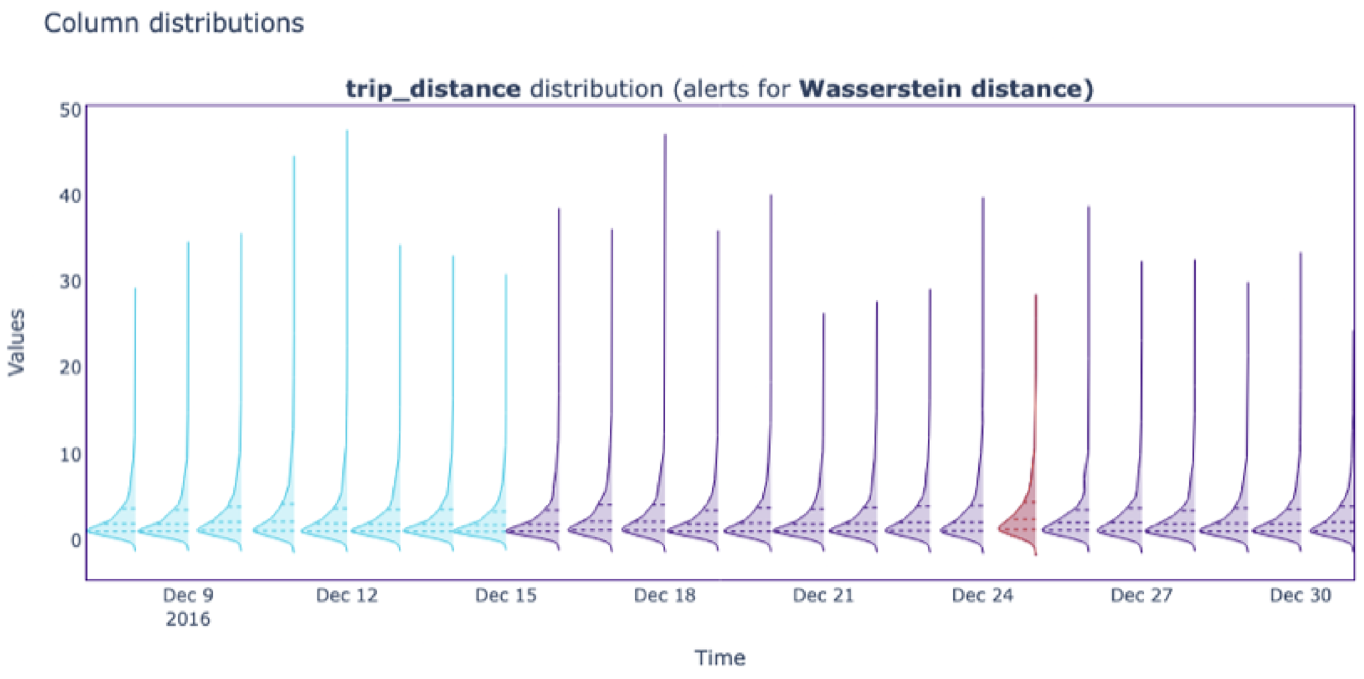
Monitoring feature's distribution

- Gives better insights and improves explainability

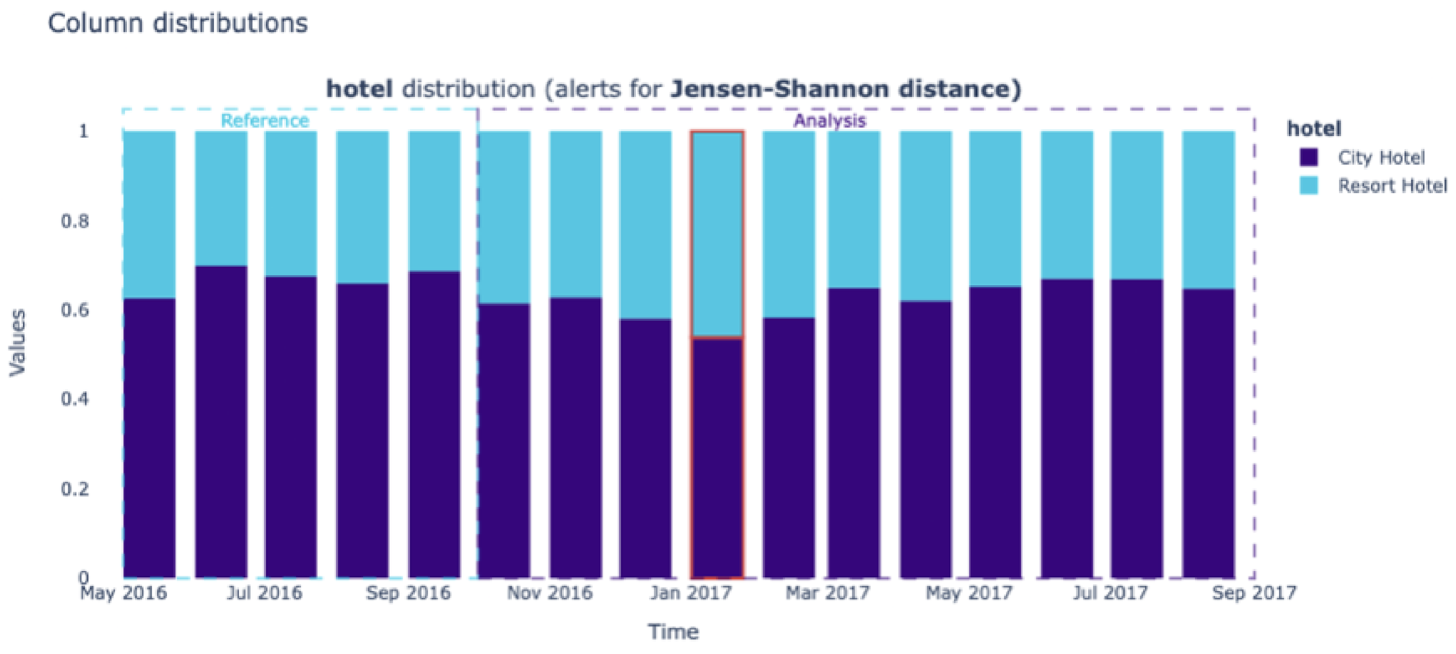
```
# Create distribution plots
distribution_results = uv_results.plot(kind='distribution')

# Show the plots
distribution_results.show()
```

Feature distribution plot

Continuous variable



Categorical variable



Let's practice!

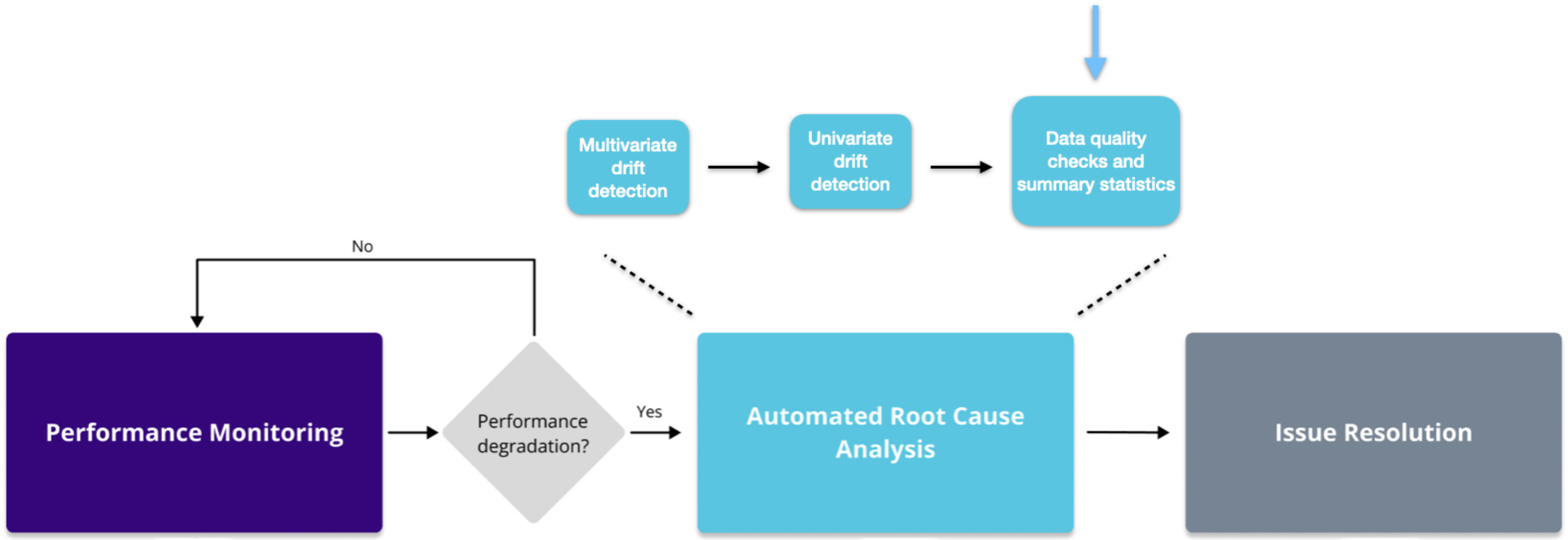
MONITORING MACHINE LEARNING IN PYTHON

Data quality checks and summary statistics

MONITORING MACHINE LEARNING IN PYTHON

Hakim Elakhrass
Co-founder and CEO of NannyML

What are data quality checks and summary statistics?



- Missing value detection
- Unseen value detection
- Summation, average, standard deviation, median and row counts

Missing values detection

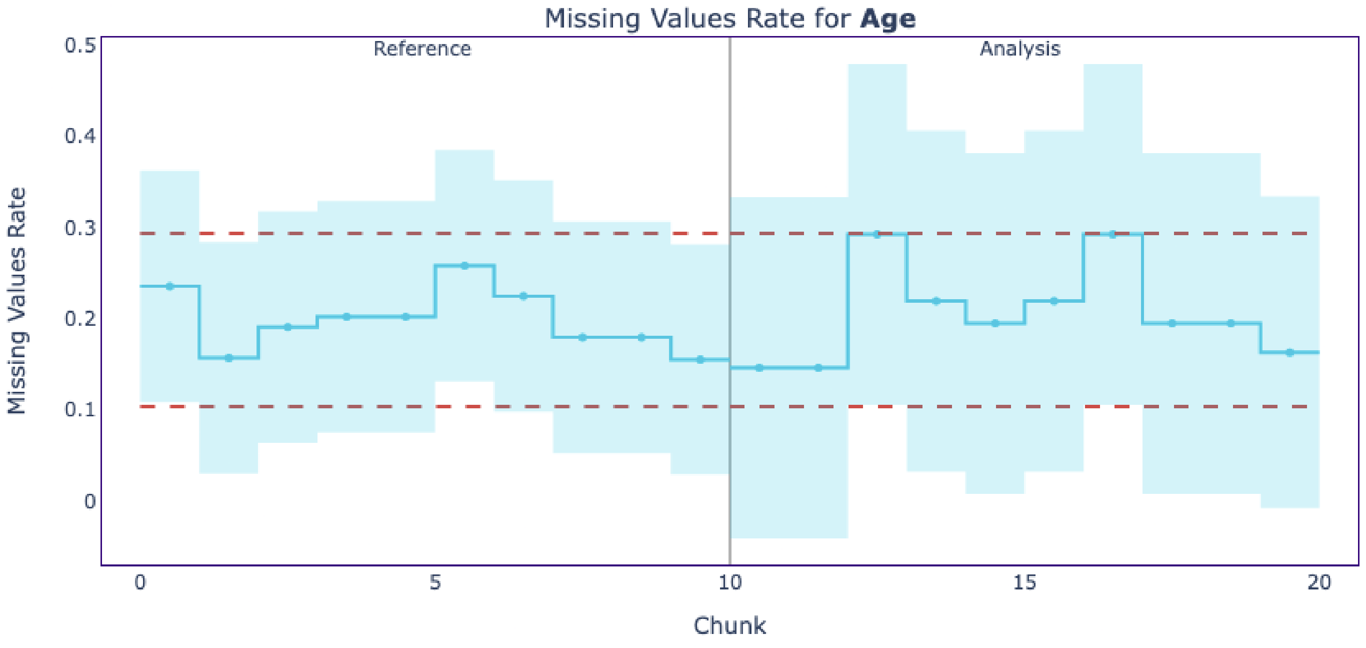
- Reduced observations in a chunk
- Loss in valuable information
- Incorrect interpretations and decisions

```
# Instantiate the missing values calculator module
ms_calc = nannyml.MissingValuesCalculator(column_names=["Age"], normalize=True)

# Fit the calculator on the reference set
ms_calc.fit(reference)

# Calculate the rate of the missing values on the analysis set
ms_results = ms_calc.calculate(analysis)
ms_results.plot()
```

Missing values plot

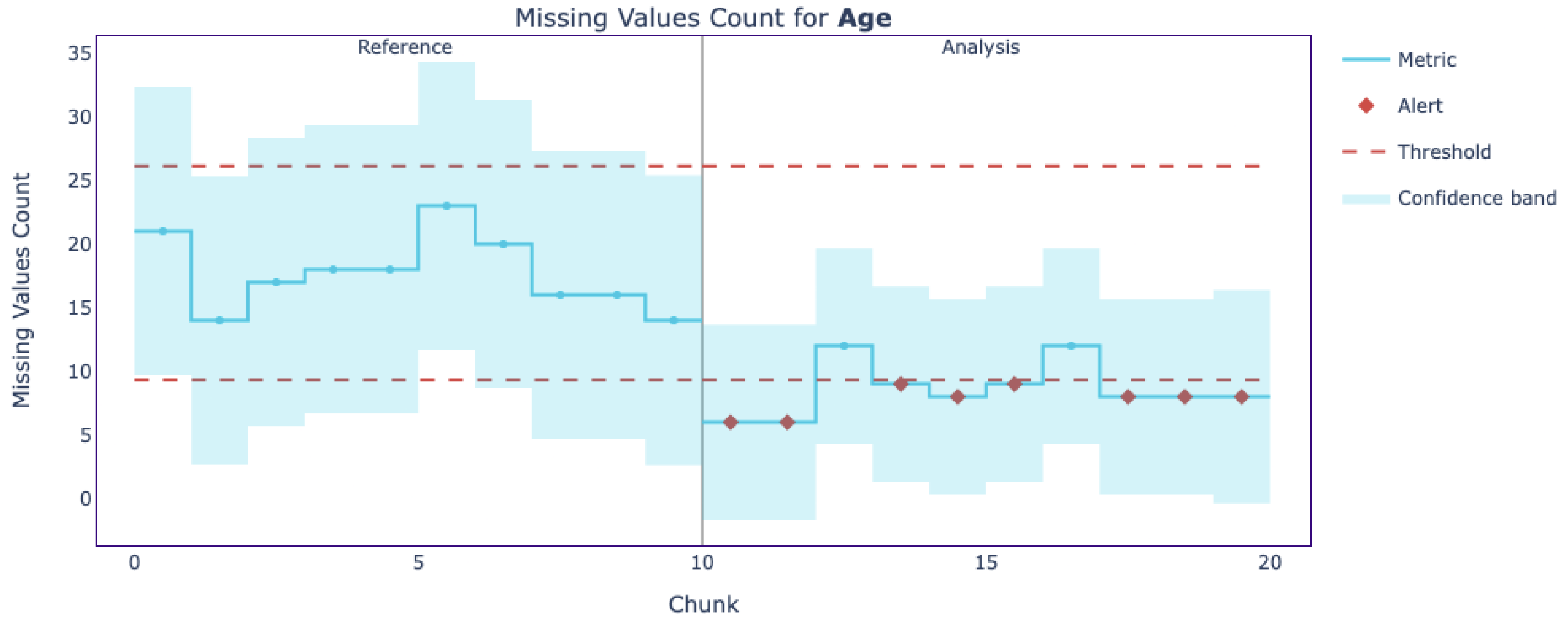
Normalize True

Data Quality



Normalize False

Data Quality

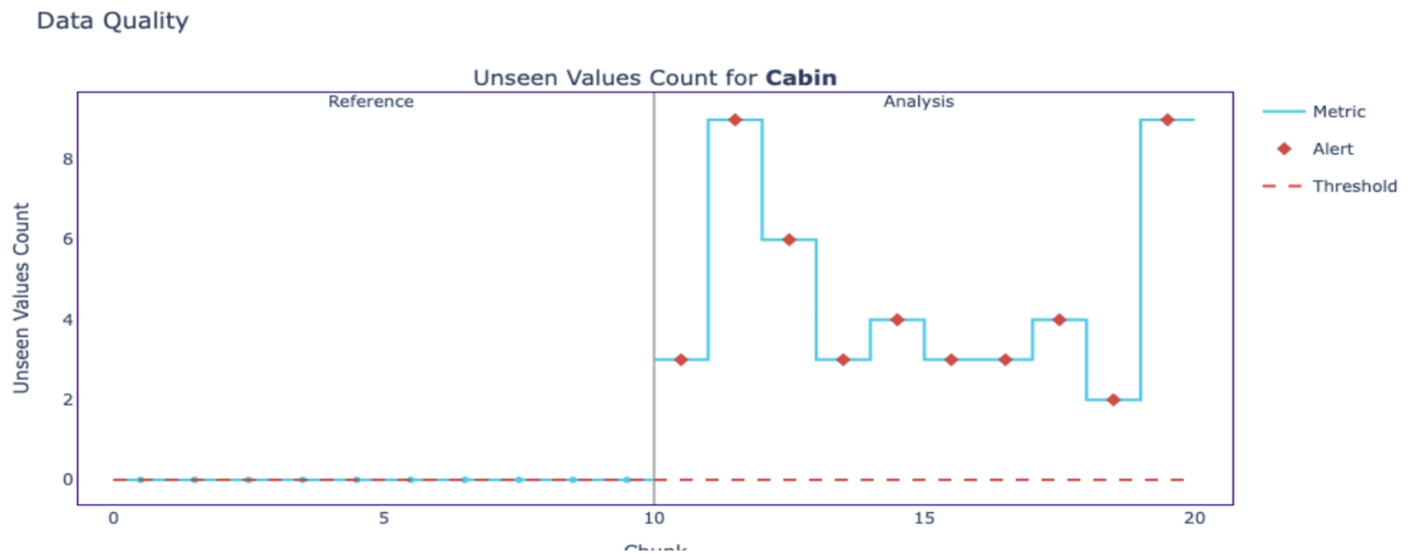


Unseen values detection

- Categorical feature values that are **not present** in the reference period
- An increment of unseen values can make the model less confident in regions

```
# Instantiate the unseen values calculator module
us_calc = nannyml.UnseenValuesCalculator(column_names=["Cabin"], normalize=False)
```

```
# Fit, calculate and plot the rate of the unseen values
us_calc.fit(reference)
us_results = us_calc.calculate(analysis)
us_results.plot()
```



Summary statistics

- **Summation:** Useful for financial data to calculate revenue, or profits for a specific period.
- **Mean** and **Standard Deviation:** Helpful for data drift check and explainability.
- **Median:** Resistant to outliers, making it useful when dealing with features that have many extreme values.
- **Row Counts:** Determine if there is enough data in each chunk.

```
sum_calc = nannyml.SummaryStatsSumCalculator(column_names=selected_columns)
avg_calc = nannyml.SummaryStatsAvgCalculator(column_names=selected_columns)
std_calc = nannyml.SummaryStatsStdCalculator(column_names=selected_columns)
med_calc = nannyml.SummaryStatsMedianCalculator(column_names=selected_columns)
rows_calc = nannyml.SummaryStatsRowCountCalculator(column_names=selected_columns)
```

Let's practice!

MONITORING MACHINE LEARNING IN PYTHON

Issue resolution

MONITORING MACHINE LEARNING IN PYTHON

Hakim Elakhrass
Co-founder and CEO of NannyML

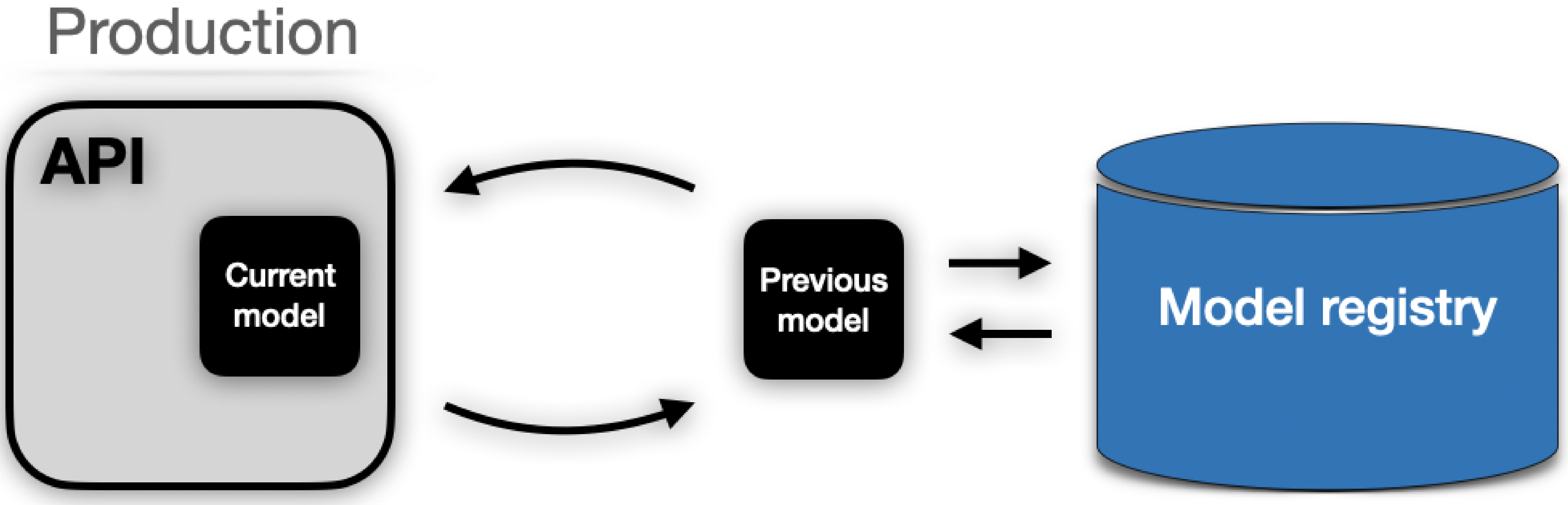
Do nothing

- Works well with up-and-running good monitoring system
- Requires an opportunity cost analysis and good understanding of a use-case
- An example is overestimating the number of calls in call center

Retraining the model

- Train on both old and new data
  - Making the model more robust
  - Learn the model as many as possible distributions
- Fine-tune the old model with the new data
  - Simply refit the model with the new data
  - More effective than training a new model from scratch every time
- Weighting Data
  - Give more importance to the recent data

Reverting back to a previous model



Change business process

- Change the business rules
- Run manual analysis on predictions

Let's practice!

MONITORING MACHINE LEARNING IN PYTHON

Congratulations

MONITORING MACHINE LEARNING IN PYTHON

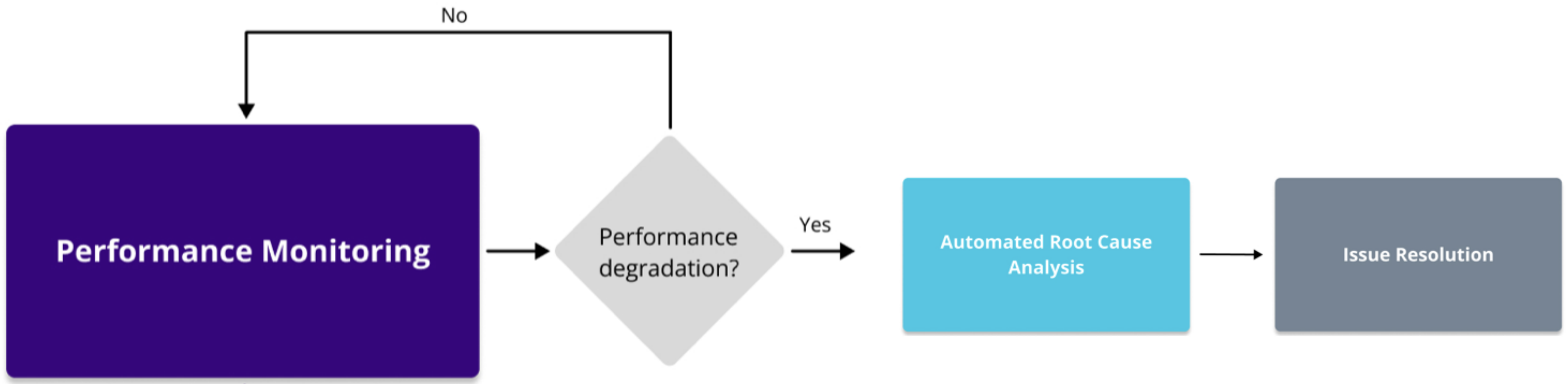
Hakim Elakhrass
Co-founder and CEO of NannyML

Chapter 1 recap

- Fundamentals of NannyML library
- Data preparation process for NYC Green Taxi dataset
- Learn how to estimate the performance using CBPE and DLE

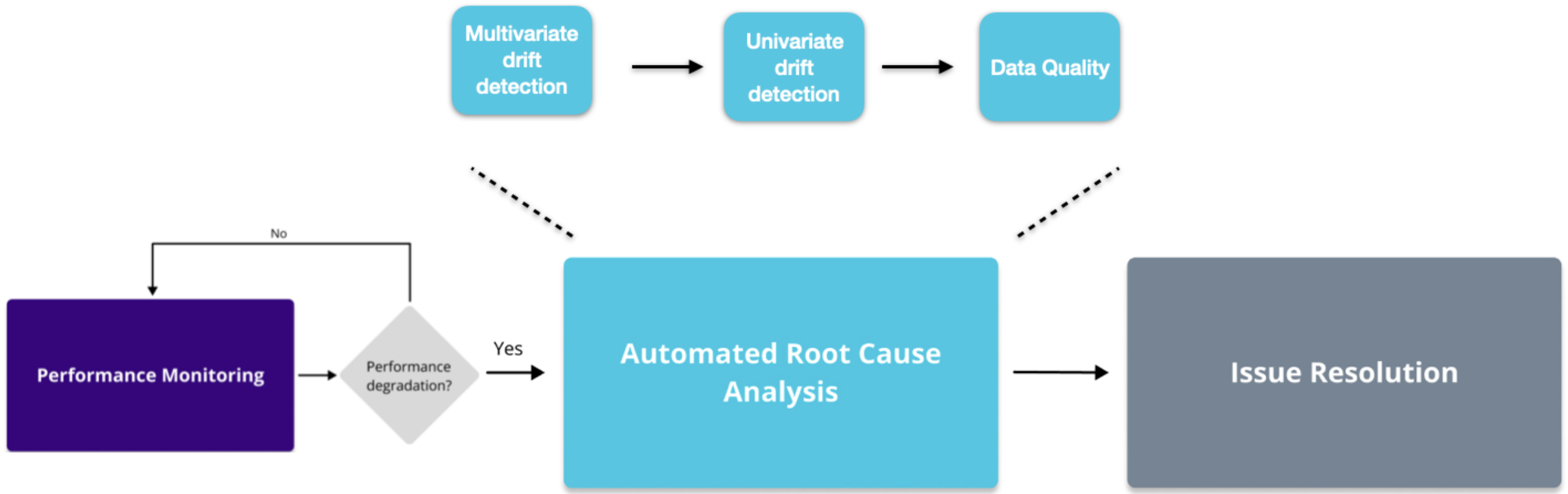
Chapter 2 recap

- Measuring performance when ground truth is available
- Learning how to filter, plot and convert results dataframe format
- Understanding chunking and thresholds
- Calculating and estimating model's business value



Chapter 3 recap

- Performing multivariate drift detection
- Testing various univariate drift detection methods
- Using data quality checks calculators
- Understanding various issue resolution methods



What's next?

- Explore NannyML's blog for tutorials
- Refer to NannyML's documentation for more information
- Consider taking additional courses on machine learning model lifecycle and MLOps
- Experiment with practical projects and incorporate NannyML

Thank you!

MONITORING MACHINE LEARNING IN PYTHON