INFINITYWORKS Part of Accenture

British Cycling Data Platform in Python

Infinity Works at-a-glance

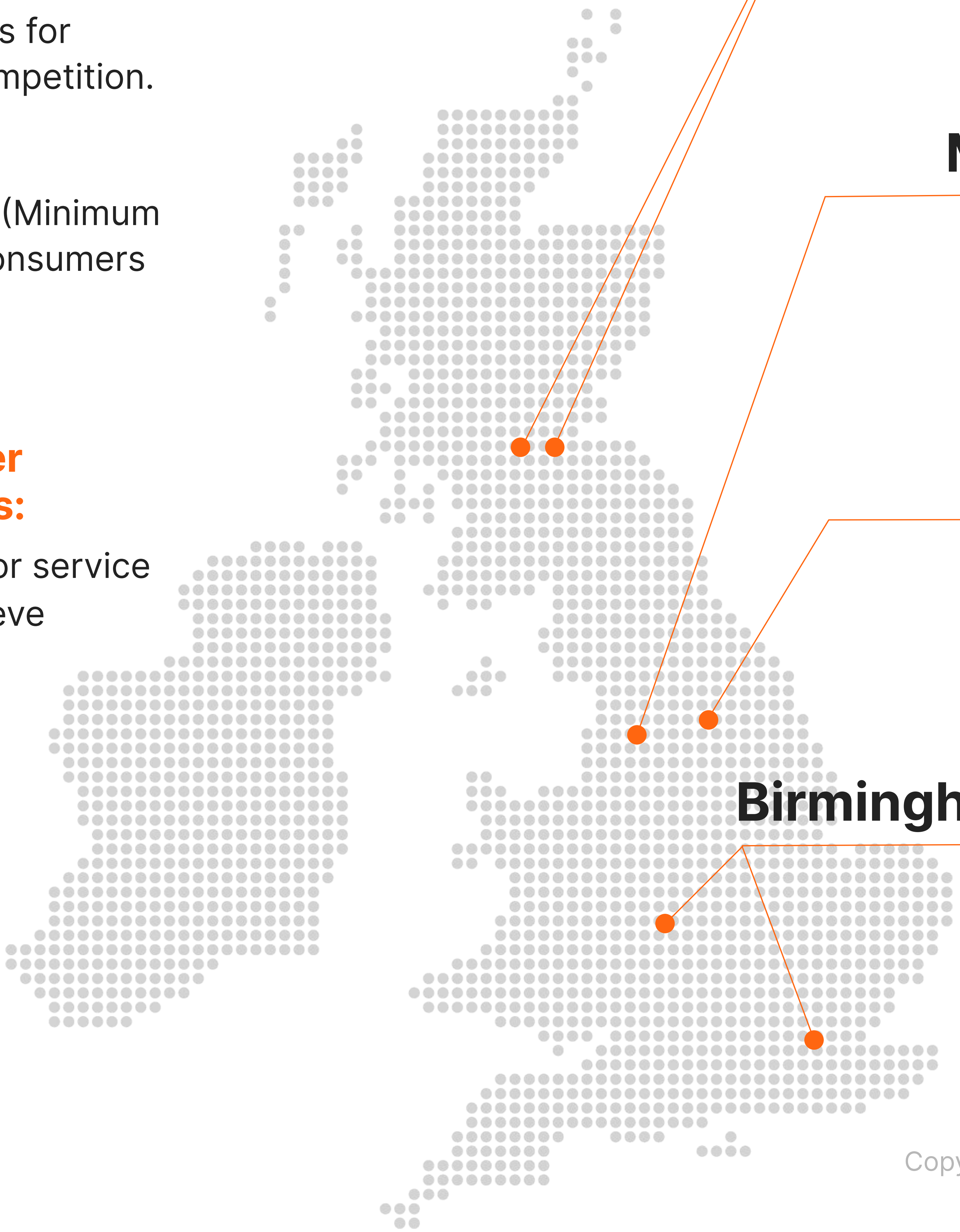
We are unique in: We deliver value fast

If a customer wants to launch new products or services for clients who want to overtake or stay ahead of their competition. **Think Infinity Works**.

We operate in **Days** & **Weeks** to build PoC's and MVPs (Minimum Viable Products) so customers can interact with the consumers at the earliest opportunity.

The customers we work with have a Challenger mindset and have the following characteristics:

- Needs to launch a new, bold and exciting product or service
- Has a strong leadership & vision they want to achieve
- Desire to challenge the market leaders
- Digital-first business
- Financially backed



Commercial Team

Andy Emmett
Head of Alliances

Charles Morgan
Business Development Executive

Edinburgh/Glasgow
80 people

Rory Patience
Client Services Lead

Craig Lumley
Technical Services Lead

Manchester
180 people

Dave Postle
Client Services Lead

Rich Handley
Technical Services Lead

Leeds
240 people

Natalie Lovett
Client Services Lead

Simon Roberts
Technical Services Lead

Birmingham/London
205 people

Elaine Shanks
Client Services Lead

Antony Cox
Technical Services Lead

INFINITYWORKS Part of Accenture

Manchester, UK

Murray Tait

Infinity Works part of Accenture

Profile Overview

- Murray is a Senior Technical Architect, Technical Lead and Delivery Lead with Infinity Works based in Manchester, Uk.
- Murray has 30 years experience in software projects implementation
- He specializes in working with agile development teams using Scrum and Kanban frameworks
- He has been involved in Agile transformation since 2005, by example and though coaching

Recent Experience - Extract

Velocity Labs, Accenture

- Technical Product Owner responsible for productizing the project from proof of value to a production-ready multi-client product.

NHSD Spine

- Technical lead of project create multiple Proof of Concept implementations to move NHSD Spine to a range of Cloud platforms include serverless AWS, AWS RDS and Azure Cosmos
- Technical lead for upgrading Spine from Python2 to Python3

Great Britain Cycling Team

- Architect and delivery lead of project re-platforming on-prem to
- Data processing and analysis tools allow performance analysts to compare multiple sessions or multiple efforts over different periods of time and in various conditions.

Skills

- Delivery management
- Data architecture
- Cloud migration

Vehicle Insurance: retail and wholesale pricing.

Telemetry insurance: data collection, aggregation, analysis and integration.

Fintech: payment gateway provider including functional innovation. Test management and client facing project management. Responsibilities in PCI-DSS compliance

Vehicle pricing and provenance: Wholesale and leasing industry. Consumer facing services

Peter Robinson

Senior Data Engineer
Great Britain Cycling Team

Background and Experience

- MEng from University of Cambridge, including a Masters project working alongside GBCT to develop a collocation based cycling simulation for optimisation problems.
- 2 years working at Softwire Ltd as a full stack developer, including both web and mobile platforms.
- 9 months working as Senior Data Engineer for GBCT, handling the entire data lifecycle from collection through to presentation.

Stefan Krawczyk
CEO DAGWorks

12+ years in ML
& Data platforms

DAGWORKS

STITCH FIX

iDiBON

nextdoor

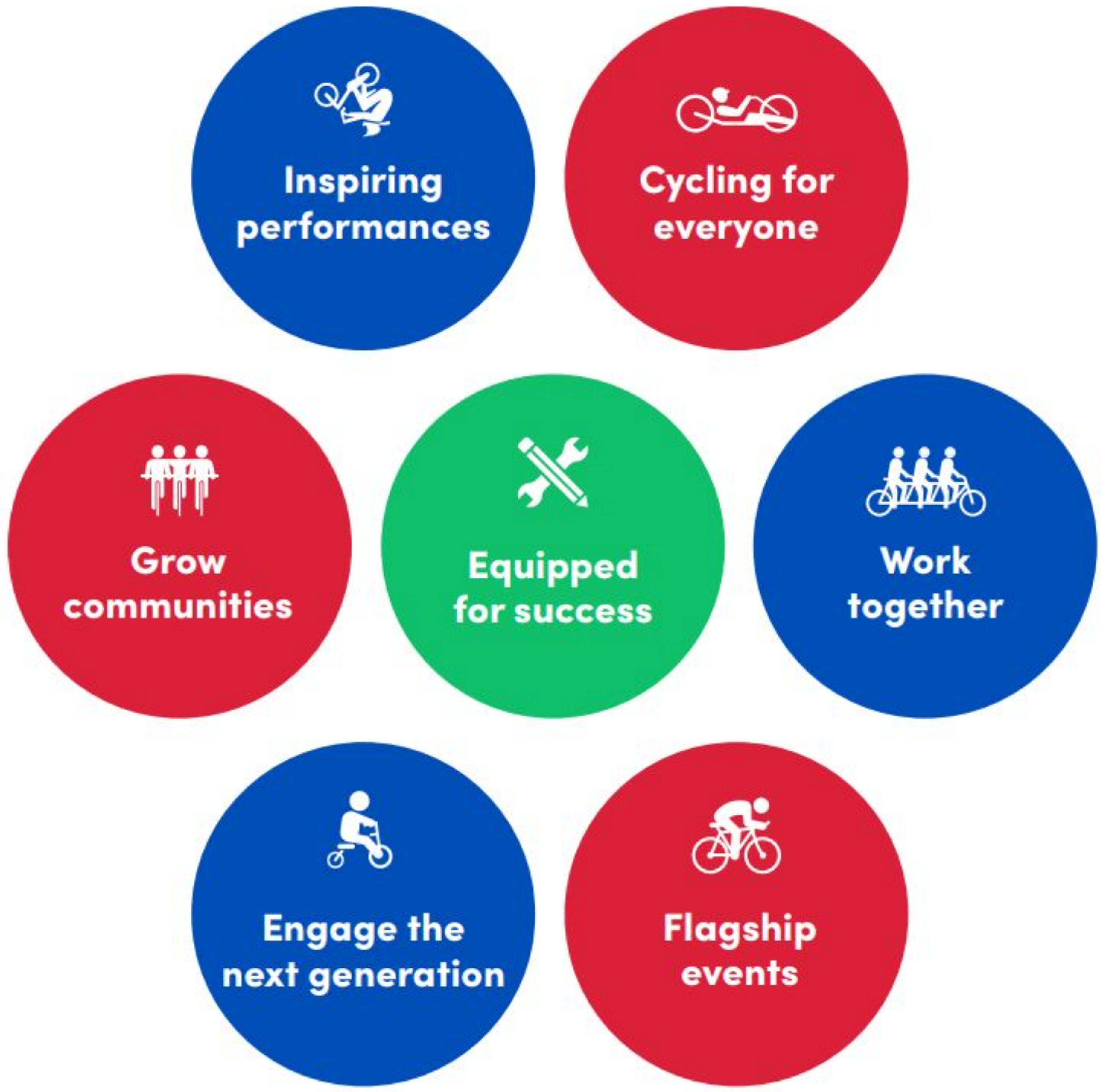
LinkedIn

IBM

Stanford University

VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

BRITISH CYCLING



Win 10 Olympic and 15 Paralympic medals

“The whole principle came from the idea that if you broke down everything you could think of that goes into riding a bike, and then improve it by 1 percent, you will get a significant increase when you put them all together.” - Dave Brailsford, Performance Director of The British Cycling Team

SPORT › OLYMPICS

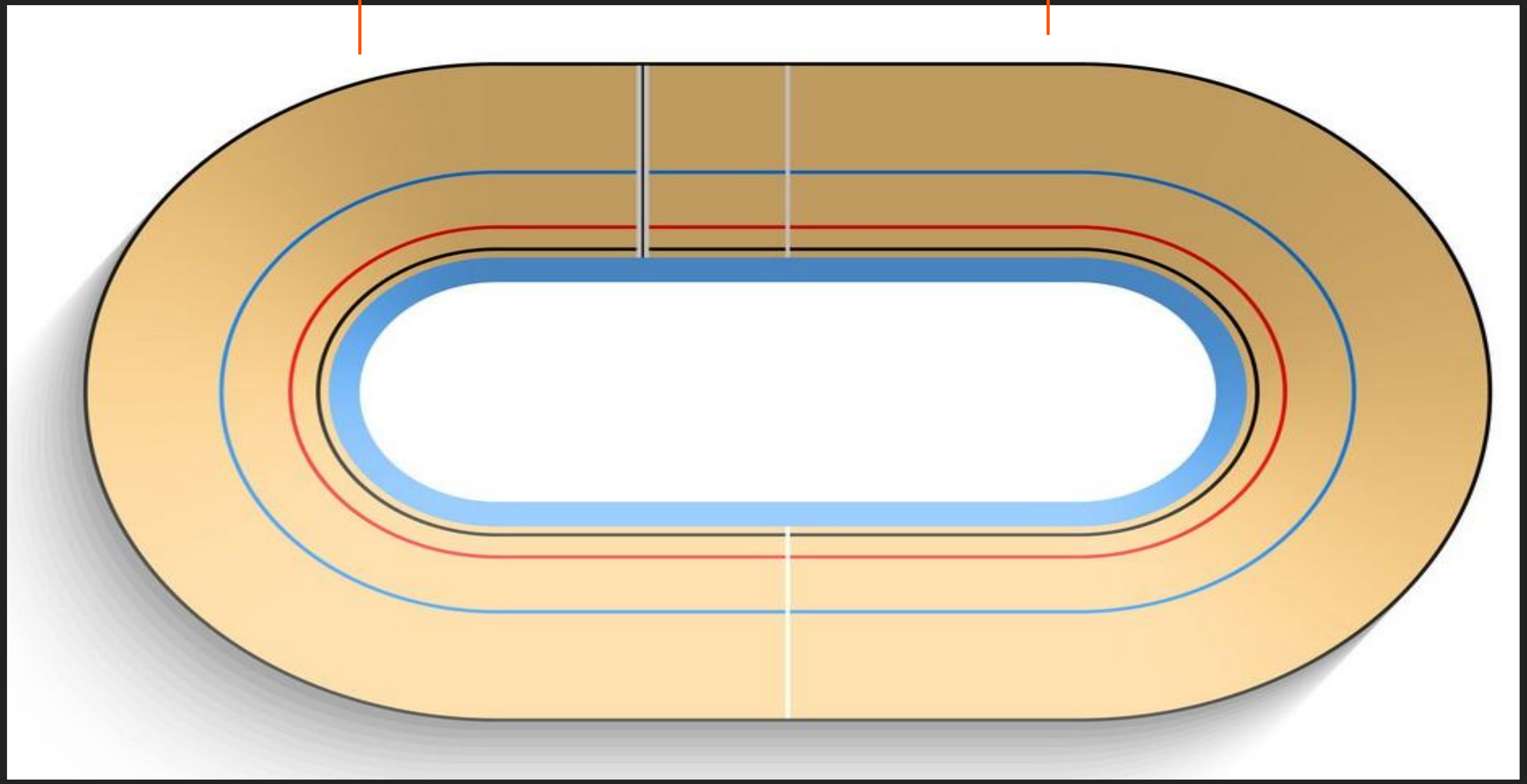
London 2012 Olympics: Bike chief's joke about roundness of our wheels fuels French row over our cycle success

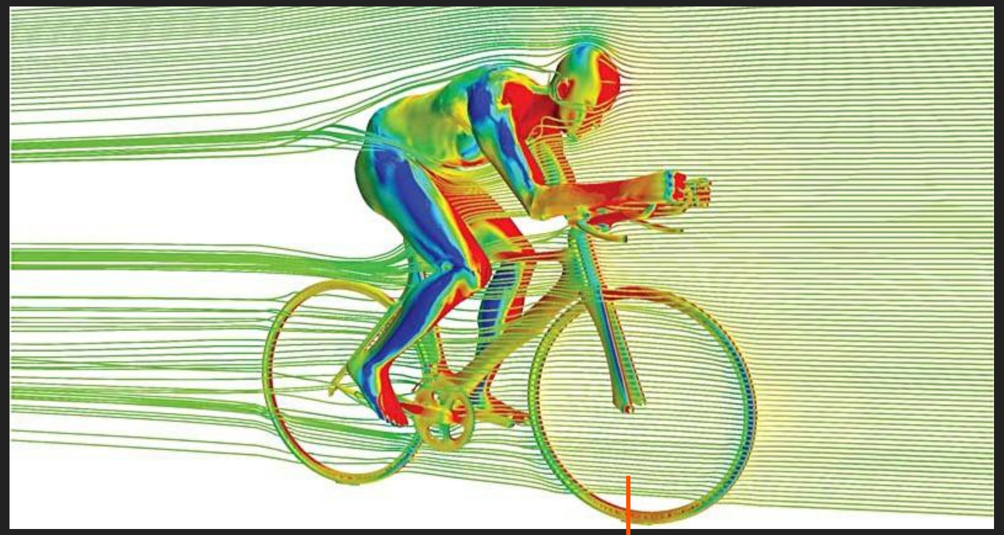
VIEW COMMENTS

Data Sources

Video from a number of cameras around the track which capture metrics on rider position and posture.

Induction Loop fitted underneath the track to monitor the rider position.





Wind Tunnel data captured through both real world wind tunnel trials and a digital wind tunnel hosted in AWS.

Weather Station next to the track to record atmospheric metrics within the velodrome

Hub fitted to each bike to record a number of 'onboard' metrics including power, speed, air pressure.

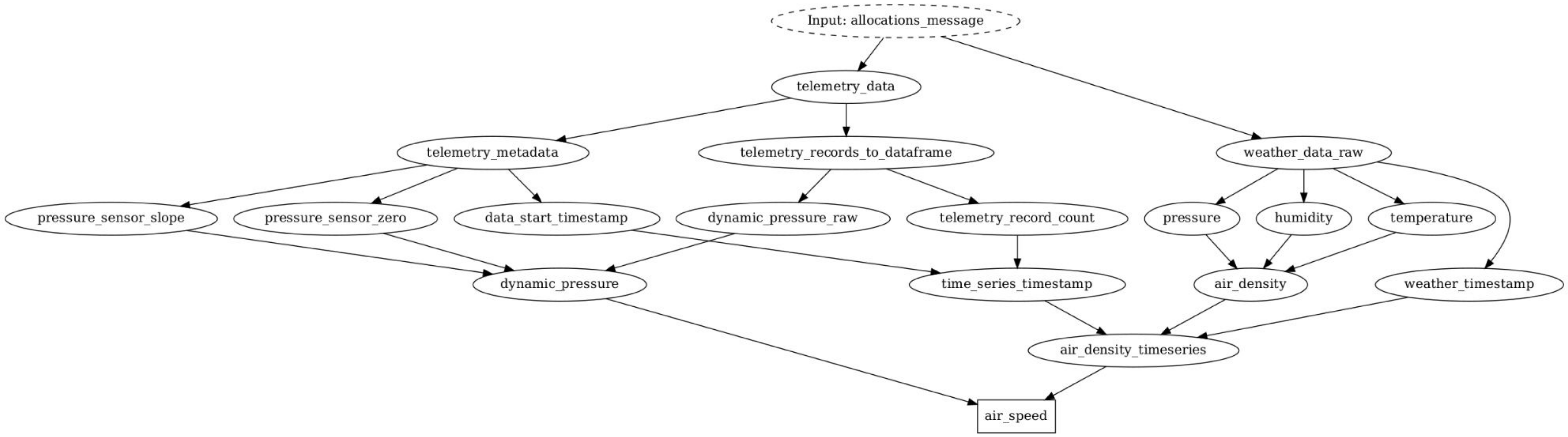
External Data Capture to capture data outside of the velodrome for road cyclists. Inc. heartrate, power & speed.

GBCT.io

Keystone

- Hamilton based
- Run as a cloud function
- Processes raw data into key metrics
- Flexible to different input sources
- Handles missing and poor quality data
- Self-documenting and modular

Data Lineage



```python
@tag(unit="m/s")
def air_speed(
    dynamic_pressure: pd.Series,
    air_density_timeseries: pd.Series,
) -> pd.Series:
    """Function to determine the oncoming air speed for timeseries data, where
    positive is a headwind.

    Units: m/s

    """
    return (2 * dynamic_pressure / air_density_timeseries).pow(0.5)
```

Introduction of Hamilton

Stefan Krawczyk

What is Hamilton?

micro-framework for defining dataflows

SWE best practices: ☑️ testing ☑️ documentation
☑️ modularity/reuse ☑️ lineage

"DBT for python functions"

`pip install sf-hamilton [came from Stitch Fix]`

www.tryhamilton.dev ← uses pyodide!

Hamilton: “a ha” moment

Table:

	spend	spend_zero_mean	spend_zero_mean_unit_variance
2023-01-01	10	-46	-1.173035
2023-01-02	10	-46	-1.173035
2023-01-03	20	-36	-0.918028
2023-01-04	40	-16	-0.408012
2023-01-05	40	-16	-0.408012

Idea: What if every output (column) corresponded to exactly one python fn?

Addendum: What if you could determine the dependencies from the way that function was written?

```python
def spend_zero_mean_unit_variance(
  spend_zero_mean: pd.Series, spend_std_dev: float
) -> pd.Series:
 """More docs would go here…"""
 return spend_zero_mean / spend_std_dev
```

Old Way vs Hamilton Way:

Instead of

```
df['c'] = df['a'] + df['b']
df['d'] = transform(df['c'])
```

Outputs == Function Name

Inputs == Function Arguments

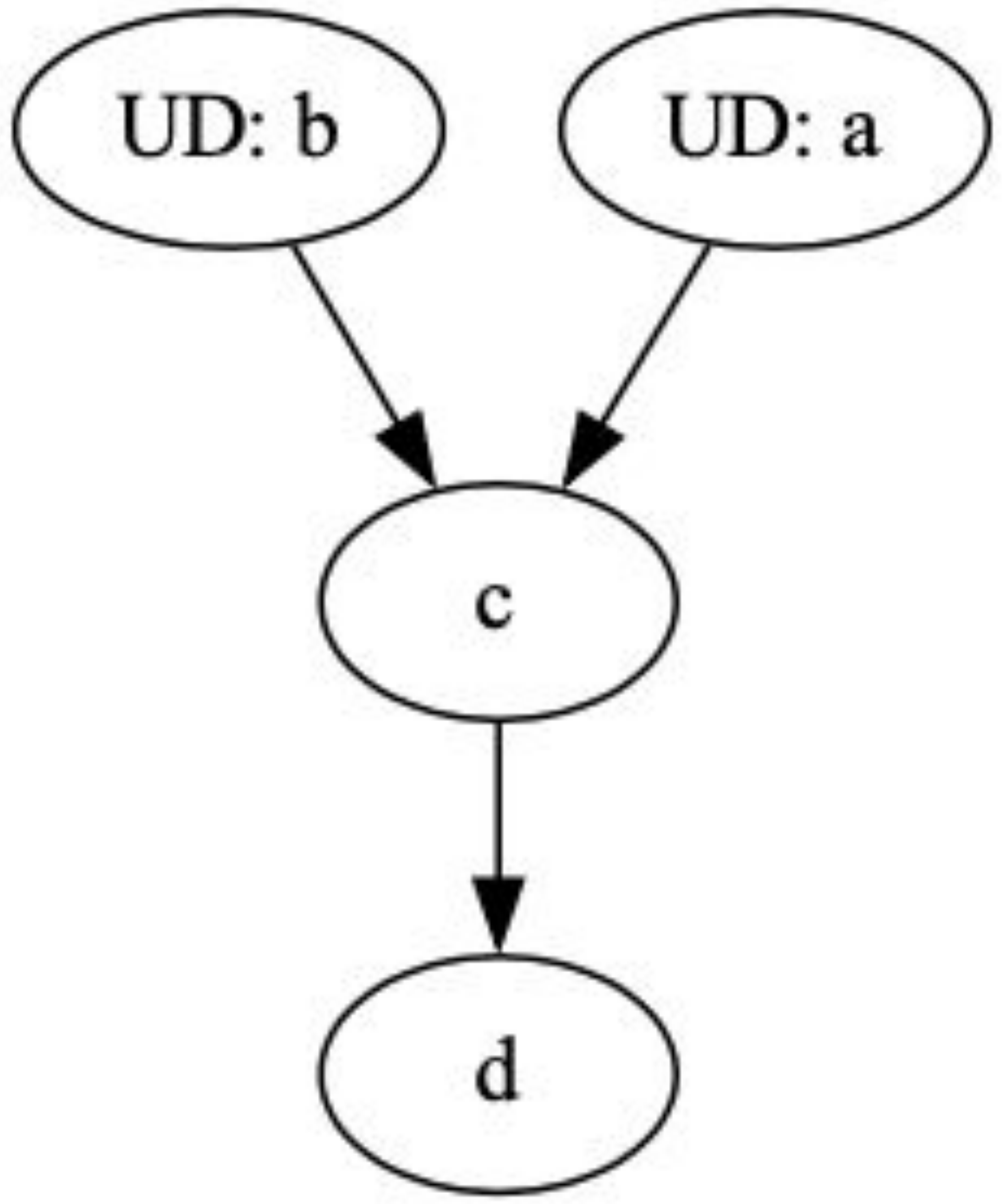
You declare

```
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```

Full Hello World

Functions

```
# feature_logic.py
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```



Driver says what/when to execute

```
# run.py
from hamilton import driver
import feature_logic
dr = driver.Driver({'a': ..., 'b': ...}, feature_logic)
df_result = dr.execute(['c', 'd'])
print(df_result)
```

Some Hamilton features that come naturally

A function:

```
# client_features.py
@tag(owner='Data-Science', pii='False')
@check_output(data_type=np.float64, range=(-5.0, 5.0), allow_nans=False)
def height_zero_mean_unit_variance(height_zero_mean: pd.Series,
                                   height_std_dev: pd.Series) -> pd.Series:
   """Zero mean unit variance value of height"""
   return height_zero_mean / height_std_dev
```

Features that come naturally with Hamilton:

● Unit & integration testing	✅ always possible & straightforward
● Documentation	✅ tags, visualization, function doc
● Modularity/reuse	✅ module curation & decoupled drivers; extensibile & expressive with decorators
● Central definition store (in code)	✅ naming, curation, versioning
● Data quality	✅ runtime checks
● It's just python code	✅ lightweight and flexible; not just for pandas

“Lineage via code”

UD: location
UD: index_col
titanic_data
UD: target_col
cabin
embarked
sex
sibsp
parch
target
cabin_encoder
embarked_encoder
pclass
sex_encoder
fare
family
cabin_category
embarked_category
encoders
sex_category
age
training_set_v1
UD: validation_size_fraction
UD: random_state
UD: max_depth
train_test_split_func
prefit_random_forest
X_train
y_train
fit_random_forest

dr.visualize_execution(...)

Combine with `@tag(...)` and can then ask questions like:

```
dr = driver.Driver(config, data_loading, features, sets, model_pipeline)
nodes = dr.what_is_upstream_of("fit_random_forest")
teams = {node.tags.get("owner") for node in nodes}
print(teams)
> {'data-science', 'data-engineering'}
```

Hamilton Summary

Write declarative functions, get a DAG!
Runs anywhere python runs

SWE best practices come out of the box - “*DBT for python functions*”

```
pip install sf-hamilton
```

www.tryhamilton.dev

⭐ https://github.com/dagworks-inc/hamilton

DAGWORKS

A quick word - we're building on top of Hamilton:

"Unifying platform layer for data & ML"

If you're interested in lineage, observability, and catalogs:

> Sign up for early access **www.dagworks.io**

Q&A