Module 1: Teradata Vantage SQL & Architecture Review

Day on the life of a Data Scientist Workshop

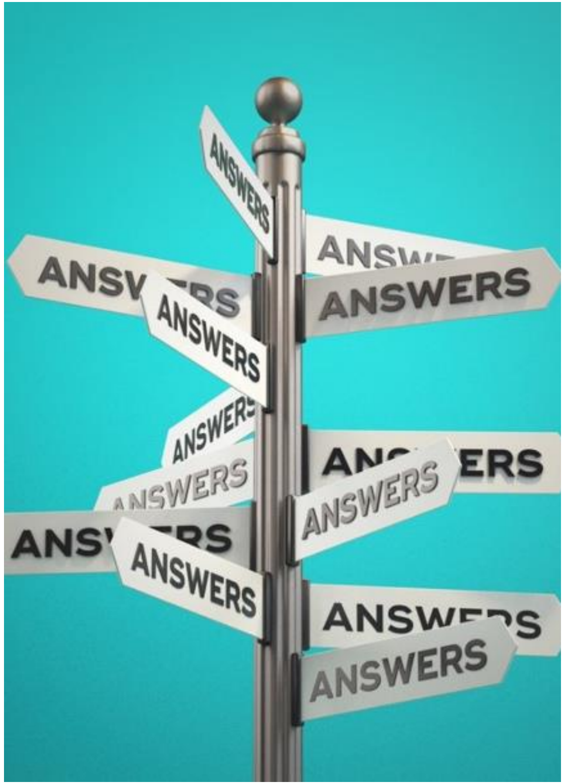
Objectives

After completing this module, you will be able to:

- Explain the differences between ANSI SQL and Vantage SQL syntax
- Explain key differences between Vantage SQL, Python, and R with Vantage functions
- List and describe key Vantage components including In-Database analytics, Vantage Analytics Library and Bring Your Own Model
- Describe Teradata's Analytics 1-2-3 and Analytic Ops

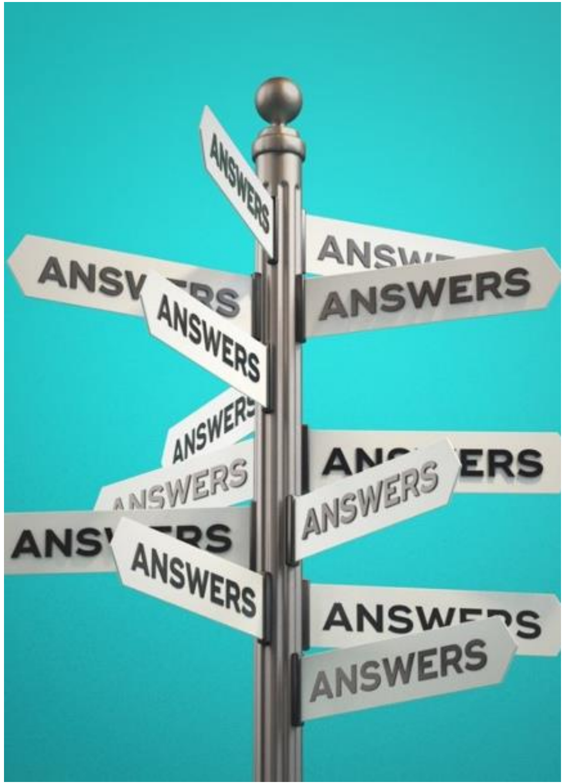
Topics

- Teradata Vantage SQL Syntax
- Teradata Vantage SQL, Python, and R Comparisons
- Teradata Vantage Analytics
- Teradata Vantage 1-2-3
- Summary



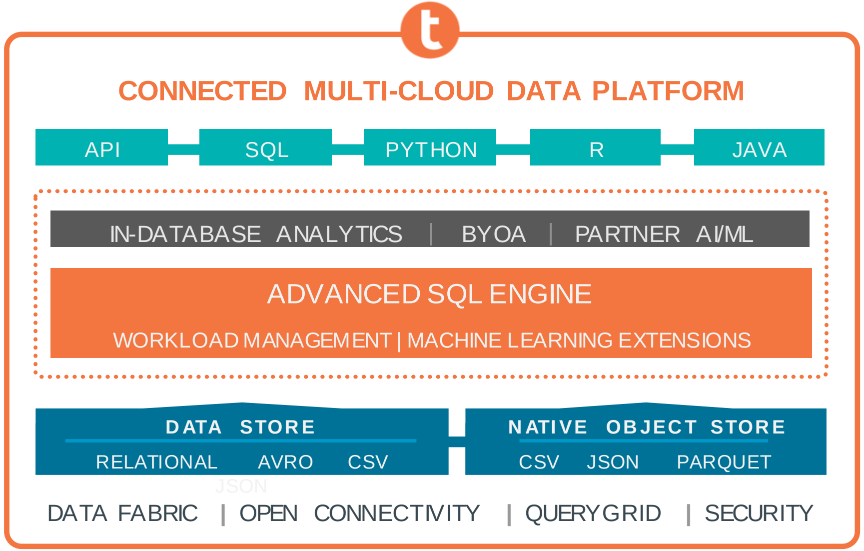
Current Topic – Vantage SQL Syntax

- **Teradata Vantage SQL Syntax**
- Teradata Vantage SQL, Python, and R Comparisons
- Teradata Vantage Analytics
- Teradata Vantage 1-2-3
- Summary



Introducing Teradata Vantage

- A **flexible, adaptive architecture** that minimizes unnecessary movement of data and aligns to the needs of users
- **Connects to customer ecosystems** and runs in **hybrid multi-cloud environments**
- Incorporates **additional tools, Languages and Engines**



Teradata Vantage Syntax

Teradata Vantage function syntax is very similar to ANSI SQL. The main differences follow:

ANSI SQL syntax

```
SELECT *
FROM my_table;
```

Teradata Vantage syntax (Advanced SQL Engine)

```
SELECT *
FROM <function_name>
ON my_table ...;
```

With an alias (optional)

```
SELECT *
FROM <function_name>
ON my_table ...AS dt;
```

<u>Key Vantage takeaway</u>:

FROM points to function
ON points to input table

- Unlike pure ANSI SQL, in Vantage SQL syntax the function's name follows the **FROM** keyword and the Teradata Input table follows the **ON** clause
- In both cases, the data table that we are interrogating appears after the **ON** keyword. Note that it is possible to have multiple **ON** clauses (both input table(s) and model dimension table(s))

Lab 1: Aliasing

The Advanced SQL Engine query runs fine with or without an alias. With alias is recommended for most functions.

Advanced SQL Engine Query (without alias)

```
SELECT * FROM Sessionize
(ON bank_web
 PARTITION BY customer_id
 ORDER BY datestamp
 USING
 TimeColumn ('datestamp')
 TimeOut (600)
);
```

Advanced SQL Engine Query (with alias)

```
SELECT * FROM Sessionize
(ON bank_web
 PARTITION BY customer_id
 ORDER BY datestamp
 USING
 TimeColumn ('datestamp')
 TimeOut (600)
) AS dt;
```

customer_id	page	datestamp	SESSIONID
36	ACCOUNT SUMMARY	2004-04-12 17:52:05.000000	0
32	ACCOUNT SUMMARY	2004-04-14 20:57:15.000000	0
36	FAQ	2004-04-12 17:56:00.000000	0
32	VIEW DEPOSIT DETAILS	2004-04-14 21:01:07.000000	0
36	FAQ	2004-04-12 17:59:58.000000	0
32	ACCOUNT SUMMARY	2004-04-15 10:24:53.000000	1

Lab 2: Function Argument Order

- TABLE arguments must appear <u>before</u> the **USING** clause
- All other function arguments must appear <u>after</u> the **USING** clause, but they need not appear in the order shown in the function syntax description

TABLE arguments appear <u>before</u> **USING** and preps data before Function runs. Order matters

FUNCTION arguments appear after **USING**

Order of arguments does not matter

```
SELECT * FROM nPath
(ON borre_z
 PARTITION BY user_id
 ORDER BY ts
 USING
 Symbols (event = 'a' AS X)
 Pattern ('X.X')
 Mode (NONOVERLAPPING)
 Result (Accumulate (event OF X) AS x_pattern))
AS dt;
```

Output

x_pattern
[a, a]
[a, a]

Lab 3: Partition By and Order By

- The next few pages will use the following dataset as the foundation for our discussion of **PARTITION BY**
- Typically, each **id** has events separated by ten (10) second intervals. Also, there is an event occurring every five (seconds) if we ignore **id**

ANSI SQL syntax

```
SELECT * FROM table1;
```

Data prior to Partitioning and Ordering

	id	event	ts
1	1	a	2001-09-27 23:00:00.000000
2	2	d	2001-09-27 23:00:05.000000
3	2	e	2001-09-27 23:00:10.000000
4	1	b	2001-09-27 23:00:10.000000
5	1	c	2001-09-27 23:00:20.000000
6	2	f	2001-09-27 23:00:25.000000
7	1	buy	2001-09-27 23:00:30.000000
8	2	buy	2001-09-27 23:00:35.000000

Lab 3: Partition By and Order By (cont.)

teradata.

Since we partitioned by **id** and ordered by **ts**, note how the **SESSIONID** value increments, starting at **0** for each partition **id**

```
SELECT * FROM Sessionize
(ON table1
 PARTITION BY id
 ORDER BY ts
 USING
 TimeColumn ('ts')
 TimeOut (5)
) ORDER BY id, ts;
```

1st: Prep data

2nd: Sessionize

Partition id = 1 (4 visits, per SESSIONID column)
Partition id = 2 (3 visits, per SESSIONID column)

	id	event	ts
1	1	a	2001-09-27 23:00:00.000000
2	1	b	2001-09-27 23:00:10.000000
3	1	c	2001-09-27 23:00:20.000000
4	1	buy	2001-09-27 23:00:30.000000
5	2	d	2001-09-27 23:00:05.000000
6	2	e	2001-09-27 23:00:10.000000
7	2	f	2001-09-27 23:00:25.000000
8	2	buy	2001-09-27 23:00:35.000000

TIMEOUT (5) specifies if rows occur within 5 seconds of one another *within each partition*, rows will be assigned to same **SESSIONID**

Per partition, any upstream click within 5 seconds of current click has same **SESSIONID** #

'Dimension' Syntax

- Certain functions require tables to be distributed to all Processing engines for performance reasons. This requires the **DIMENSION** keyword in your code
- This is typical when you have Multiple **ON** clauses in your code and want the Model table to be distributed to all AMPs/vWorkers prior to processing
- Typically, these dimension tables can be thought of as "metadata" tables, housing argument values for the function that you are invoking

Process rows on the current Processing Engine (AMPS-SQL, vWorkers-MLE) on which they exist (in other words, don't shuffle rows)

Shuffle entire table to every AMP/vWorker

```
SELECT * FROM NaiveBayesPredict
(ON income_table PARTITION BY ANY
 ON nb_table_model AS model DIMENSION
 USING
 IDCol ('id') ...
```

EXPLAIN PLAN shows duplicating table on all AMPs

33	operator TblOpInputSpool) (all_amps), which is duplicated on
34	all AMPs in TD_Map1. The size of Spool 3 is estimated with

Lab 4: Dimension Tables


```
SELECT * FROM conv_table;

SELECT * FROM model1_table ORDER BY id;

EXPLAIN
SELECT * FROM Attribution
(ON borre_y PARTITION BY user_id ORDER BY ts
 ON conv_table AS conversion DIMENSION
 ON model1_table AS model1 DIMENSION
 USING
 EventColumn ('event')
 TimestampColumn ('ts')
 WindowSize ('rows:12')
) AS dt ORDER BY user_id, ts;
```

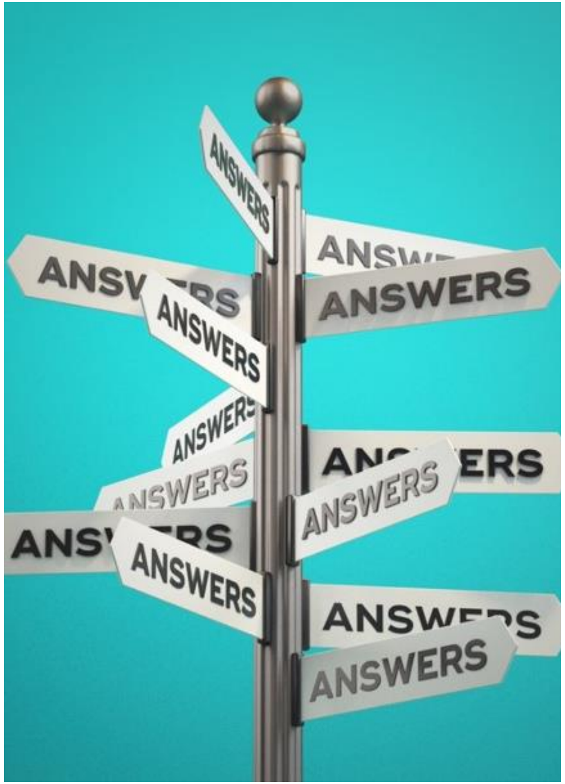
conversion_events
buy

id	model
0	SEGMENT_ROWS
1	3:0.4:UNIFORM:NA
2	3:0.3:LAST_CLICK:NA
3	3:0.2:EXPONENTIAL:0.5,ROW
4	3:0.1:FIRST_CLICK:NA

	user_id	event	ts	attribution	time_to_conversion
1	1	a	2017-01-01 13:21:01.000000	0	
2	1	a	2017-01-01 13:21:02.000000	0	
3	1	a	2017-01-01 13:21:03.000000	0	
4	1	a	2017-01-01 13:21:04.000000	0.1000...	-12
5	1	a	2017-01-01 13:21:05.000000	0	
6	1	a	2017-01-01 13:21:06.000000	0	
7	1	a	2017-01-01 13:21:07.000000	0.0285...	-9
8	1	a	2017-01-01 13:21:08.000000	0.0571...	-8
9	1	a	2017-01-01 13:21:09.000000	0.11428...	-7
10	1	a	2017-01-01 13:21:10.000000	0	
11	1	a	2017-01-01 13:21:11.000000	0	
12	1	a	2017-01-01 13:21:12.000000	0.3000...	-4
13	1	a	2017-01-01 13:21:13.000000	0.1333...	-3
14	1	a	2017-01-01 13:21:14.000000	0.1333...	-2
15	1	a	2017-01-01 13:21:15.000000	0.1333...	-1
16	1	buy	2017-01-01 13:21:16.000000		

Current Topic – SQL, Python, and R Comparisons

- Teradata Vantage SQL Syntax
- **Teradata Vantage SQL, Python, and R Comparisons**
- Teradata Vantage Analytics
- Teradata Vantage 1-2-3
- Summary



Before We Begin: Compare Languages via 'Sessionize' Function (SQL vs. Python vs. R)

teradata.

SQL syntax

```
SELECT * FROM SESSIONIZE
 ( ON compnew_tdtable
   PARTITION BY CUSTOMER_ID
   ORDER BY DATESTAMP
   USING
    TIMECOLUMN('DATESTAMP')
    TIMEOUT(86400) );
```

CUSTOMER_ID	DATESTAMP	EVENT	CHURN_FLAG	SESSIONID
1,455.00000	2018-04-23 04:09:00.0...	Neutral Call	N	0
1,484.00000	2018-04-12 13:09:00.0...	Web Chat	N	0
1,455.00000	2018-04-24 10:09:00.0...	Store Visit	N	1
1,484.00000	2018-04-13 20:32:00.0...	Product Browsing	N	1
1,455.00000	2018-04-24 10:15:00.0...	Purchase	N	1

Python syntax

```
Psession_obj = Sessionize
 ( data = compnew_df
 , data_partition_column = 'CUSTOMER_ID'
 , data_order_column = 'DATESTAMP'
 , time_column = 'DATESTAMP'
 , time_out = 86400.0 )
# Print result
print(Psession_obj)
```

	CUSTOMER_ID	DATESTAMP	EVENT	CHURN_FLAG	SESSIONID
0	1526.000	2018-04-03 15:22:00.000000	Store Visit	N	0
1	1526.000	2018-04-05 03:28:00.000000	Web Chat	N	1
2	1526.000	2018-04-05 03:34:00.000000	Service Inquiry	N	1
3	1484.000	2018-04-12 13:09:00.000000	Web Chat	N	0
4	1484.000	2018-04-14 21:20:00.000000	Service Inquiry	N	2
5	1484.000	2018-04-19 17:37:00.000000	Store Visit	N	3

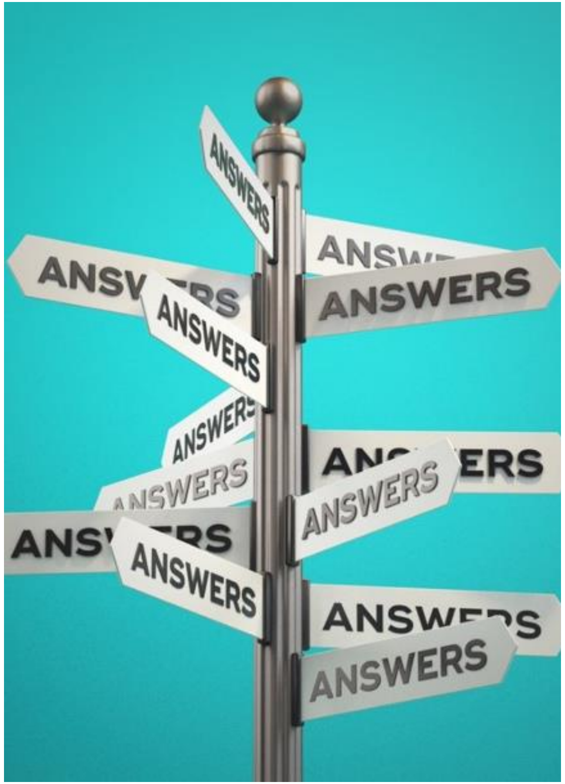
R syntax

```
Rsession_obj <- td_sessionize
 ( data = compnew_df
 , data.partition.column = 'CUSTOMER_ID'
 , data.order.column = 'DATESTAMP'
 , time.column = 'DATESTAMP'
 , time.out = 86400 )
# Print result
print(Rsession_obj)
```

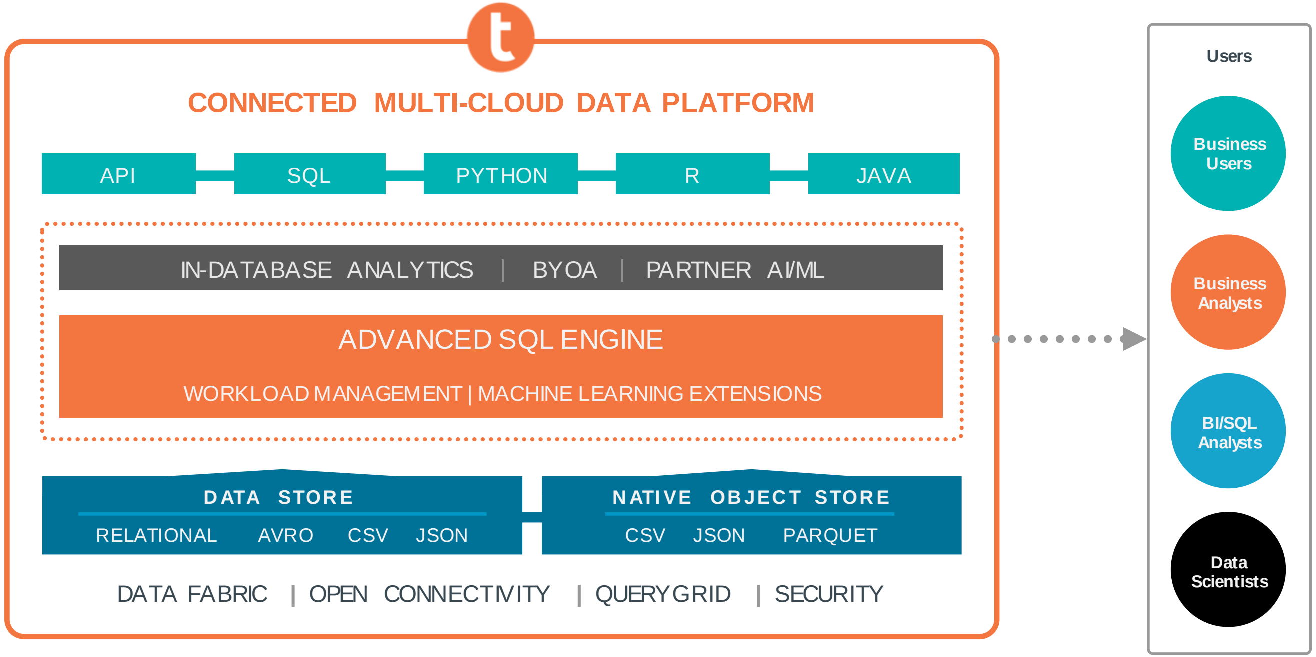
	CUSTOMER_ID	DATESTAMP	EVENT	CHURN_FLAG	SESSIONID
	<dbl>	<dttm>	<chr>	<chr>	<int>
1	1484	2018-04-14 21:20:00	Service Inquiry	N	2
2	1484	2018-04-19 17:37:00	Store Visit	N	4
3	1484	2018-04-19 17:42:00	Purchase	N	4
4	1578	2018-03-23 13:30:00	Return Policy Inquiry	N	0
5	1455	2018-04-24 10:09:00	Store Visit	N	1

Current Topic – Teradata Vantage Analytics

- Teradata Vantage SQL Syntax
- Teradata Vantage SQL, Python, and R Comparisons
- **Teradata Vantage Analytics**
- Teradata Vantage 1-2-3
- Summary



Teradata Vantage



USERS

Business Users

Business Analysts

BI/SQL Analysts

Data Scientist
s

TOOLS

VANTAGE CONSOLE

jupyter

R Studio

sas

data iku

+ a b | e a u

LANGUAGES

SQL

R

python™

Java

C++

APIs/SDKs

T E R A D A T A V A N T A G E

In-DB Analytic Functions 17.10

Data Cleaning

SimpleImputeFit
SimpleImputeTransform
GetRowsWithMissinValues
OutlierFilterFit
OutlierFilterTransform
ConvertTo

Data Exploration

ColumnSummary
CategoricalSummary
UnivariateStatistics
GetRowsWithoutMissingValues
WhichMin
WhichMax
Histogram
QQNorm
Z-test
F-test
Chi-Square
Cramer's V

Feature Engineering

CATEGORICAL VARIABLE TRANSFORM

One-hot Encoding Fit
One-hot Encoding Transform

CONTINUOUS VARIABLE TRANSFORM

ScaleFit
ScaleTransform

FunctionFit
FunctionTransform
PolynomialFeatureFit
PolynomialFeatureTransform

RowNormalizeFit
RowNormalizeTransform
BinCodeFit
BinCodeTransform

UTILITIES

NumApply
StrApply
FillRowID
RoundColumns

SQLE 16.20

MODEL BUILDING

nPath
Attribution
Sessionize

SCORING

DecisionForestPredict
DecisionTreePredict
GLMPredict
NaiveBayesPredict
NaiveBayesTextClassifierPredict
SVMSparsePredict

FEATURE ENGINEERING

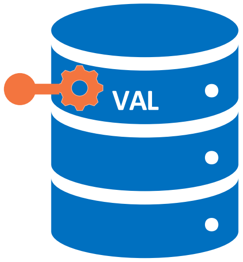
MovingAverage
NGramSplitter

DATA CLEANING AND EXPLORATION

Pack
Unpack
StringSimilarity

Features of Vantage Analytics Library (VAL)

- Robust, flexible, and mature analytics available as an every-unit item.
- Runs on SQLE 16.20, 17.0 and above.
- Compatible with all operating environments (AWS, Azure, GCP, TDVM, IFX).
- VAL external stored procedure (XSP) behaves like a database version of an API.
- Easy to install and requires only 40MB of space in the SQLE database for the functions and sample data. No other system resources needed outside of the database.
- Can be installed at existing customer sites.
- Release cadence for VAL independent of SQLE or Vantage.
- Works with NOS.



teradata.

Bring Your Own Model (BYOM): Overview

What is "Bring Your Own Model" in Vantage?

A new Vantage capability that extends model scoring capability to models created and trained outside Vantage. It allows:

- Importing and storing models (saved in binary format in model table) within Vantage
- Scoring / predicting the saved models at scale using Vantage's scoring function, on all data

Supported Types of Standard Model Format

- **Available Now: Predictive Model Markup Language (PMML)***
  - XML-based predictive model interchange forma
- Planned in future:
  - Open Neural Network Exchange (ONNX)
  - MLeap
  - H2O Model Object, Optimized (MOJO)

PMML Predictive Model Markup Language

ONNX

mleap

H2O.ai

* Supported PMML versions: 2.0 or 4.4

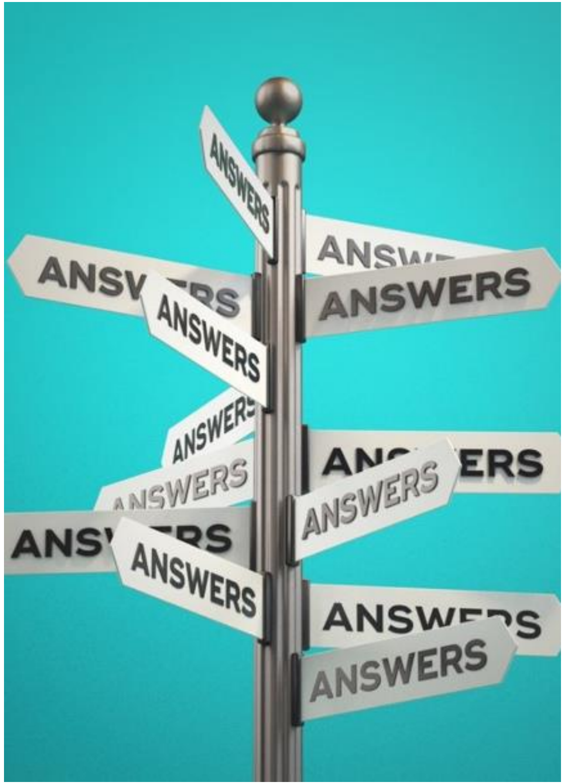
Bring Your Own Model: Benefits

Build predictive models in **any language** using **most-popular tools / platforms** and **operationalize them at scale** in Vantage using **all of its data**.

- **Flexibility:** Use existing models created and trained by open-source / third party vendor products; users can continue to use languages and tools / platform of their choice for model creation.
- **Score / Train without Coding:** Utilize Vantage's scoring functions; users do not have to code complex scoring algorithm.
- **Score new data without data movement:** No need to move data out of Vantage to score.
- **Performant:** Scoring functions run significantly faster than analytics running outside Vantage:
  - **Linearly Scalable:** Inherits Vantage's MPP architecture and its linear scalability.
  - **Concurrency:** Multiple function calls can run without significantly impacting overall system performance.
  - **Workload Management:** Workloads associated to the scoring function can be easily managed using Vantage workload management feature.

Current Topic – Teradata Vantage 1-2-3

- Teradata Vantage SQL Syntax
- Teradata Vantage SQL, Python, and R Comparisons
- Teradata Vantage Analytics
- **Teradata Vantage 1-2-3**
- Summary



Analytics 1-2-3 Strategy

Prepare data

50-80% of time is taken preparing raw data:

- Data integration
- Data access and exploration
- Data cleansing
- Feature engineering
- Feature selection

Leverage **Data Labs** to support rapid experimentation and build a **Feature Store** of variables with known predictive value.

Train model

Fit ML algorithm to the training data:

- Algorithm selection
- Test and training data-set split
- Model training and evaluation
- Model optimization
- Model export

Be prepared to use multiple analytic tools—but ensure that they are trained on data pulled from the Feature Store and that **models are consumable.**

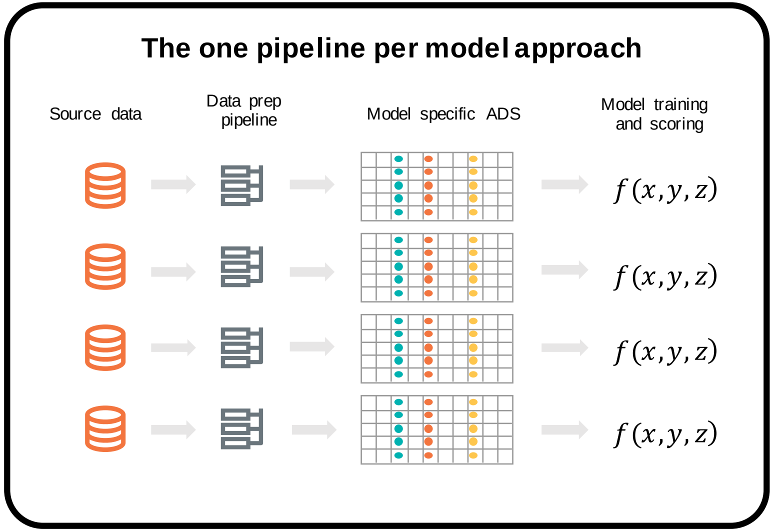
Deploy model

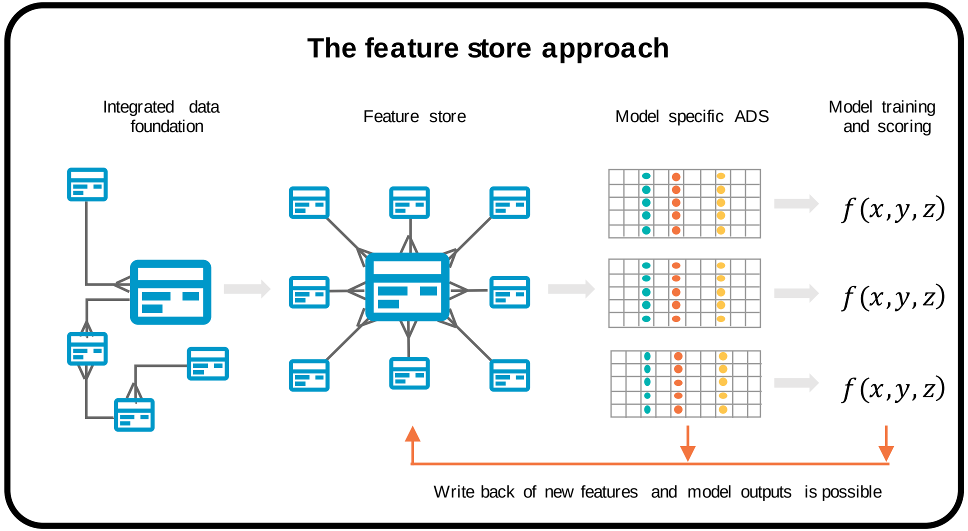
Operationalize model to predict outcomes:

- Write-back new features
- Import model to model repository
- Operational scoring
- Business process integration
- Model monitoring

Bring models to the data in the Feature Store wherever possible; instrument models to capture meta-data and predictions.

The Enterprise Feature Store – Data Management for AI



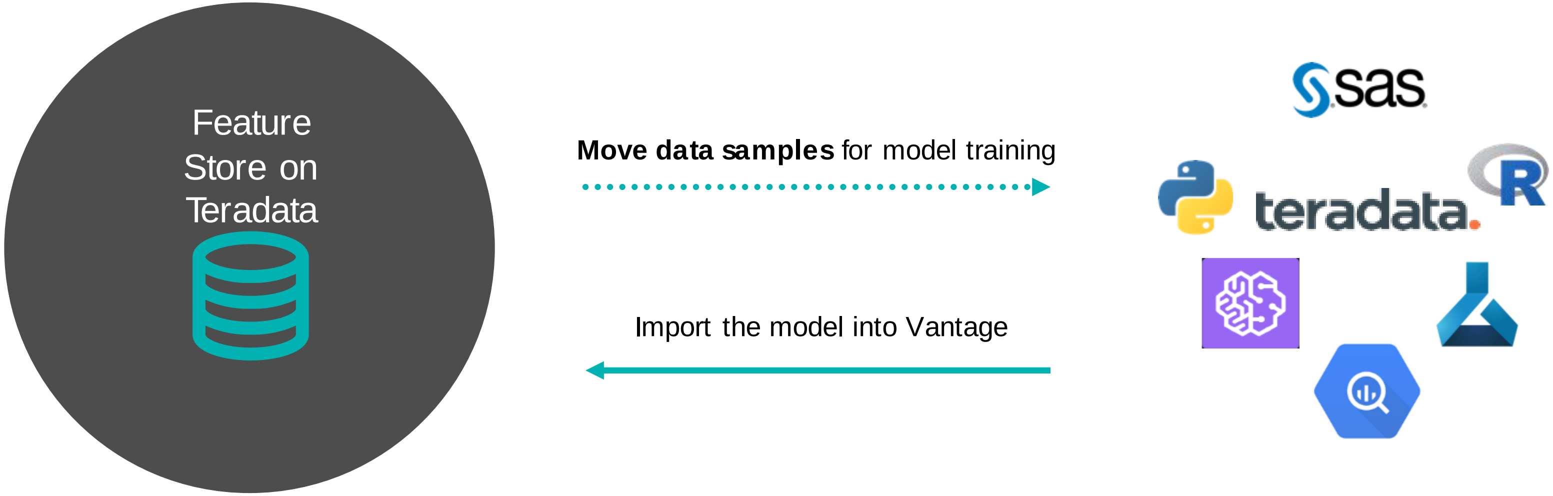


One pipeline per model → Long data prep cycles and poor time to market
Redundant infrastructure, processing, and effort → High TCO
Limited re-use of pipeline or features → Poor productivity and data silos
DSs functioning mostly as data janitors → Inefficient allocation of resources

"Off the peg" features dramatically improve analytic cycle times and time-to-market
Extensive re-use reduces TCO and improves analytic data quality and predictive model accuracy
ADS layer enables model-specific customization, whilst eliminating analytic data silos
Separation of duties and improved productivity

Train a Model with Any Tool

Allowing the data science freedom to use their preferred tools



Data movement is minimized by importing the model back into Vantage for production scoring

Build model with any tool you prefer

BYOM – Use Any Tool & Productionize with Teradata

Radical architectural simplification, performance at-scale, reduced TCO

Convert models to SQL queries

SQL

EMEA Bank: round-trip credit scoring in < 2s running 15 jobs in parallel.

Use language-specific model format with in-DBMS interpreters

R python™

Global Bank: complex income estimation models scored for 7M customers in < 23m (previous approach would not scale past 20k test records).

Use common serialization format and IVSM accelerator

PMML Predictive Model Markup Language ONNX MOJO H2O.ai

Asian Lottery: estimated 10X reduction in model scoring costs on Teradata compared with current Databricks-based approach.

Teradata AnalyticOps Accelerator – Automated Data Science Workflows

- Core capabilities and practices enabled by technology

Data Ingestion

Quickly expose both highly and lightly integrated data to your data scientists

Create Project

Create a use case project and Git repository

Discovery

Create a notebook, analyze the data, create new features, and model experimentation

Create AI Model

Once a good model is found, create the necessary code with the help of our CLI and commit

Train and Evaluate

Perform full training and evaluate new AI model version for approval

Approve and Deploy

Review model reports, compare, approve, and deploy

Score and Monitor

Create predictions (batch, in-database, real-time), capture and monitor use case performance

Analytics 1-2-3 Strategy

Prepare data	Train model	
Reuse	**Flexibility**	**Value**
Efficiency	Innovation	Automation
Speed to market	Open to all Tools	ROI
Data Integration Feature Engineering Enterprise Feature Store	Train with open source or proprietary tools Use production Data	BYOM Model Scoring Enterprise-wide results
Experience with large Enterprises	Insights, use cases and expertise	Knowledge of end-to-end process

Summary

In this module, you learned how to:

- Explain the differences between ANSI SQL and Vantage SQL syntax
- Explain key differences between Vantage SQL, Python, and R with regard to Vantage functions
- List and describe key Vantage components including In-Database analytics, Vantage Analytics Library and Bring Your Own Model
- Describe Teradata's Analytics 1-2-3

Thank you.