AZURE DATA FACTORY – DATA FLOWS

BRINGING THE "T" IN ELT

WHO AM I?

t: @bizdataviz
In/cseferlis

- Practice Manager – Pragmatic Works – www.pragmaticworks.com
- Former CIO
- Blog, Speak, Record, Tweet
    - Twitter: @bizdataviz
    - LinkedIn: in/cseferlis
    - http://blog.bizdataviz.com
- MBA from UMass
- Ski, Bike, Hike, Run and drag the family

along…

WHY ARE WE HERE?

t: @bizdataviz
In/cseferlis

- Modern Data Warehouse (Platform)
- Azure Data Orchestration
- Preview New Azure Data Factory Features
- Full Circle ELT with Azure Data Factory

AZURE DATA FACTORY… QUICKLY

t: @bizdataviz
In/cseferlis

- Native Cloud Data Orchestration Tool
- Started with v1, but lacked many features
- v2 goes GA 6/2018 – work has begun on Data Flows
- Data Flows and other new features currently in Limited Preview

LIMITED PREVIEW FEATURES

t: @bizdataviz
In/cseferlis

- Debug Mode
- Data Inspection
- Transformations
- GitHub Support
- Detailed Flow Inspection
- All-New Expression Builder

VISUAL DATA FLOW AUTHORING

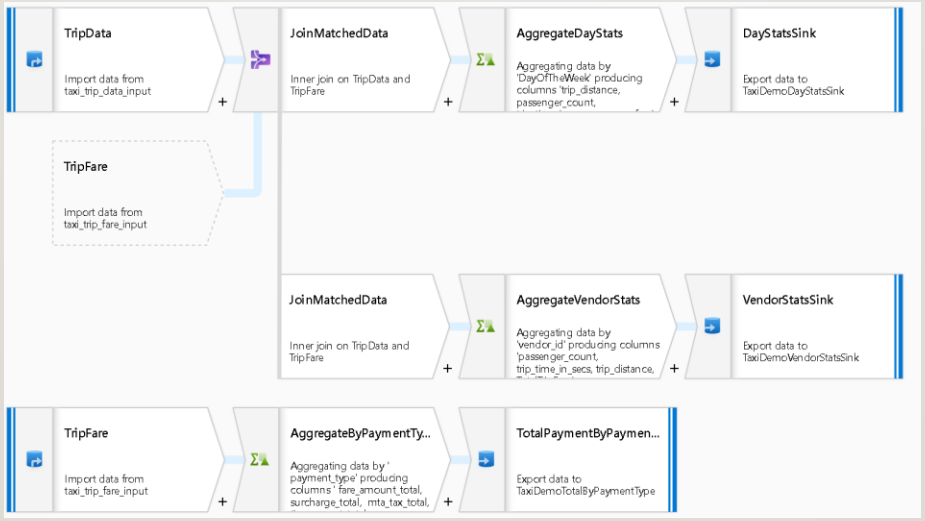
t: @bizdataviz
In/cseferlis

- Transform Data, At Scale, in the Cloud, Zero-Code
  - Cloud-first, scale-out ELT
  - Code-free dataflow pipelines
- Serverless scale-out transformation execution engine
- Maximum Productivity for Data Engineers
  - Does NOT require understanding of Spark / Scala / Python / Java
- Resilient Data Transformation Flows
  - Built for big data scenarios with unstructured data requirements
  - Operationalize with Data Factory scheduling, control flow and monitoring

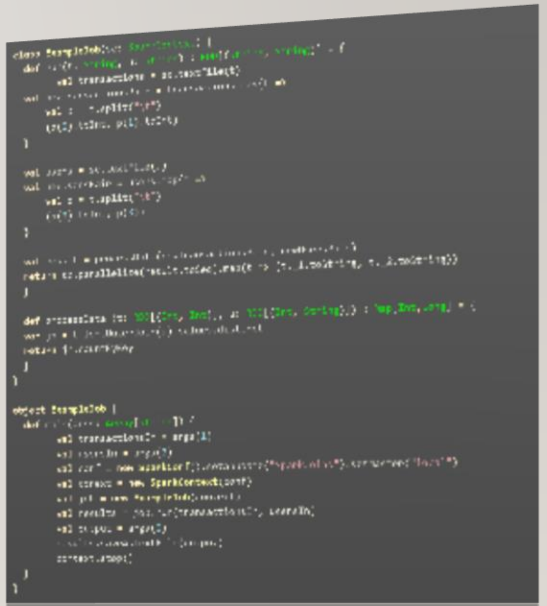
CODE-FREE DATA TRANSFORMATION AT SCALE

t: @bizdataviz
In/cseferlis

- Does not require understanding of Spark, Big Data Execution Engines, Clusters, Scala, Python …
- Focus on building business logic and data transformation
  - Data cleansing
  - Aggregation
  - Data conversions
  - Data prep
  - Data exploration

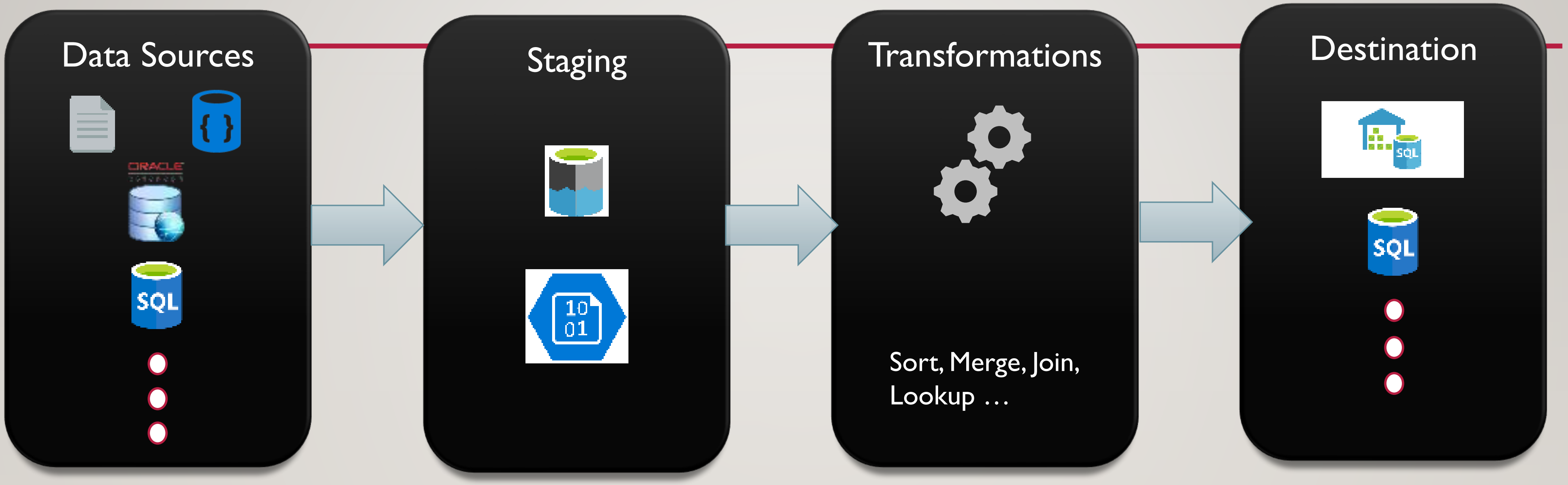

TripData
Import data from taxi_trip_data_input
JoinMatchedData
Inner join on TripData and TripFare
AggregateDayStats
Aggregating data by 'DayOfTheWeek' producing columns 'trip_distance, passenger_count,
DayStatsSink
Export data to TaxiDemoDayStatsSink
TripFare
Import data from taxi_trip_fare_input
JoinMatchedData
Inner join on TripData and TripFare
AggregateVendorStats
Aggregating data by 'vendor_id' producing columns 'passenger_count, trip_time_in_secs, trip_distance,
VendorStatsSink
Export data to TaxiDemoVendorStatsSink
TripFare
Import data from taxi_trip_fare_input
AggregateByPaymentTy...
Aggregating data by 'payment_type' producing columns ' fare_amount_total, surcharge_total, mta_tax_total,
TotalPaymentByPaymen...
Export data to TaxiDemoTotalByPaymentType

… not …



ADF DATA FLOW WORKSTREAM

t: @bizdataviz
In/cseferlis



- Explicit user action
- User places data source(s) on design surface, from toolbox
- Select explicit sources

- Implicit/Explicit
- Data Lake staging area as default
- User does not need to configure this manually

- Explicit user action
- User places transformations on design surface, from toolbox
- User must set properties for transformation steps and step connectors

- Explicit user action
- User chooses destination connector(s)
- User sets connector property options

DATA FLOW LIMITED PREVIEW

t: @bizdataviz
In/cseferlis

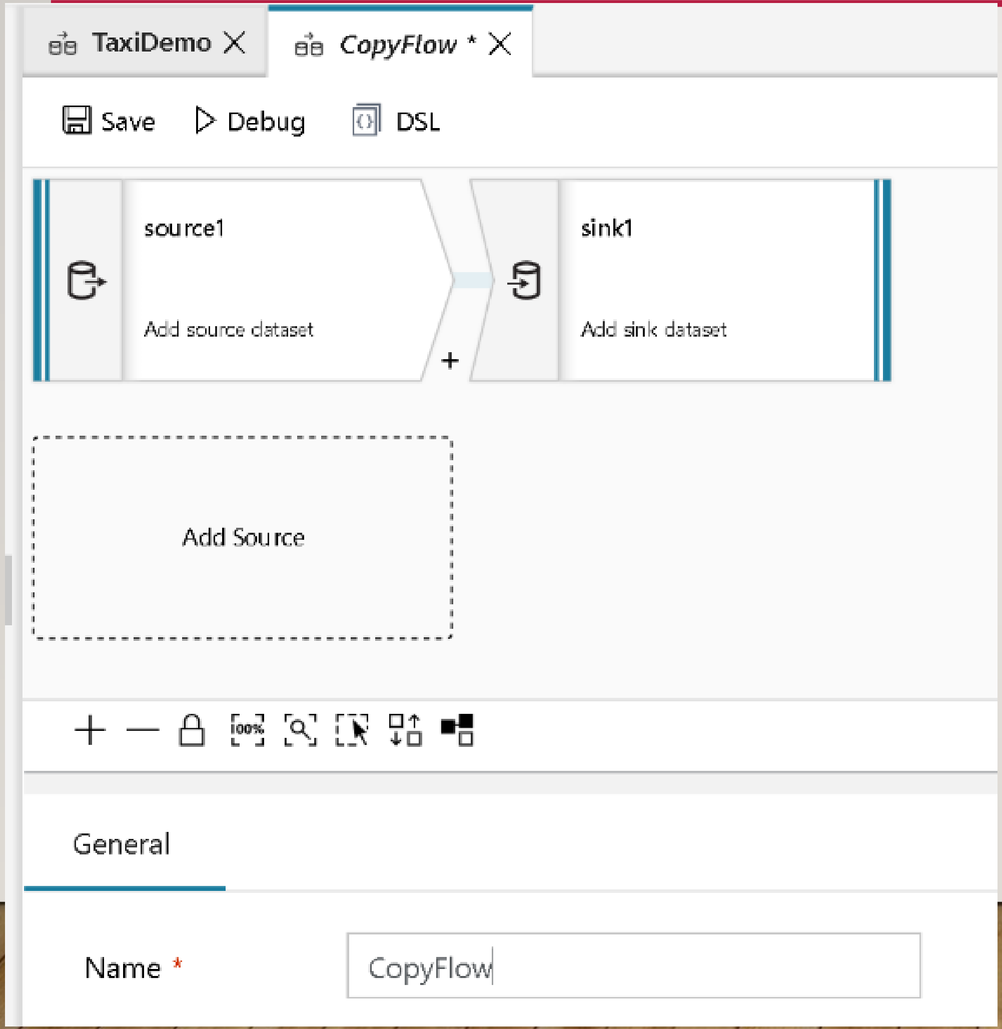
- Azure SLAs are NA for preview services (private or public preview) until GA of the service.
- Limited Preview Support
  - Handled directly with the Azure Engineering team via adfdataflowext@microsoft.com.
- Sign-up for ADF Data Flow service
  - http://aka.ms/dataflowpreview
  - Microsoft Azure must whitelist your subscription ID to turn on the feature for you
- Public Preview Support
  - Normal Azure customer service channels

t: @bizdataviz
In/cseferlis

COMMON ELT SCENARIOS HANDLED WITH ADF – DATA FLOWS

SIMPLE COPY FLOW

t: @bizdataviz
In/cseferlis

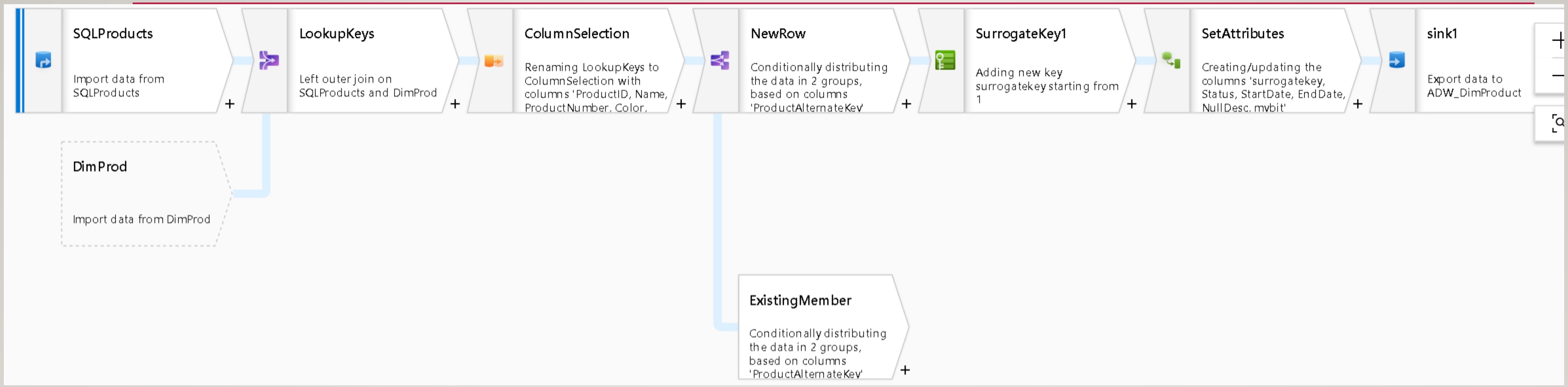


Name * CopyFlow

- ADF Data Flow is a guided construction process
- Begin by defining the Datasets for your Source and Sink
- Add Transformations to each node in your data flow
- Or simply copy from source to sink with no transformation
- Map columns and fields along the way

SLOWLY CHANGING DIMENSION

t: @bizdataviz
In/cseferlis



SQLProducts
Import data from SQLProducts

DimProd
Import data from DimProd

LookupKeys
Left outer join on SQLProducts and DimProd

ColumnSelection
Renaming LookupKeys to ColumnSelection with columns 'ProductID, Name, ProductNumber, Color,

NewRow
Conditionally distributing the data in 2 groups, based on columns 'ProductAlternateKey'

ExistingMember
Conditionally distributing the data in 2 groups, based on columns 'ProductAlternateKey'

SurrogateKey1
Adding new key surrogatekey starting from 1

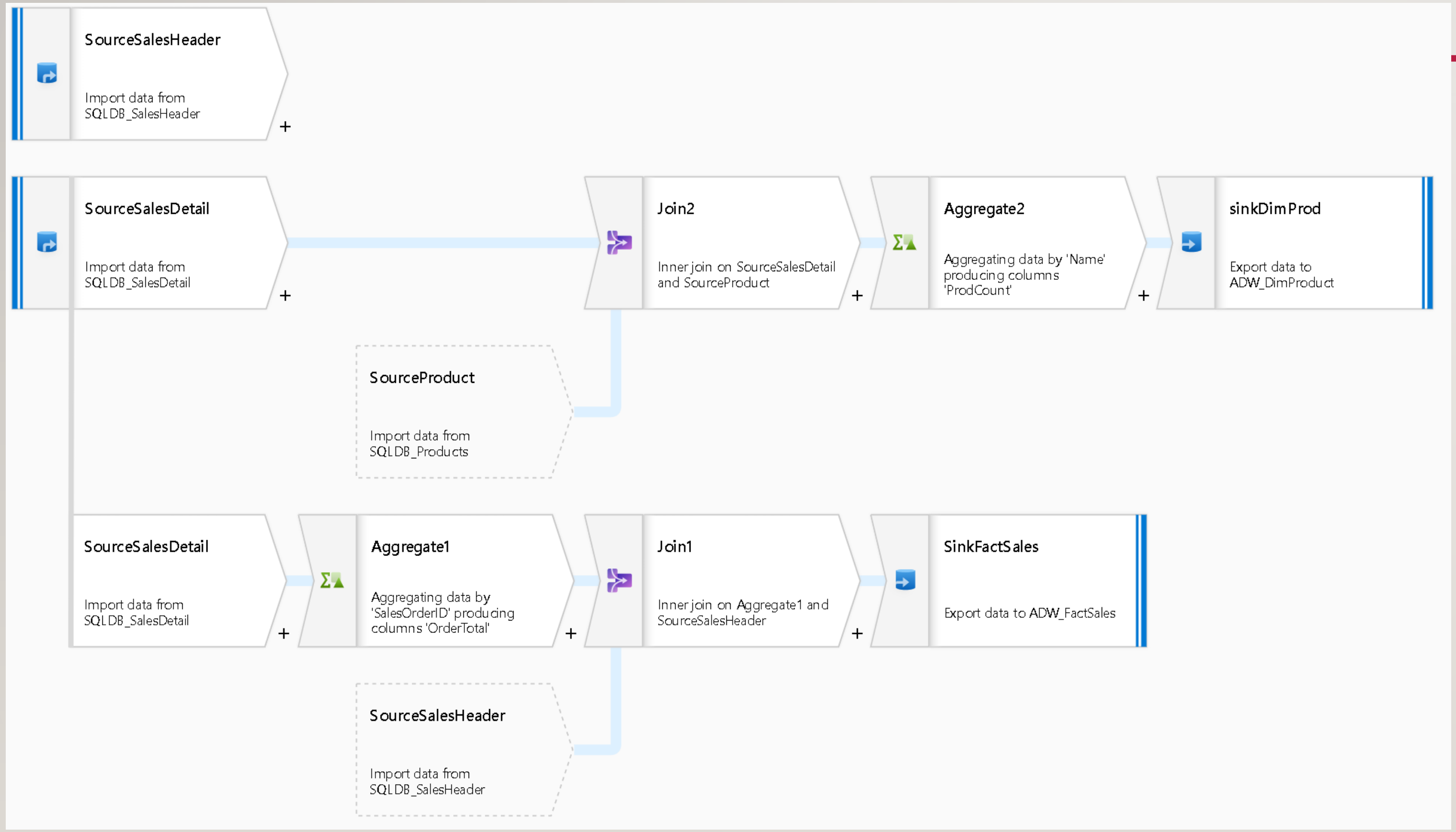
SetAttributes
Creating/updating the columns 'surrogatekey, Status, StartDate, EndDate, NullDesc, mvbit'

sink1
Export data to ADW_DimProduct

- Common DW pattern to manage changing attributes to dimension members
- Graphically build code-free SCD ETL pattern to load your data warehouse
- Connect directly to Azure SQL DB and Azure SQL DW
- Use Lookup, Surrogate Key, Derived Column and Select transforms

LOAD STAR SCHEMA DW

t: @bizdataviz
In/cseferlis



SourceSalesHeader
Import data from SQLDB_SalesHeader

SourceSalesDetail
Import data from SQLDB_SalesDetail

Join2
Inner join on SourceSalesDetail and SourceProduct

Aggregate2
Aggregating data by 'Name' producing columns 'ProdCount'

sinkDimProd
Export data to ADW_DimProduct

SourceProduct
Import data from SQLDB_Products

SourceSalesDetail
Import data from SQLDB_SalesDetail

Aggregate1
Aggregating data by 'SalesOrderID' producing columns 'OrderTotal'

Join1
Inner join on Aggregate1 and SourceSalesHeader

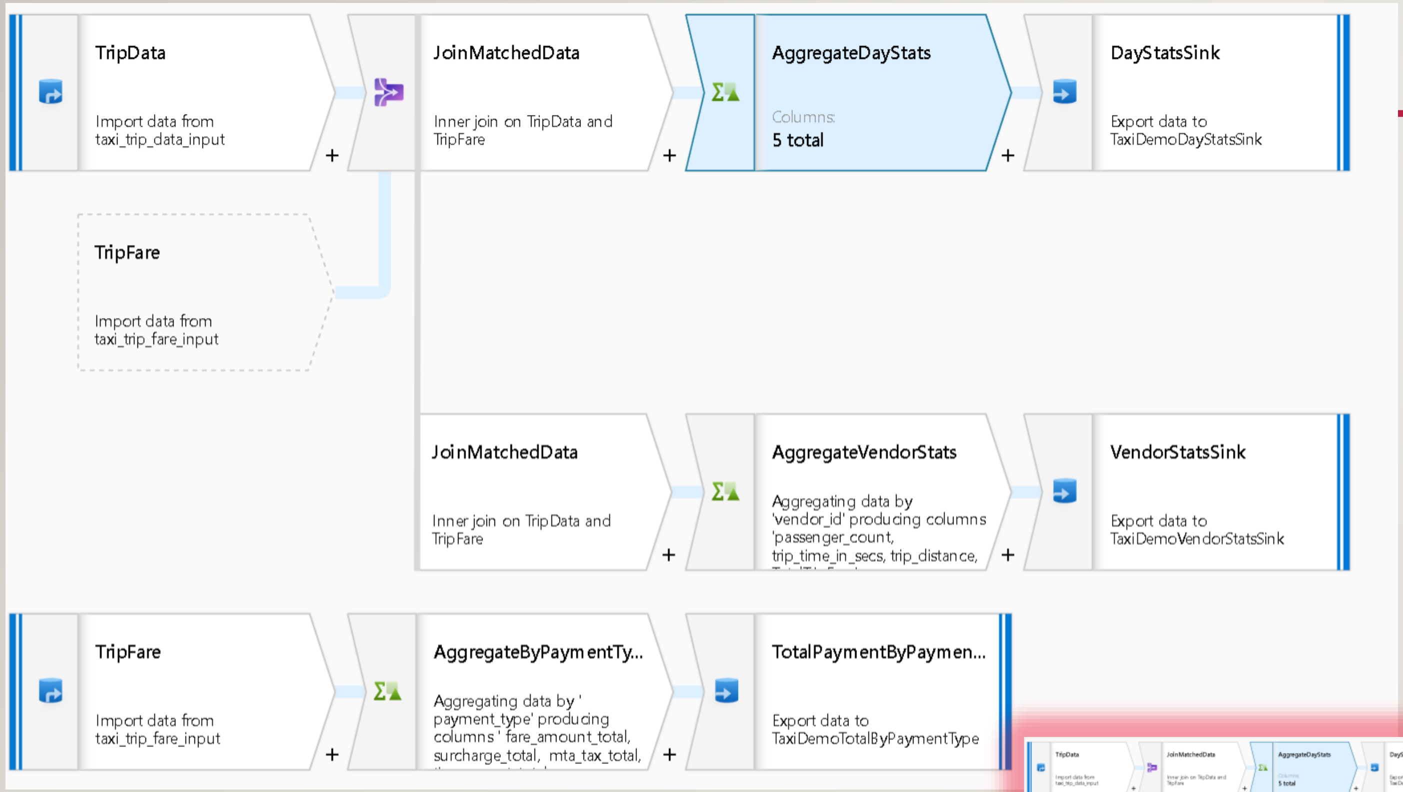
SinkFactSales
Export data to ADW_FactSales

SourceSalesHeader
Import data from SQLDB_SalesHeader

- Classic ETL pattern is easy to build in ADF's code-free Data Flow visual data transformation environment
- Add Aggregate transforms to produce calculations that you store in your analytical database schema
- Use Join transform to combine data from multiple data sources and data streams inside your data flow
- Land your data in your Lake folders or direct to Azure SQL DW

DATA LAKE/DATA SCIENCE

t: @bizdataviz
In/cseferlis


TripData
Import data from taxi_trip_data_input

JoinMatchedData
Inner join on TripData and TripFare

AggregateDayStats
Columns
5 total

DayStatsSink
Export data to TaxiDemoDayStatsSink

TripFare
Import data from taxi_trip_fare_input

JoinMatchedData
Inner join on TripData and TripFare

AggregateVendorStats
Aggregating data by 'vendor_id' producing columns 'passenger_count, trip_time_in_secs, trip_distance,

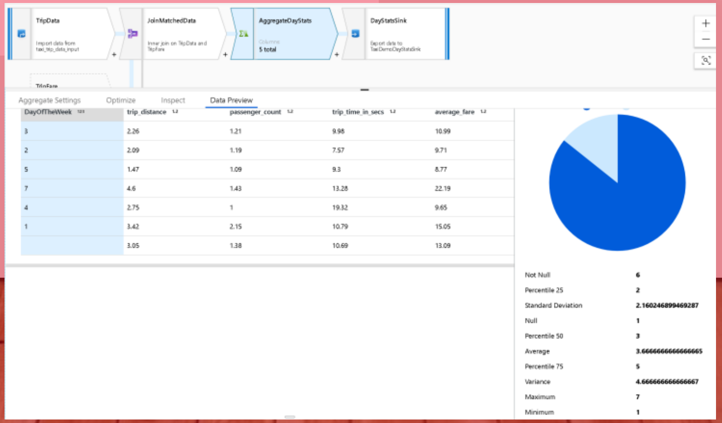
VendorStatsSink
Export data to TaxiDemoVendorStatsSink

TripFare
Import data from taxi_trip_fare_input

AggregateByPaymentTy...
Aggregating data by 'payment_type' producing columns 'fare_amount_total, surcharge_total, mta_tax_total,

TotalPaymentByPaymen...
Export data to TaxiDemoTotalByPaymentType

- ADF supports building visual data transformations against your data directly in Data Lake locations (i.e. Azure Blob Store, Azure Data Lake Store)
- Built-in handling of schema drift for frequent changes in data lake file formats, columns, and data types
- Perform data exploration and data profiling across your data lake in ADF Data Flow with interactive debug data preview



DayOfTheWeek	trip_distance	passenger_count	trip_time_in_secs	average_fare
3	2.26	1.21	9.98	10.99
2	2.09	1.19	7.57	9.71
5	1.47	1.09	9.3	8.77
7	4.6	1.43	13.28	22.19
4	2.75	1	19.32	9.65
1	3.42	2.15	10.79	15.05
	3.05	1.38	10.69	13.09

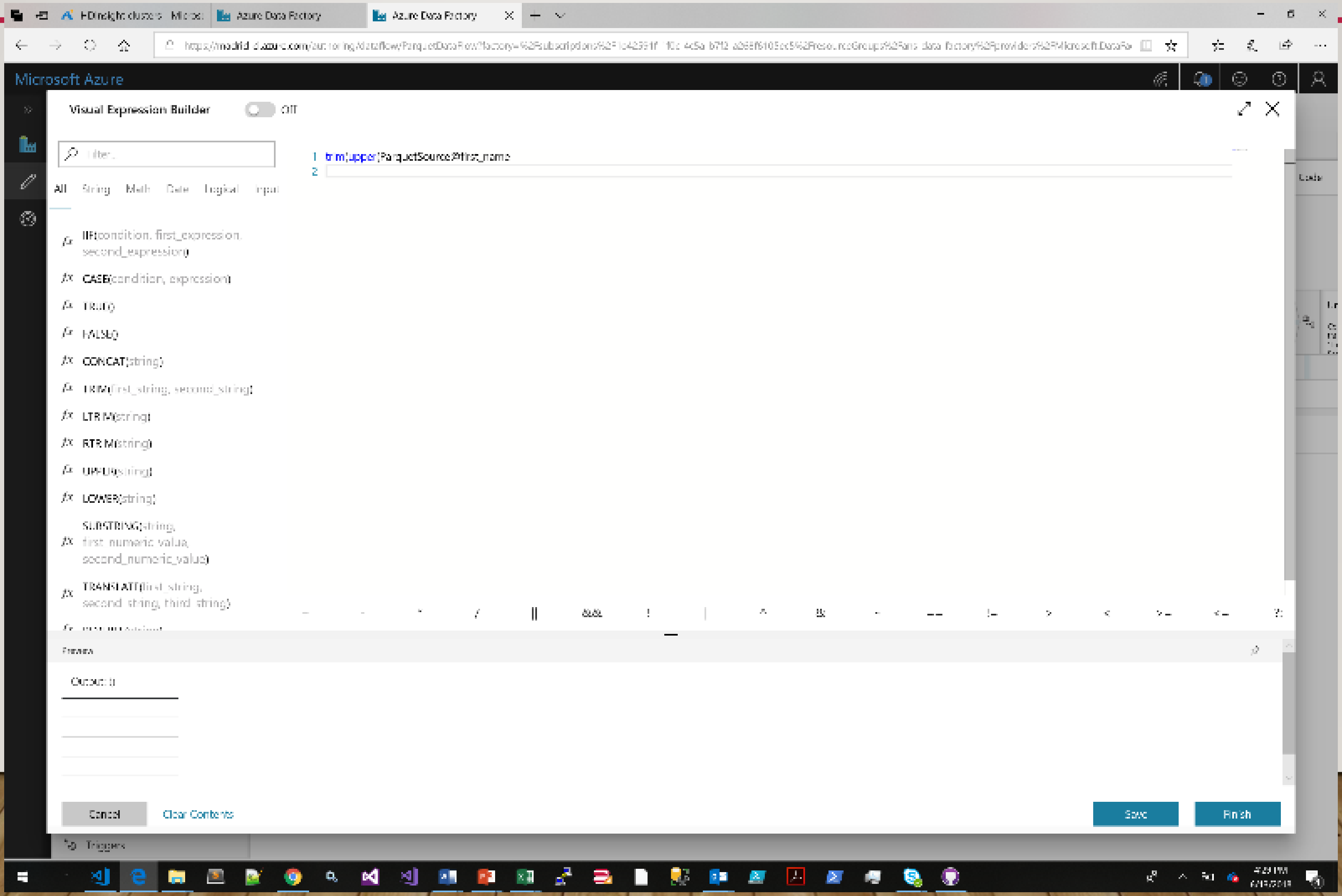
Not Null	6
Percentile 25	2
Standard Deviation	2.160246899469287
Null	1
Percentile 50	3
Average	3.6666666666666665
Percentile 75	5
Variance	4.666666666666667
Maximum	7
Minimum	1

t: @bizdataviz
In/cseferlis

NEW FEATURES IN AZURE DATA FACTORY

INTERACTIVE EXPRESSION BUILDER – BUILD TRANSFORM EXPRESSIONS, NOT CODE

t: @bizdataviz
In/cseferlis

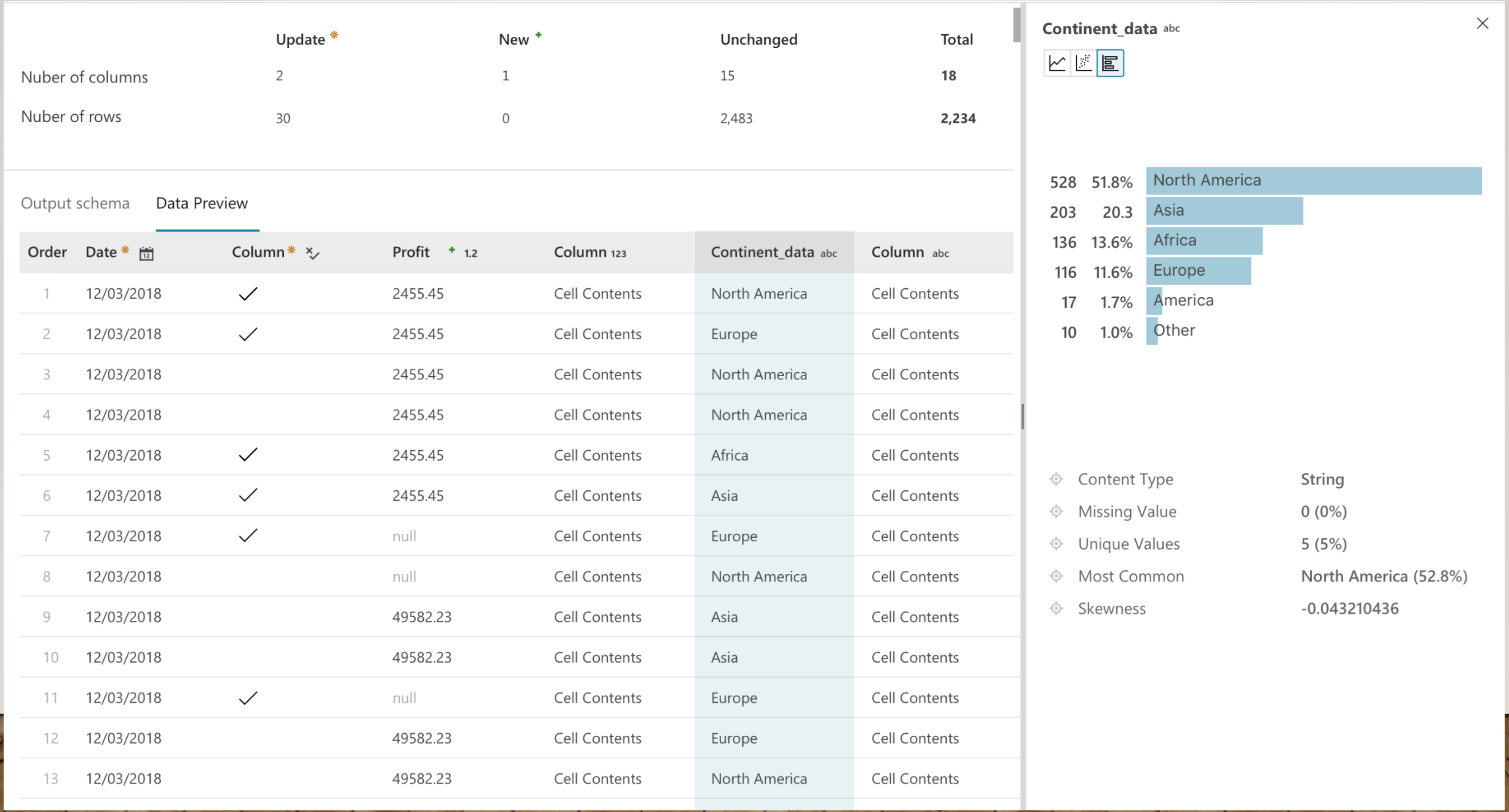


```
trim(upper(ParquetSource@first_name
```

IIF(condition, first_expression, second_expression)
CASE(condition, expression)
TRUE()
FALSE()
CONCAT(string)
TRIM(first_string, second_string)
LTRIM(string)
RTRIM(string)
UPPER(string)
LOWER(string)
SUBSTRING(string, first_numeric_value, second_numeric_value)
TRANSLATE(first_string, second_string, third_string)

DEBUG DATA FLOWS WITH DATA PREVIEW AND DATA SAMPLING

t: @bizdataviz
In/cseferlis



	Update *	New *	Unchanged	Total
Nuber of columns	2	1	15	**18**
Nuber of rows	30	0	2,483	**2,234**

Output schema | Data Preview

Order	Date *	Column *	Profit * 1.2	Column 123	Continent_data abc	Column abc
1	12/03/2018	✓	2455.45	Cell Contents	North America	Cell Contents
2	12/03/2018	✓	2455.45	Cell Contents	Europe	Cell Contents
3	12/03/2018		2455.45	Cell Contents	North America	Cell Contents
4	12/03/2018		2455.45	Cell Contents	North America	Cell Contents
5	12/03/2018	✓	2455.45	Cell Contents	Africa	Cell Contents
6	12/03/2018	✓	2455.45	Cell Contents	Asia	Cell Contents
7	12/03/2018	✓	null	Cell Contents	Europe	Cell Contents
8	12/03/2018		null	Cell Contents	North America	Cell Contents
9	12/03/2018		49582.23	Cell Contents	Asia	Cell Contents
10	12/03/2018		49582.23	Cell Contents	Asia	Cell Contents
11	12/03/2018	✓	null	Cell Contents	Europe	Cell Contents
12	12/03/2018		49582.23	Cell Contents	Europe	Cell Contents
13	12/03/2018		49582.23	Cell Contents	North America	Cell Contents

Continent_data abc

528	51.8%	North America
203	20.3	Asia
136	13.6%	Africa
116	11.6%	Europe
17	1.7%	America
10	1.0%	Other

Content Type	String
Missing Value	0 (0%)
Unique Values	5 (5%)
Most Common	North America (52.8%)
Skewness	-0.043210436

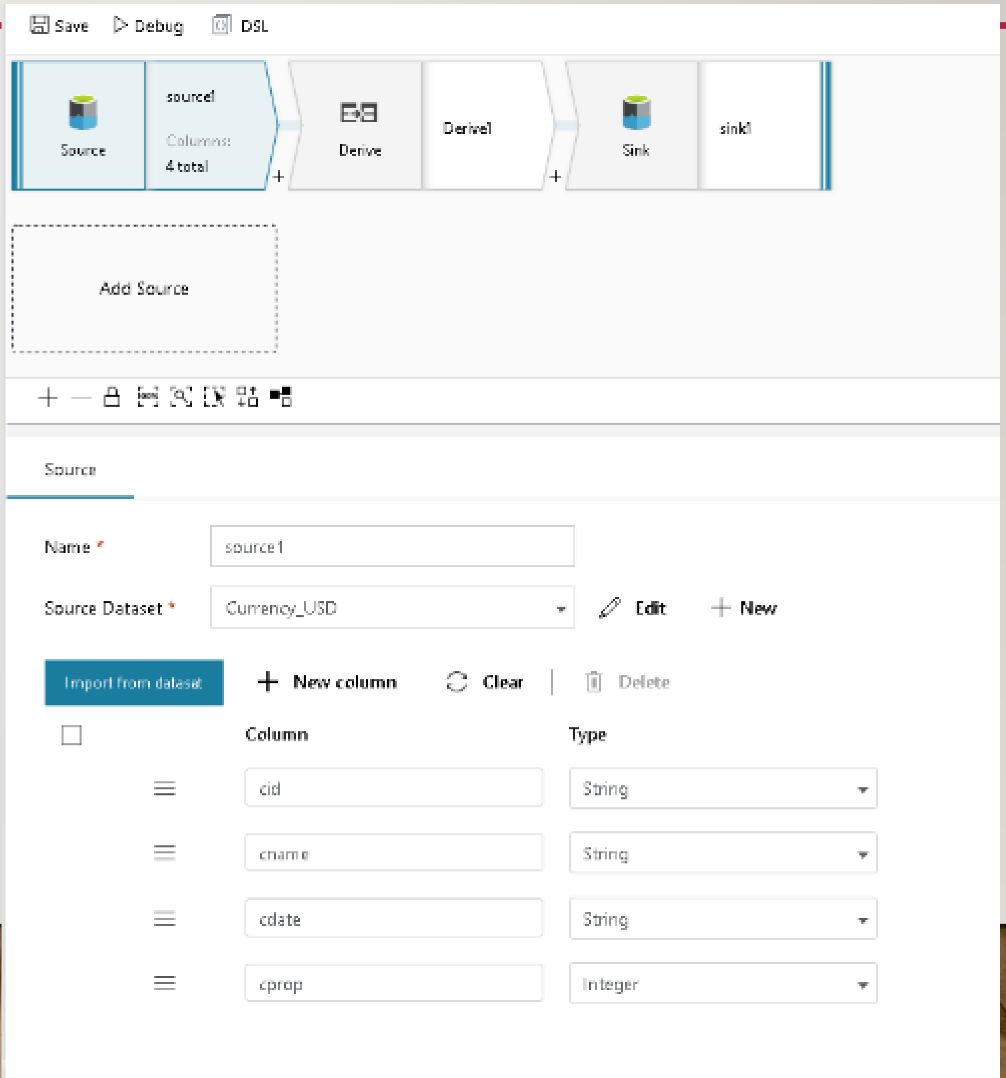
t: @bizdataviz
In/cseferlis

BUILD RESILIENT DATA FLOWS WITH SCHEMA DRIFT HANDLING

HANDLING SOURCE CHANGES

t: @bizdataviz
In/cseferlis

- Data Engineer Defines Source and you take ALL fields from source w/flexible schema



Name * source1

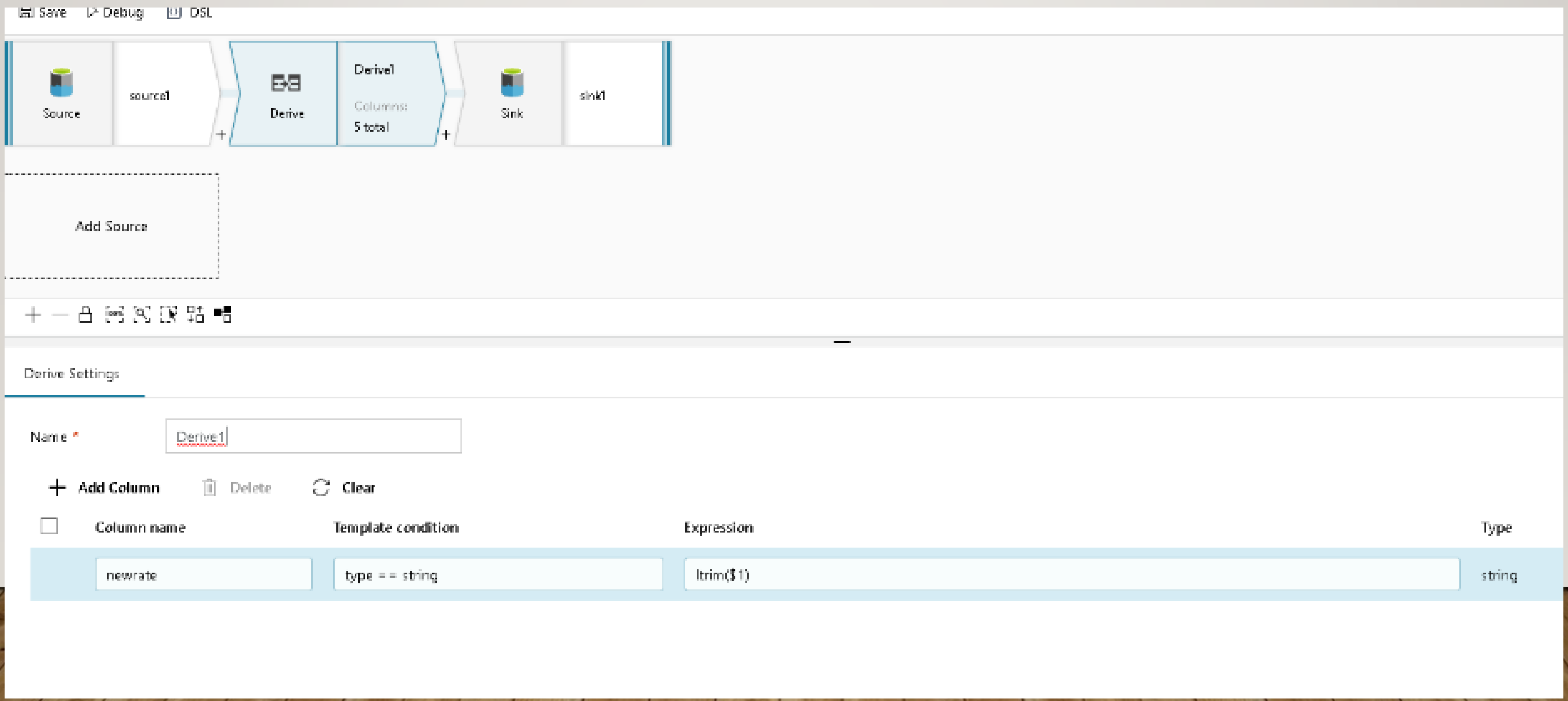
Source Dataset * Currency_USD

Column	Type
cid	String
cname	String
cdate	String
cprop	Integer

HANDLING SOURCE CHANGES

t: @bizdataviz
In/cseferlis

- Data Engineer derives columns using template expression patterns based on name and type matching. No need to define static field names.



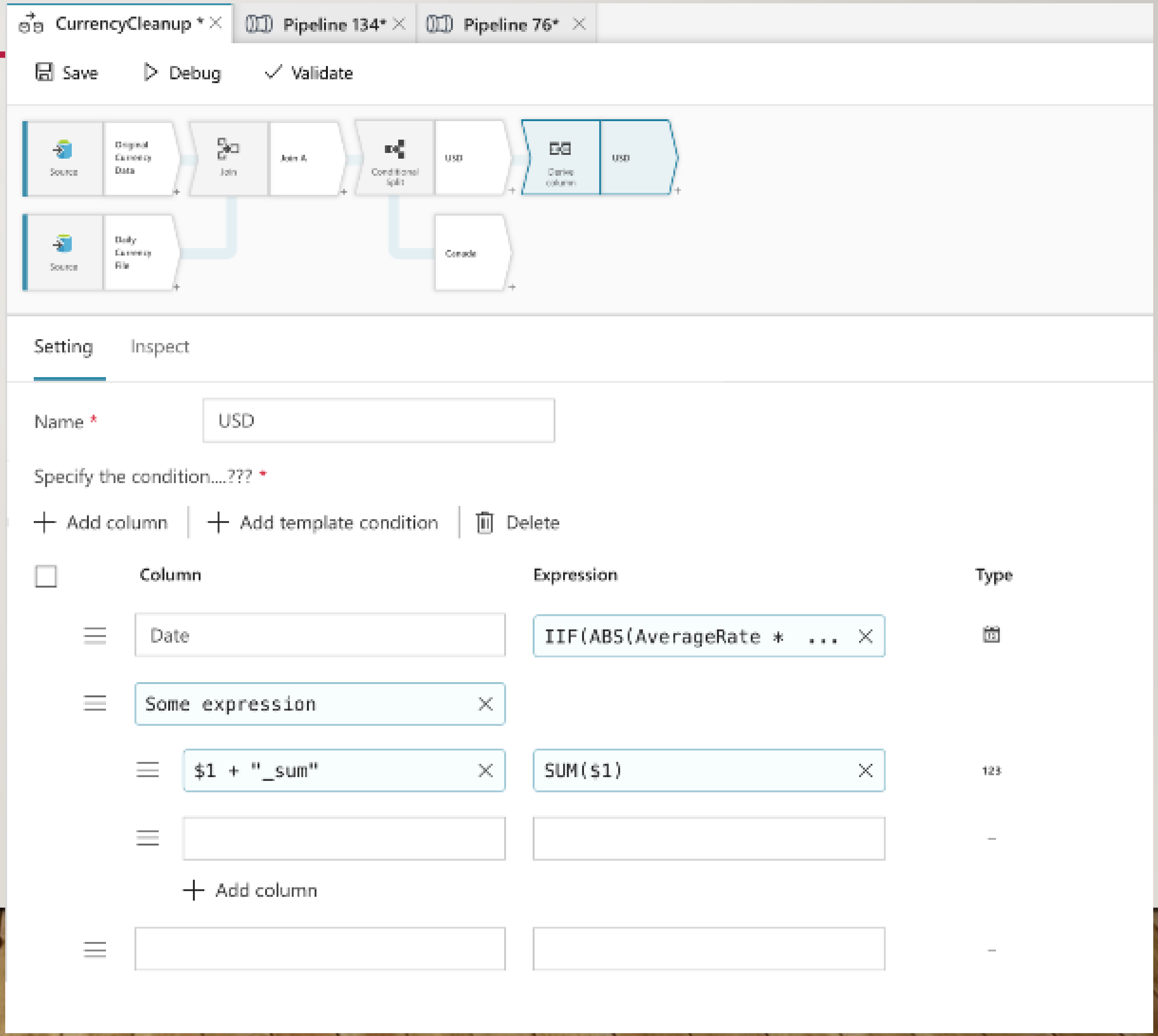
Name * Derive1

Column name	Template condition	Expression	Type
newrate	type == string	ltrim($1)	string

HANDLING SOURCE CHANGES

t: @bizdataviz
In/cseferlis

- Data Engineer derives columns using template expression based on name and type matching. No need to define static field names.



Name * USD

Specify the condition....??? *

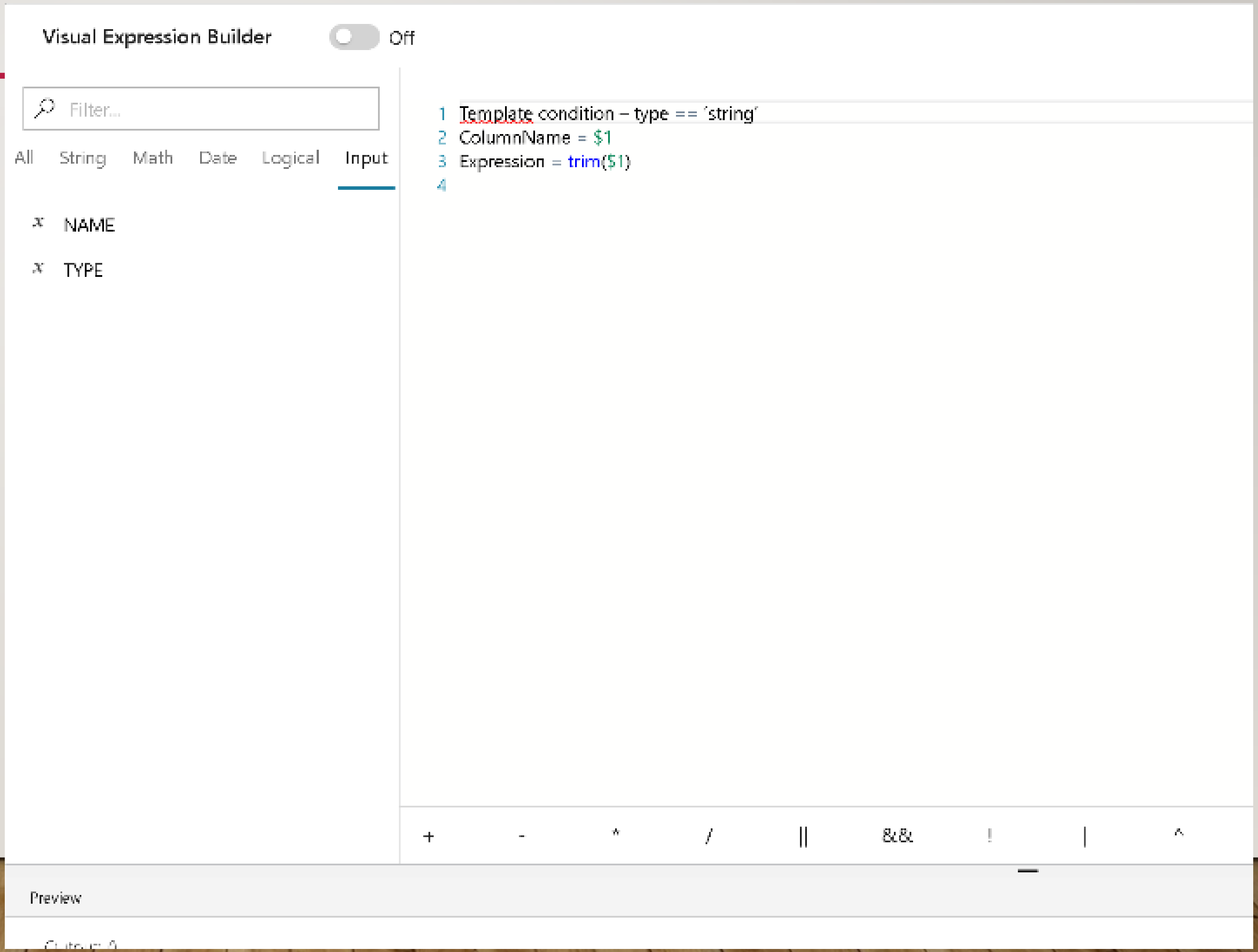
+ Add column | + Add template condition | Delete

Column	Expression	Type
Date	IIF(ABS(AverageRate * ...	
Some expression		
$1 + "_sum"	SUM($1)	123
		-
+ Add column		
		-

HANDLING SOURCE CHANGES

t: @bizdataviz
In/cseferlis

- Data Engineer derives columns using template expression based on name and type matching

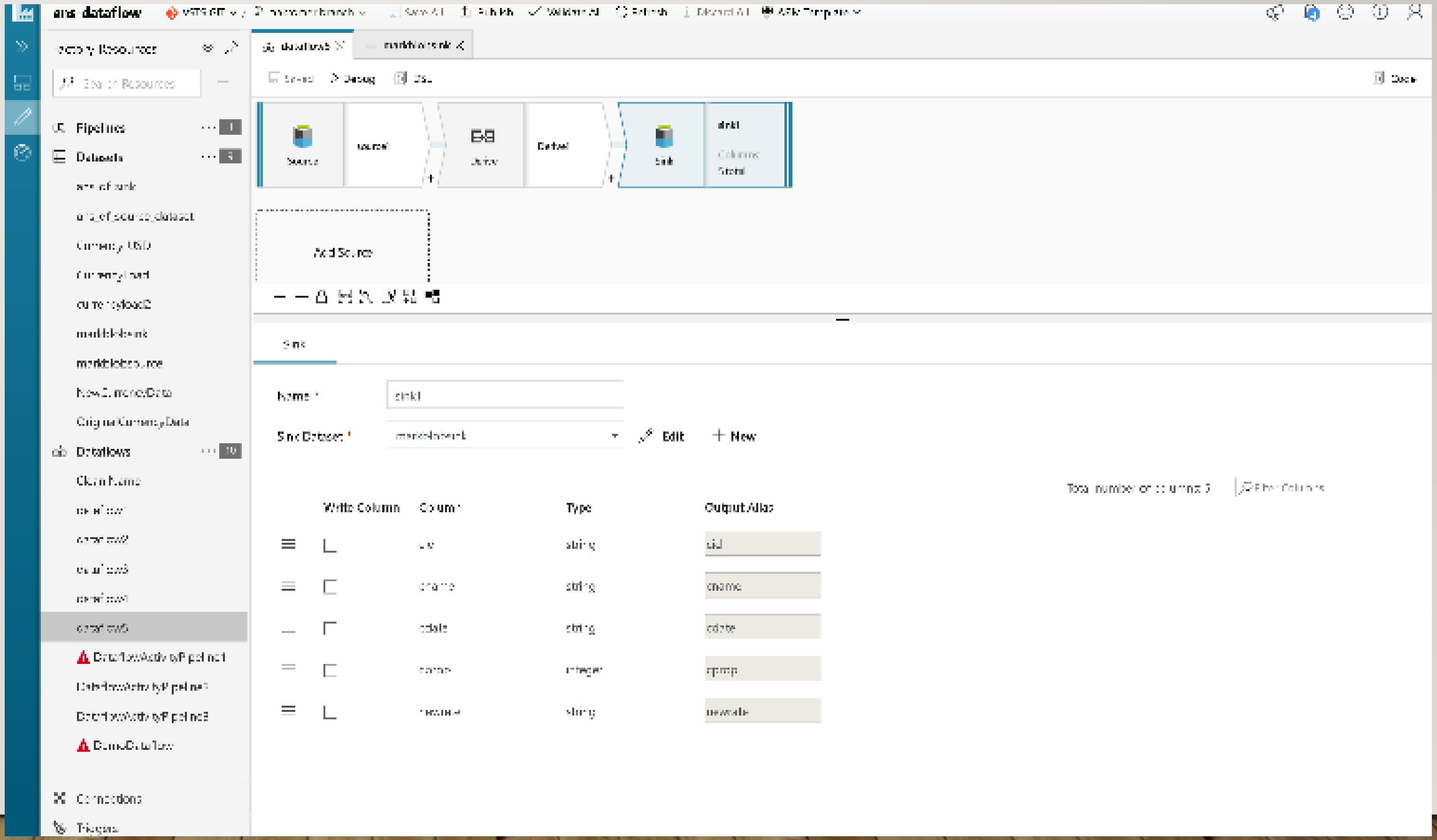


```
Template condition – type == 'string'
ColumnName = $1
Expression = trim($1)
```

HANDLING SOURCE CHANGES

t: @bizdataviz
In/cseferlis

- Sink all incoming fields along with new derived field



Name * sink1

Sink Dataset * markblobsink

Total number of columns: 5

Write Column	Column	Type	Output Alias
	cid	string	cid
	cname	string	cname
	cdate	string	cdate
	cpmp	integer	cpmp
	newdate	string	newdate

IMPORTANT LINKS:

t: @bizdataviz
In/cseferlis

- Sign-up for ADF Data Flow service:
  - http://aka.ms/dataflowpreview
  - Microsoft Azure must whitelist your subscription ID to turn on the feature for you
- GitHub Repository for documentation:
  - https://github.com/kromerm/adfdataflowdocs/

THANK YOU!