Module 3: Data Preparation & Transformation Analytic Functions

Teradata Vantage Analytics Workshop

BASIC

Objectives

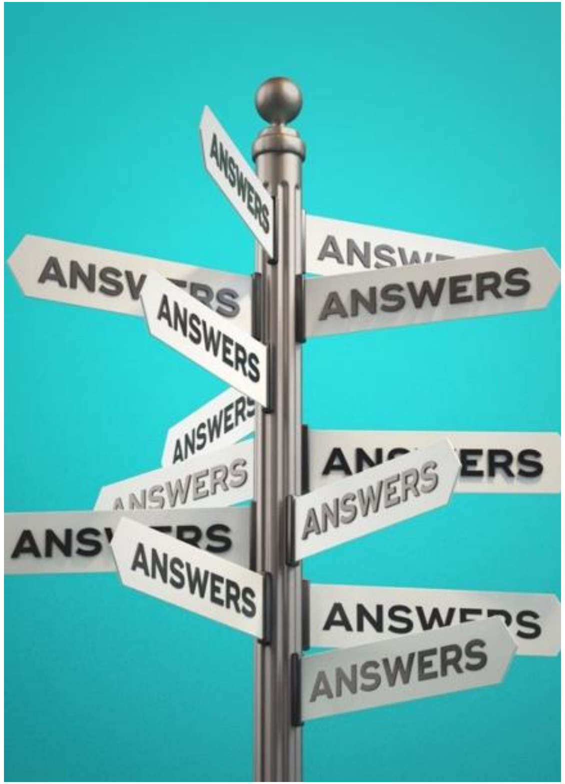
After completing this module, you will be able to:

- Describe Data Science Pipelines
- Write queries using these Teradata Vantage Transformation functions:
  - **Outlier Filtering**
  - **Normalization (Scale Functions)**
  - **Histogram**

For more info go to docs.teradata.com click Teradata Vantage, download: Teradata Vantage Analytic Function Reference Guide

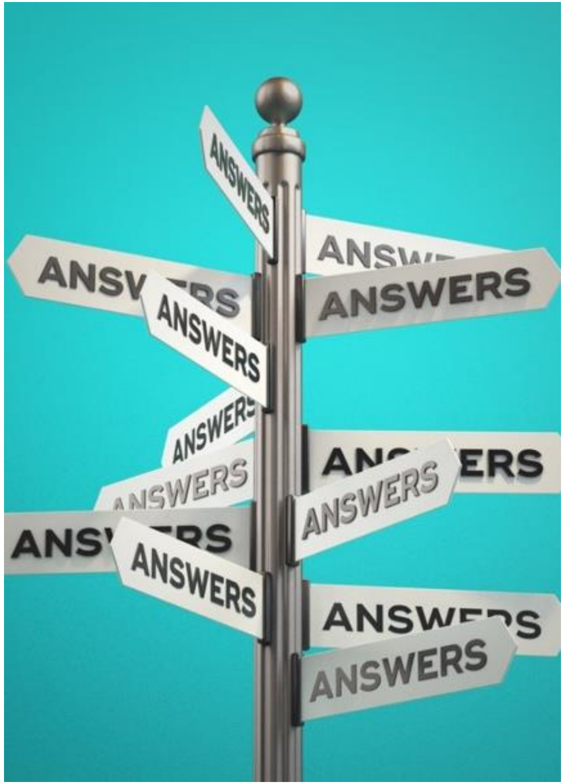
Topics

- Data Science Pipelines
- Outlier Filtering
- Normalization (Scale functions)
- Histogram
- Review & Summary



Current Topic – Data Science Pipelines

- **Data Science Pipelines**
- Outlier Filtering
- Normalization (Scale functions)
- Histogram
- Review & Summary



Possible Pipelines in 17.10

Green – In-DB function
Blue - VAL

Pipeline 1 – Anomaly Detection

Get rows with missing values → Simple Impute → Univariate Statistics → Scale Fit → Outlier Fit

Pipeline 2 – Regression/Classification Pipeline

ConvertTo → Categorical Summary → Chi-Square test → One-hot encoding → Logistic Regression

Pipeline 3 – Clustering Pipeline

Column Summary → Round Columns → NumApply → Row Normalize Fit → K-Means Clustering

Fit and Transform Framework

Operationalizing Data Preparation Pipelines

The popular 'Fit and Transform' methodology is getting introduced as part of the In-DB functions 17.10

- The 'Fit' method performs the calculations necessary to transform the data. For ex, **OutlierFilterFit** will calculate the lower_percentile, upper_percentile, count of rows, and median for the specified input table columns.
- The 'Transform' method then uses the calculations performed by the 'Fit' method and applies the necessary transformation on the data. The calculated values for each column help the **OutlierFilterTransform** function detect outliers in the input table.
- Data Scientists use this 'Fit and Transform' framework on Python and Spark to operationalize their data analysis – fit functions will be applied on the **training data** to gather the necessary metrics and the transform functions apply these metrics on the **test data**.
- In 17.10 release, we are releasing 7 Fit and Transform functions – *Bincode Fit/Transform, OutlierFilterFit/Transform,FunctionFit/Transform, ScaleFit/Transform, PolynomialFit/Transform, RowNormalizeFit/Transform and OnehotEncodingFit/Transform.*

Fit and Transform in Action: OutlierFilterFit and OutlierFilterTransform

The goal of these two functions is to eliminate outliers from the dataset

- **TD_OutlierFilterFit** function calculates the lower_percentile, upper_percentile, count of rows, and median for the specified input table columns
- The calculated values for each column help the **TD_OutlierFilterTransform** function detect outliers in the input table.

```
CREATE TABLE fit_table AS (
  SELECT * FROM TD_OutlierFilterFit (
    ON { table | view | (query) } AS InputTable
    USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
    [ GroupColumns ('group_column') ]
    OutlierMethod ({ 'percentile' | 'tukey' | 'carling' })
    LowerPercentile (min_value)
    UpperPercentile (max_value)
    [ IQRMultiplier (k) ]
    ReplacementValue ({ 'delete' | 'null' | 'median' | replacement_value})
    [ RemoveTail ({ 'both' | 'upper' | 'lower' }) ]
    PercentileMethod ({ 'PercentileCont' | 'PercentileDISC' })
  ) AS alias
) WITH DATA;
```

```
SELECT * FROM TD_OutlierFilterTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  ON { table | view | (query) } AS FitTable DIMENSION
) AS alias;
```

Outlier Detection: OutlierFilterFit and OutlierFilterTransform

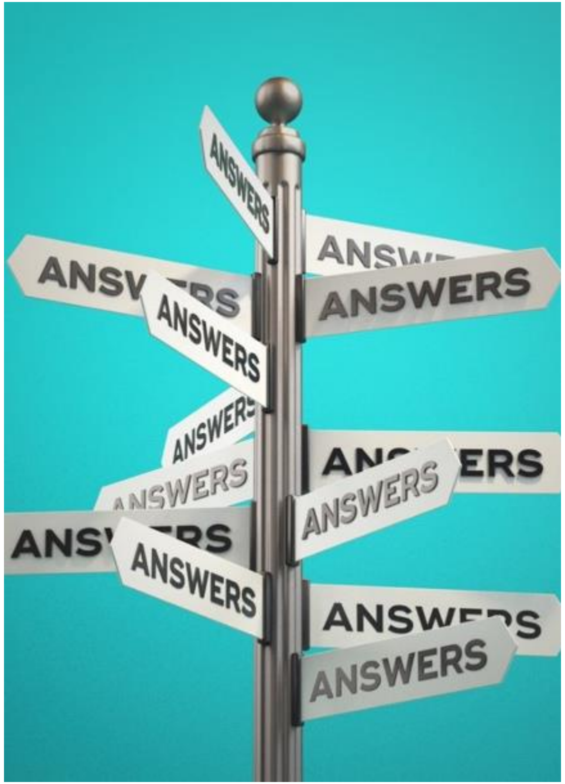
The goal of these two functions is to replace **outliers in TotalQuantity & TotalPrice** with median values (i.e., Zero Vaues and Negative Values majorly)

```
CREATE TABLE OutlierFit_CS as (
select * from TD_OutlierFilterFit(
on customerSegmentGroup as inputTable
Using
TargetColumns('TotalQuantity','TotalPrice')
LowerPercentile(0.03)
UpperPercentile(0.97)
OutlierMethod('Percentile')
PercentileMethod('PercentileCont')
ReplacementValue('Median')
)as dt)With data;
```

```
CREATE TABLE outliertransform_CS as (
select * from TD_OutlierFilterTransform(
On customerSegmentGroup as inputtable
on outlierFit_CS as fittable dimension
)as dt)with data;
```

Current Topic – OutlierFilter

- Data Science Pipelines
- **Outlier Filtering**
- Normalization (Scale functions)
- Histogram
- Review & Summary



OutlierFilter – Description

- In statistics, an Outlier is an observation point that is distant from other observations
- The TD_OutlierFilterFit and TD_OutlierFilterTransform functions filter outliers from a data set, either deleting them or replacing them with a specified value
- The output from this function can serve as input for prediction and clustering functions like GLM, LAR, LinReg, PCA, and KMeans
- The Input table must have one column that contains numeric data to be filtered for outliers, and you must specify its name with the 'TargetColumns' argument

OutlierFilter – Workflow

Input Table → TD_OutlierFit → TD_OutlierTransform → Console or Output Table

Function	Description
TD_OutlierFit	TD_OutlierFilterFit function calculates the lower_percentile, upper_percentile, count of rows, and median for the specified input table columns. The calculated values for each column help the TD_OutlierFilterTransform function detect outliers in the input table.
TD_OutlierTransform	TD_OutlierFilterTransform filters outliers from the input table. The metrics for determining outliers come from TD_OutlierFilterFit output.

OutlierFilter – Syntax


```
CREATE TABLE fit_table AS (
SELECT * FROM  TD_OutlierFilterFit
(ON { table | view | (query) } AS InputTable
USING
TargetColumns ({ 'target_column' | target_column_range }[,...])
[ GroupByColumns ({ 'group_by_column' | group_by_column_range }[,...]) ]
OutlierMethod ({ 'percentile' | 'tukey' | 'carling' })
LowerPercentile (min_value)
UpperPercentile (max_value)
[ IQRMultiplier (k) ]
ReplacementValue ({ 'delete' | 'null' | 'median' | 'replacement_value' })
[ RemoveTail ({ 'both' | 'upper' | 'lower' }) ]
PercentileMethod ({ 'PercentileCont' | 'PercentileDISC' })
) AS alias
) WITH DATA;
```

[ ] indicates Optional argument

Required Arguments

The required arguments for the **OutlierFilterFit** are:

- **TargetColumns** Specify the names of the input table columns that contain numeric data to filter
- **LowerPercentile** Specify a lower range of percentile to use to detect whether the value is an outlier. Value 0 to 1 is supported. For Tukey and Carling, use 0.25 as the lower percentile
- **UpperPercentile** Specify an upper range of percentile to use to detect whether the value is an outlier. Value 0 to 1 is supported. For Tukey and Carling, use 0.75 as the upper percentile
- **PercentileMethod** Specify either the PercentileCont or the PercentileDISC method for calculating the upper and lower percentiles of the input data values

Required Arguments

OutlierMethod Specify one or more of following filtering methods:

Method	Description
percentile	[min_value, max_value].
tukey	Tukey's test: (resistant to extreme values) Outlier is defined as any observation smaller than V1 - k*(V3-V1) or larger than V3 + k*(V3-V1), where V1 and V3 are 25th and 75th percentiles of data and k is specified by IQRMultiplier argument
carling	Carling's modification to Tukey's test: (less affected by sample size than tukey) An outlier is defined as an observation outside the range V2 +- c*(V3 - V1), where V2 is median of data, V1 and V3 are 25th and 75th percentiles of data, and c is constant (which you cannot change)

OutlierMethod ('percentile')

Below, we provide details behind each **OutlierMethod**, assuming all default settings. For this scenario, imagine a dataset of 100 rows of temperatures, comprised of sequential integers ranging from 1 to 100. Assume this same dataset for next few pages

- percentile: If LESS THAN lower quantile or GREATER THAN upper quantile, it's an Outlier. If no OutlierMethod is specified, the function will use percentile. Furthermore, the default values for percentile = (5, 95), meaning in essence, that anything below the bottom 5% threshold and above the top 5% threshold of records will be identified as Outliers
- In our dataset, nine records are identified as outliers. Since the default is (5, 95), any record below the 5th or above the 95th percentiles will be flagged. This equates to temperatures 1 through 5 and 97 through 100 as Outliers

1	26	51	76
2	27	52	77
3	28	53	78
4	29	54	79
5	30	55	80
6	31	56	81
7	32	57	82
8	33	58	83
9	34	59	84
10	35	60	85
11	36	61	86
12	37	62	87
13	38	63	88
14	39	64	89
15	40	65	90
16	41	66	91
17	42	67	92
18	43	68	93
19	44	69	94
20	45	70	95
21	46	71	96
22	47	72	97
23	48	73	98
24	49	74	99
25	50	75	100

See the **Notes** page for syntax to create and populate such an example table if you wish to experiment with the various OutlierMethod arguments

OutlierMethod ('tukey')

- tukey: Outlier is defined as any observation smaller than V1 - k*(V3-V1) or larger than V3 + k*(V3-V1), where V1 and V3 = 25th and 75th percentiles of data and k is specified by IQRMultiplier argument (default value 1.5)
- In our dataset, zero records are identified as outliers. In our dataset, the 25th and 75th percentile values are 25 and 75, respectively
  - **V1 - k*(V3-V1)** = 25 - 1.5*(75 - 25) = **-50**, therefore any temperature less than this would be an Outlier for lower end
  - **V3 + k*(V3-V1)** = 75 + 1.5*(75 - 25) = **150**, therefore any temperature greater than this would be an Outlier for higher end

OutlierMethod ('carling')

- carling: An outlier is defined as an observation outside the range V2 +/- c*(V3 - V1), where V2 = median of data, V1 and V3 = 25th and 75th percentiles of data, and c is constant (which you cannot change). Note that c is approximately 2.237624

- In our dataset, zero records are identified as outliers. In our dataset, the 25th and 75th percentile values are 25 and 75, respectively. Our median is 51.5
  - **V2 - c*(V3 - V1) =** 51.5 - 2.237624*(75 – 25) = -60.381200, therefore any temperature less than this would be an Outlier for lower end
  - **V2 + c*(V3 - V1) =** 51.5 + 2.237624*(75 – 25) = 163.381200, therefore any temperature greater than this would be an Outlier for higher end

Required Arguments

ReplacementValue: Specify how to filter Outliers

Option	Description
delete	Do not copy row to output table.
null	Copy row to output table, replacing each outlier with NULL.
median	Copy row to output table, replacing each outlier with median value for its group.
replacement_value (numeric)	Copy row to output table, replacing each outlier with replacement_value.

Optional Arguments

- **GroupColumns:** Specify the name of the InputTable column by which to group the input data. Default behavior: Function does not group input data.
- **IQRMultiplier:** Specify multiplier of interquartile range for 'tukey' filtering. Default: 1.5
- **RemoveTail:** Specify whether to remove the upper tail, the lower tail, or both. Default: 'both'

Tails are distributions more likely to have values away from the mean/median than 'typical'

Lab 1: OutlierFilterFit (tukey and delete)


```
CREATE TABLE outlier_fit AS (
SELECT * FROM TD_OutlierFilterFit (
ON TRNG_TDU_TD01.input_tbl2 AS
InputTable
USING
TargetColumns ('val')
GroupColumns ('var_name')
LowerPercentile (0.25)
UpperPercentile (0.75)
OutlierMethod ('tukey')
IQRMultiplier(1.5)
ReplacementValue ('delete')
RemoveTail ('both')
PercentileMethod ('PercentileCont')
) AS dt
) WITH DATA;
```

Look for Outliers in this column

Delete Outlier rows in Output table

These appear to be Outliers

Input

rownum	groupcol	var_name	val
1	P	Pressure	100
2	P	Pressure	125
3	P	Pressure	150
4	P	Pressure	175
5	P	Pressure	9999
6	T	Temperature	350
7	T	Temperature	375
8	T	Temperature	400
9	T	Temperature	425
10	T	Temperature	-9999

Note '**GroupByColumns**' argument. Range of Outlier values will be Partitioned by this column prior to calculating Outliers

Lab 2: OutlierFilterTransform (tukey and delete)

teradata.


```
SELECT * FROM OutlierFilter
(ON input_tbl2 AS InputTable
 OUT TABLE OutputTable (output_tbl2)
 USING
 TargetColumns('val')
 OutlierMethod('tukey')
 IQRMultiplier(1.5)
 ReplacementValue('delete')
 GroupByColumns('var_name')
) as dt;
```

Look for Outliers in this column

Delete Outlier rows in Output table

message
Output tables created successfully

```
SELECT * FROM output_tbl2
ORDER BY rownum;
```

Outlier rows gone

Output

rownum	groupcol	var_name	val
1	P	Pressure	100
3	P	Pressure	150
4	P	Pressure	175
2	P	Pressure	125
8	T	Temperature	400
7	T	Temperature	375
6	T	Temperature	350
9	T	Temperature	425

Lab 3a: OutlierFilterFit (Percentile and Null)


```
CREATE TABLE outlier_fit AS (
SELECT * FROM TD_OutlierFilterFit (
ON TRNG_TDU_TD01.ville_pressuredata
AS InputTable
USING
TargetColumns ('pressure_mbar')
GroupColumns ('city')
LowerPercentile (0.1)
UpperPercentile (0.95)
OutlierMethod ('Percentile')
ReplacementValue ('null')
RemoveTail ('upper')
PercentileMethod ('PercentileCont')
) AS dt
) WITH DATA;
```

If outside these %, replace value

Use % to make decision

Replace Outlier values with NULL

Only remove upper tail

Input

sn	city	ts	pressure_mbar
1	ashville	20100101 00:00	1020.5
2	ashville	20100101 01:00	9000
3	ashville	20100101 02:00	1020
4	ashville	20100101 03:00	10000
5	ashville	20100101 04:00	1020.2
6	ashville	20100101 05:00	1020
7	ashville	20100101 06:00	1020.3
8	ashville	20100101 07:00	1020.8
9	ashville	20100101 08:00	1021.3
10	ashville	20100101 09:00	1021.7
11	ashville	20100101 10:00	1022.1
12	ashville	20100101 11:00	1022
13	ashville	20100101 12:00	1021.1
14	ashville	20100101 13:00	1020
15	ashville	20100101 14:00	1019.3
16	ashville	20100101 15:00	1019
17	ashville	20100101 16:00	1019.2
18	ashville	20100101 17:00	1019.6
19	ashville	20100101 18:00	1020.1
20	ashville	20100101 19:00	1020.6
21	ashville	20100101 20:00	1020.9
22	ashville	20100101 21:00	1021.1
23	ashville	20100101 22:00	1021
24	ashville	20100101 23:00	1020.9

Lab 3b: OutlierFilterTransform (Percentile and Null)

teradata.


```
CREATE TABLE of_output1 AS (
SELECT * FROM
TD_OutlierFilterTransform (
ON TRNG_TDU_TD01.ville_pressuredata
AS InputTable PARTITION BY city
ON outlier_fit AS FitTable PARTITION
BY city
) AS dt
) WITH DATA;
```

Output

sn	city	ts	pressure_mbar
1	ashville	20100101 00:00	1020.5
2	ashville	20100101 01:00	
3	ashville	20100101 02:00	1020
4	ashville	20100101 03:00	
5	ashville	20100101 04:00	1020.2
6	ashville	20100101 05:00	1020
7	ashville	20100101 06:00	1020.3
8	ashville	20100101 07:00	1020.8
9	ashville	20100101 08:00	1021.3
10	ashville	20100101 09:00	1021.7
11	ashville	20100101 10:00	1022.1
12	ashville	20100101 11:00	1022
13	ashville	20100101 12:00	1021.1
14	ashville	20100101 13:00	1020
15	ashville	20100101 14:00	1019.3
16	ashville	20100101 15:00	1019
17	ashville	20100101 16:00	1019.2
18	ashville	20100101 17:00	1019.6
19	ashville	20100101 18:00	1020.1
20	ashville	20100101 19:00	1020.6
21	ashville	20100101 20:00	1020.9
22	ashville	20100101 21:00	1021.1
23	ashville	20100101 22:00	1021
24	ashville	20100101 23:00	1020.9

teradata.

Lab 4a: OutlierFilterFit (carling and median)

```
CREATE TABLE outlier_fit AS (
SELECT * FROM TD_OutlierFilterFit (
ON TRNG_TDU_TD01.ville_pressuredata AS
InputTable
USING
TargetColumns ('pressure_mbar')
GroupColumns ('city')
LowerPercentile (0.25)
UpperPercentile (0.75)
OutlierMethod ('carling')
IQRMultiplier(1.5)
ReplacementValue ('median')
RemoveTail ('both')
PercentileMethod ('PercentileCont')
) AS dt
) WITH DATA;
```

Replace Outlier values with median

Input

sn	city	ts	pressure_mbar
1	ashville	20100101 00:00	1020.5
2	ashville	20100101 01:00	9000
3	ashville	20100101 02:00	1020
4	ashville	20100101 03:00	10000
5	ashville	20100101 04:00	1020.2
6	ashville	20100101 05:00	1020
7	ashville	20100101 06:00	1020.3
8	ashville	20100101 07:00	1020.8
9	ashville	20100101 08:00	1021.3
10	ashville	20100101 09:00	1021.7
	ashville	20100101 10:00	1022.1
	ashville	20100101 11:00	1022
13	ashville	20100101 12:00	1021.1
14	ashville	20100101 13:00	1020
15	ashville	20100101 14:00	1019.3
16	ashville	20100101 15:00	1019
17	ashville	20100101 16:00	1019.2
18	ashville	20100101 17:00	1019.6
19	ashville	20100101 18:00	1020.1
20	ashville	20100101 19:00	1020.6
21	ashville	20100101 20:00	1020.9
22	ashville	20100101 21:00	1021.1
23	ashville	20100101 22:00	1021
24	ashville	20100101 23:00	1020.9

Lab 4b: OutlierFilterTransform (carling and median)

teradata.


```
CREATE TABLE of_output1 AS (
SELECT * FROM TD_OutlierFilterTransform (
ON TRNG_TDU_TD01.ville_pressuredata AS
InputTable PARTITION BY city
ON outlier_fit AS FitTable PARTITION BY city
) AS dt
) WITH DATA;

CREATE TABLE outlier_output1 AS (
SELECT m.sn, m.city, m.ts, m.pressure_mbar as
outlier_mbar, p.pressure_mbar
from TRNG_TDU_TD01.ville_pressuredata AS m
JOIN of_output1 AS p
on m.sn = p.sn
where m.pressure_mbar NE p.pressure_mbar
) WITH DATA;
```

Output

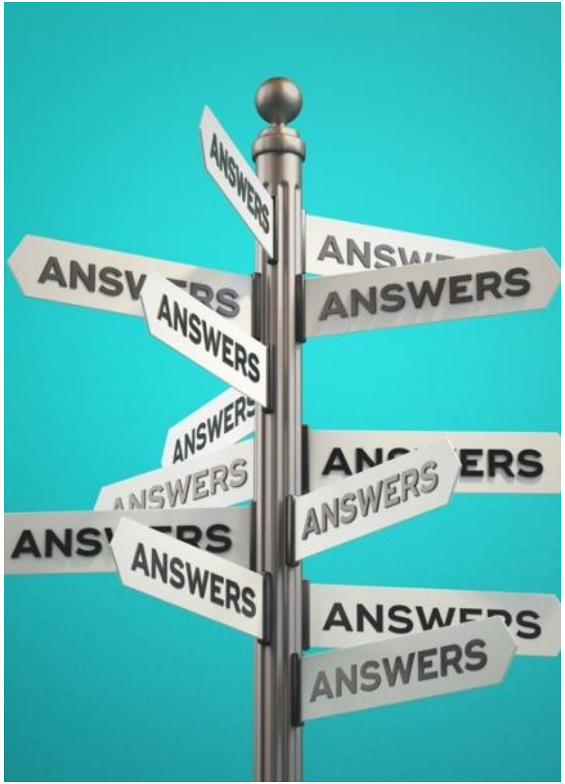
sn	city	ts	pressure_mbar
1	ashville	20100101 00:00	1020.5
2	ashville	20100101 01:00	1020.7
3	ashville	20100101 02:00	1020
4	ashville	20100101 03:00	1020.7
5	ashville	20100101 04:00	1020.2
6	ashville	20100101 05:00	1020
7	ashville	20100101 06:00	1020.3
8	ashville	20100101 07:00	1020.8
9	ashville	20100101 08:00	1021.3
10	ashville	20100101 09:00	1021.7
11	ashville	20100101 10:00	1022.1
12	ashville	20100101 11:00	1022

Outlier Output

sn	city	ts	outlier_mbar	pressure_mbar
2	ashville	20100101 01:00	9000	1020.7
4	ashville	20100101 03:00	10000	1020.7
26	greenville	20100101 01:00	9000	1020.7
28	greenville	20100101 03:00	10000	1020.7
50	brownsville	20100101 01:00	9000	1020.65
52	brownsville	20100101 03:00	10000	1020.65
74	nashville	20100101 01:00	9000	1020.5
76	nashville	20100101 03:00	10000	1020.5
98	knoxville	20100101 01:00	9000	1020.55
100	knoxville	20100101 03:00	10000	1020.55

Current Topic – Normalization

- Data Science Pipelines
- Outlier Filtering
- **Normalization (Scale functions)**
- Histogram
- Review & Summary



Description – Normalization (Scale and ScaleMap)

- In statistics and applications of statistics, Normalization can have a range of meanings. In the simplest cases, normalization of ratings means adjusting values measured on different scales to a notionally common scale, often prior to averaging
- In more complicated cases, Normalization may refer to more sophisticated adjustments where the intention is to bring the entire probability distributions of adjusted values into alignment. In the case of normalization of scores in educational assessment, there may be an intention to align distributions to a normal distribution

https://www.google.com/search?q=how+to+normalize+data&rlz=1C1GCEV_enUS844US844&oq=how+to+normalize+&aqs=chrome.0.0j69i57j0l6.3628j0j4&sourceid=chrome&ie=UTF-8#kpvalbx=_d8uNXtq2BpqxtQbnn5qACQ38

High-bias ML algorithms (like Linear Regression, Logistic Regression, Kmeans) can underfit Model; i.e., can't make accurate Models on your TRAIN set. Normalization can minimize this tendency

Why Use Normalization?

- The black box answer is you can’t train models when your features have different ranges (1-5 vs 1-5000)
- In essence, Normalization is done to have the same range of values for each of the inputs to the Model. This can guarantee stable convergence of weight and biases

id	room area	room number	height	price
1	100	3	2.6	200,000
2	150	4	3	300,000
...	...	...	...	...

If one of the features has a broad range of values, the distance will be governed by this particular feature

Range in column 'room area' is 50 and it is significantly larger than the range in column 'height'. So we can't compare them directly

Workflow – Normalization

Input Table → TD_ScaleFit → TD_ScaleTransform → Console or Output Table

Function	Description
TD_ScaleFit	TD_ScaleFit outputs a table of statistics to input to TD_ScaleTransform, which scales specified input table columns.
TD_ScaleTransform	TD_ScaleTransform scales specified input table columns, using TD_ScaleFit output.

Syntax – TD_ScaleFit


```
SELECT * FROM TD_ScaleFit (
ON { table | view | (query) } AS InputTable
[ PARTITION BY ANY [ ORDER BY order_column ] ]
[ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
USING
TargetColumns ( { 'target_column' | target_column_range }[,...] )
ScaleMethod ('scale_method' [,...])
[ Multiplier ('multiplier' [,...]) ]
[ Intercept ('intercept' [,...]) ]
[ GlobalScale ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ MissValue ({ 'KEEP' | 'ZERO' | 'LOCATION' }) ]
) AS alias;
```

Arguments – TD_ScaleFit

- **TargetColumns:** Specify the names of input table columns for which to calculate statistics. The columns must contain numeric values
- **MissValue:** [Optional] Specify how the function is to process **NULL** values in input, as follows:

Option	Description
Keep(Default)	Keep **NULL** values
Zero	Replace each **NULL** value with zero
Location	Replace each **NULL** value with its location value. **Note:** Location definition varies by Method; e.g., for Method "midrange", defined as (max X + min X) / 2

Arguments – Scale

- **GlobalScale:** [Optional] Specify whether all input columns scaled to same location and scale. Default: 'false' (scale each target column separately)
- **TargetColumns:** Specify the names of the InputTable columns for which to output statistics. The columns must contain numeric data in the range (-1e308, 1e308)
- **OutputTable:** [Optional] Specify a name for the output table
- **Multiplier:** [Optional] Specify either one multiplier for all target columns or one multiplier for each target_column. The function uses the nth multiplier for the nth target_column. Default: '1'

Arguments – Scale (cont.)

Intercept: [Optional] Specify one or more addition factors incrementing the scaled results - intercept in the following formula:

X' = intercept + multiplier * (X - location)/scale

If you specify only one intercept, it applies to all columns specified by the TargetColumns argument. If you specify multiple addition factors, each intercept applies to the corresponding input column. This is the syntax of intercept:

[-]{number | min | mean | max }

where min, mean, and max are the scaled global minimum, maximum, mean values of the corresponding columns. This is the formula for computing the scaled global minimum: scaledmin = (minX - location)/scale Default: intercept is 0

Arguments – TD_ScaleFit

ScaleMethod: Specify one or more statistical methods to Scale the data set

Method	Location	Scale
mean	Xmean	1
sum	0	Σ X
ustd	0	Standard deviation, calculated according to biased estimator of variance
std	Xmean	Standard deviation, calculated according to unbiased estimator of variance
range	minx	maxX – minx
midrange	(maxx+minx)/2	(maxX - minX)/2
maxabs	0	Maximum of absolute value of X
rescale		See table after RESCALE syntax.

Rescale

RESCALE ({ lb=lower_bound | ub=upper_bound | lb=lower_bound, ub=upper_bound })

ub=upper_bound })r more statistical methods to Scale the data set

	Location	Scale
Lower bound only	Xmin - lower_bound	1
Upper bound only	Xmax - upper_bound	1
Lower and upper bounds	Xmin - (lower_bound/ (upper_bound - lower_bound))	(Xmax - Xmin)/(upper_bound - lower_bound)

Method ('mean')

Below, we provide examples behind each Method, assuming all default settings. For this scenario, imagine a dataset of 100 rows of temperatures, comprised of sequential integers ranging from 1 to 100. <u>Assume this same dataset for the next several pages</u>

- mean: *datapoint value – AVG datapoint value*
  (If AVG datapoint value = ***50.5****, then …)*

Original value		New value
1 – 50.5	=	*-49.5*
25 – 50.5	=	*-25.5*
50 – 50.5	=	*-0.5*
75 – 50.5	=	*24.5*
100 – 50.5	=	*49.5*

The mean method shows how far away the value to scale is from mean. Negative numbers are less than the mean, while positive numbers are greater than the mean. A value of 0 signifies the datapoint is the same value as the mean.

1	26	51	76
2	27	52	77
3	28	53	78
4	29	54	79
5	30	55	80
6	31	56	81
7	32	57	82
8	33	58	83
9	34	59	84
10	35	60	85
11	36	61	86
12	37	62	87
13	38	63	88
14	39	64	89
15	40	65	90
16	41	66	91
17	42	67	92
18	43	68	93
19	44	69	94
20	45	70	95
21	46	71	96
22	47	72	97
23	48	73	98
24	49	74	99
25	50	75	100

See the **Notes** page for syntax to create and populate such an example table if you wish experiment with the various Method arguments

Method ('sum')

sum: *datapoint value / SUM of all datapoint values*
(If SUM of all datapoint values = 5050, then …)

Original value	New value
1 / 5050	= *0.000198*
25 / 5050	= *0.004950*
50 / 5050	= *0.009901*
75 / 5050	= *0.014851*
100 / 5050	= *0.019802*

Note: The sum method shows what portion the value to scale is of the SUM of all datapoint values. All scaled values SUM to 1

Method ('std')

std: (*datapoint value – AVG datapoint value) / standard deviation sample*

- *If AVG datapoint value = 50.5*
- *And Standard deviation sample = 29.01*

Original value	New value

- *(1 – 50.5) / 29.01 = -1.706*
- *(25 – 50.5) / 29.01 = -0.878*
- *(50 – 50.5) / 29.01 = -0.017*
- *(75 – 50.5) / 29.01 = 0.844*
- *(100 – 50.5) / 29.01 = 1.706*

Note: The std method shows how many standard deviations away from the mean the datapoint is:

- A negative number signifies value is less than the mean
- A positive number signifies it is greater than the mean
- A value of 0 signifies it is the same value as the mean

Method ('range')

range: (*datapoint value – MIN datapoint value) / range*

- *If MIN datapoint value = 1*
- *And Range = 99, then*

Original value	New value

- *(1 – 1) / 99 = 0*
- *(25 – 1) / 99 = 0.242*
- *(50 – 1) / 99 = 0.495*
- *(75 – 1) / 99 = 0.747*
- *(100 – 1) / 99 = 1*

Note: The range method scales/normalizes all values to be between 0 and 1. The MIN value will be scaled to 0 and the MAX value will be scaled to 1. The AVG scaled value is 0.5

Method ('midrange')

midrange: (*datapoint value – location*) / *scale*

- *location* = (*MAX* + *MIN*) / 2 = (100 + 1) / 2 = 50.5
- *scale* = (*MAX* – *MIN*) / 2 = (100 – 1) / 2 = 49.5

Original value		New value
(1 – 50.5) / 49.5	=	-1
(25 – 50.5) / 49.5	=	-0.515
(50 – 50.5) / 49.5	=	-0.010
(75 – 50.5) / 49.5	=	0.495
(100 – 50.5) / 49.5	=	1

Note: The midrange method scales/normalizes all values to be between -1 and 1. The MIN value will be scaled to -1 and the MAX value will be scaled to 1. Scaled values add up to 0.

Method ('maxabs')

maxabs: *datapoint value / MAX datapoint value*

(If MAX datapoint value = 100 , then …)

Original value	New value

- *1 / 100 = 0.01*
- *25 / 100 = 0.25*
- *50 / 100 = 0.5*
- *75 / 100 = 0.75*
- *100 / 100 = 1*

Note: The maxabs method calculates what portion the datapoint value is of MAX value. Scaled values SUM to non-scaled AVG.

Lab 5a: Data We'll be Using

The Input variables are as follows

```
SELECT * FROM scale_housing;
```

0	1	2	3	4	5	6
types	id	price	lotsize	bedrooms	bathrms	stories
classic	1	42000	5850	3	1	2
classic	2		4000	2	1	1
classic	3	49500	3060	3	1	1
classic	4	60500	6650	3	1	2
classic	5	61000	6360	2	1	1
bungalow	6	66000	4160	3	1	1
bungalow	7	66000	3880		2	2
bungalow	8	69000	4160	3	1	3
bungalow	9	83800	4800	3	1	1
bungalow	10	88500	5500	3	2	4

Lab 5b: TD_ScaleFit Method('midrange')

```
SELECT * FROM TD_ScaleFit (
ON TRNG_TDU_TD01.scale_housing AS InputTable
OUT PERMANENT TABLE OutputTable (scaleFitOut)
USING
TargetColumns
('price','lotsize','bedrooms','bathrms',
'stories')
MissValue ('keep')
ScaleMethod ('midrange')
GlobalScale ('f')
) AS dt;
```

Output

TD_STATTYPE_SCLFIT	price	lotsize	bedrooms	bathrms	stories
min	42000	3060	2	1	1
max	88500	6650	3	2	4
sum	586300	48420	25	12	18
count	9	10	9	10	10
null	1	0	1	0	0
avg	65144.4...	4842	2.77777...	1.2	1.8
multiplier	1	1	1	1	1
intercept	0	0	0	0	0
location	65250	4855	2.5	1.5	2.5
scale	23250	1795	0.5	0.5	1.5
globalscale_false					
ScaleMethodNumberMa...	5	5	5	5	5
missvalue_KEEP					

Lab 5b: TD_ScaleTransform Method('midrange')

teradata.

```
SELECT * FROM TD_scaleTransform (
ON TRNG_TDU_TD01.scale_housing AS InputTable
ON scaleFitOut AS FitTable DIMENSION
USING
Accumulate ('types','id')
) AS dt ORDER BY 2;
```

Midrange will scale from -1 to 1

Output

types	id	price	lotsize	bedrooms	bathrms	stories
classic	1	-1	0.55431...	1	-1	-0.33333...
classic	2		-0.47632...	-1	-1	-1
classic	3	-0.67741...	-1	1	-1	-1
classic	4	-0.20430...	1	1	-1	-0.33333...
classic	5	-0.18279...	0.83844...	-1	-1	-1
bungalow	6	0.03225...	-0.38718...	1	-1	-1
bungalow	7	0.03225...	-0.54317...		1	-0.33333...
bungalow	8	0.16129...	-0.38718...	1	-1	0.33333...
bungalow	9	0.79784...	-0.03064...	1	-1	-1
bungalow	10	1	0.35933...	1	1	1

Lab 5c: VAL Variable Transformation

teradata.


```
Scale housing data with Vantage Analytics Library function
variable transformation, 'vartran'

call
TRNG_XSP.td_analyze('vartran','database=TRNG_TDU_TD01;tablename=scale_
housing;keycolumns=id;
retain=columns(types,id);
rescale =
{rescalebounds (lowerbound/-1, upperbound/1), columns
(price,lotsize,bedrooms,bathrms,stories)}
');
```

Lab 5c: VAL Variable Transformation (cont.)

types	id	price	lotsize	bedrooms	bathrms	stories
classic	1	-1	0.55431…	1	-1	-0.33333…
classic	2		-0.4763…	-1	-1	-1
classic	3	-0.67741…	-1	1	-1	-1
classic	4	-0.20430…	1	1	-1	-0.33333…
classic	5	-0.18279…	0.83844…	-1	-1	-1
bungalow	6	0.032258…	-0.3871…	1	-1	-1
bungalow	7	0.032258…	-0.5431…		1	-0.33333…
bungalow	8	0.161290…	-0.3871…	1	-1	0.33333…
bungalow	9	0.797849…	-0.0306…	1	-1	-1
bungalow	10	1	0.35933…	1	1	1

Lab 6a: TD_ScaleFit (Multiple Method)


```
SELECT * FROM TD_ScaleFit (
ON TRNG_TDU_TD01.scale_housing AS InputTable
OUT PERMANENT TABLE OutputTable (scaleFitOut)
USING
TargetColumns
('price','lotsize','bedrooms','bathrms','stories')
MissValue ('keep')
ScaleMethod ('midrange','mean','maxabs','range','std')
GlobalScale ('f')
) AS dt;
```

Output

TD_STATTYPE_SCLFIT	price	lotsize	bedrooms	bathrms	stories
min	42000	3060	2	1	1
max	88500	6650	3	2	4
sum	586300	48420	25	12	18
count	9	10	9	10	10
null	1	0	1	0	0
avg	65144.444...	4842	2.777777...	1.2	1.8
variance	2.1612527...	1.418239999...	0.194444...	0.177777...	1.0666...
std	13860.424...	1129.785820...	0.415739...	0.4	0.9797...
ustd	14701.199...	1190.898820...	0.440958...	0.421637...	1.0327...
multiplier	1	1	1	1	1
intercept	0	0	0	0	0
location	65250	4842	0	1	1.8
scale	23250	1	3	1	0.9797

Lab 6b: TD_ScaleTransform (Multiple Method)

```
SELECT * FROM TD_scaleTransform (
ON TRNG_TDU_TD01.scale_housing AS InputTable
ON scaleFitOut AS FitTable DIMENSION
USING
Accumulate ('id')
) AS dt ORDER BY 1;
    price     lotsize   bedrooms   bathrms  stories
    midrange   mean      maxabs     range     std
```

Each feature is scaled by the method in order.

Price scales with midrange since they are both first in order

Output

id	price	lotsize	bedrooms	bathrms	stories
1	-1	1008	1	0	0.20412414...
2		-842	0.66666666...	0	-0.81649658...
3	-0.67741935...	-1782	1	0	-0.81649658...
4	-0.20430107...	1808	1	0	0.20412414...
5	-0.18279569...	1518	0.66666666...	0	-0.81649658...
6	0.032258064...	-682	1	0	-0.81649658...
7	0.032258064...	-962		1	0.20412414...
8	0.161290322...	-682	1	0	1.22474487...

Lab 7a: TD_ScaleFit: Method('maxabs')


```
SELECT * FROM TD_ScaleFit (
ON TRNG_TDU_TD01.computers_train1 AS InputTable
OUT PERMANENT TABLE OutputTable (scaleFitOut)
USING
TargetColumns
('price','speed','hd','ram','screen')
MissValue ('zero')
ScaleMethod ('maxabs')
GlobalScale ('f')
) AS dt;
```

Input

id	price	speed	hd	ram	screen
3683	2490	75	426	8	15
753	1890	25	214	4	14
3700	2970	66	420	16	15
549	1825	50	170	4	14
4832	2145	66	420	8	15
3258	2319	33	340	4	14
6014	1354	50	528	4	14
1788	1499	33	212	4	14
3970	2929	66	527	16	15
967	1399	25	170	4	14

Lab 7b: TD_ScaleTransform: Method('maxabs')


```
CREATE TABLE scaleTransformOut AS (
SELECT * FROM TD_scaleTransform (
ON TRNG_TDU_TD01.computers_train1 AS
InputTable
ON scaleFitOut AS FitTable DIMENSION
USING
Accumulate ('id')
) AS dt
) WITH DATA;
```

Output

id	price	speed	hd	ram	screen
3018	0.45730690...	0.33	0.1619047...	0.125	0.88235294...
4425	0.46175217...	1	0.2514285...	0.25	0.82352941...
6036	0.26764215...	1	0.2514285...	0.125	0.82352941...
1876	0.43785886...	0.33	0.1009523...	0.125	0.82352941...
469	0.48138544...	0.5	0.1928571...	0.25	0.82352941...
4894	0.32209668...	0.33	0.1009523...	0.125	1
3487	0.37025375...	0.33	0.2	0.25	0.88235294...
2345	0.33246897...	0.33	0.2	0.25	0.88235294...
938	0.51842933...	0.5	0.1523809...	0.25	0.88235294...
5363	0.32487497...	0.33	0.2514285...	0.25	0.82352941...
3956	0.38803482...	1	0.1019047...	0.125	0.82352941...
2814	0.49435080...	0.5	0.1619047...	0.25	0.82352941...
1407	0.48972031...	0.66	0.2028571...	0.25	0.82352941...
265	0.35173180...	0.5	0.0571428...	0.125	0.82352941...
5832	0.27597703...	0.66	0.1619047...	0.125	0.88235294...
3283	0.27690313...	0.33	0.1619047...	0.125	0.82352941...
1203	0.45749212...	0.5	0.1619047...	0.25	0.88235294...
5159	0.35173180...	0.66	0.3476190...	0.25	0.82352941...
4221	0.31394702...	0.66	0.2023809...	0.25	0.88235294...
3079	0.31950361...	0.33	0.1009523...	0.125	0.88235294...

All Output values between 0 and 1 if all input numbers are positive

Lab 7c: VAL Variable Transformation

```
Scale computer data with Vantage Analytics Library function
variable transformation, 'vartran'

call
TRNG_XSP.td_analyze('vartran','database=TRNG_TDU_TD01;tablename=comput
ers_train1;keycolumns=id;
retain=columns(id);
rescale =
{rescalebounds (lowerbound/0, upperbound/1), nullstyle (literal, 0)
columns (price,speed,hd,ram,screen)}
');
```

Lab 7c: VAL Variable Transformation (cont.)

id	price	speed	hd	ram	screen
3018	0.34157303...	0.106666...	0.128712...	0.06666...	0.33333...
4425	0.34696629...	1	0.221782...	0.2	0
6036	0.11146067...	1	0.221782...	0.06666...	0
1876	0.31797752...	0.106666...	0.065346...	0.06666...	0
469	0.37078651...	0.333333...	0.160891...	0.2	0
4894	0.17752808...	0.106666...	0.065346...	0.06666...	1
3487	0.23595505...	0.106666...	0.168316...	0.2	0.33333...
2345	0.19011235...	0.106666...	0.168316...	0.2	0.33333...
938	0.41573033...	0.333333...	0.118811...	0.2	0.33333...
5363	0.18089887...	0.106666...	0.221782...	0.2	0
3956	0.25752808...	1	0.066336...	0.06666...	0
2814	0.38651685...	0.333333...	0.128712...	0.2	0
1407	0.38089887...	0.546666...	0.171287...	0.2	0
265	0.21348314...	0.333333...	0.019801...	0.06666...	0
5832	0.12157303...	0.546666...	0.128712...	0.06666...	0.33333...
3283	0.12269662...	0.106666...	0.128712...	0.06666...	0
1203	0.34179775...	0.333333...	0.128712...	0.2	0.33333...
5159	0.21348314...	0.546666...	0.321782...	0.2	0
4221	0.16764044...	0.546666...	0.170792...	0.2	0.33333...
3079	0.17438202...	0.106666...	0.065346...	0.06666...	0.33333...

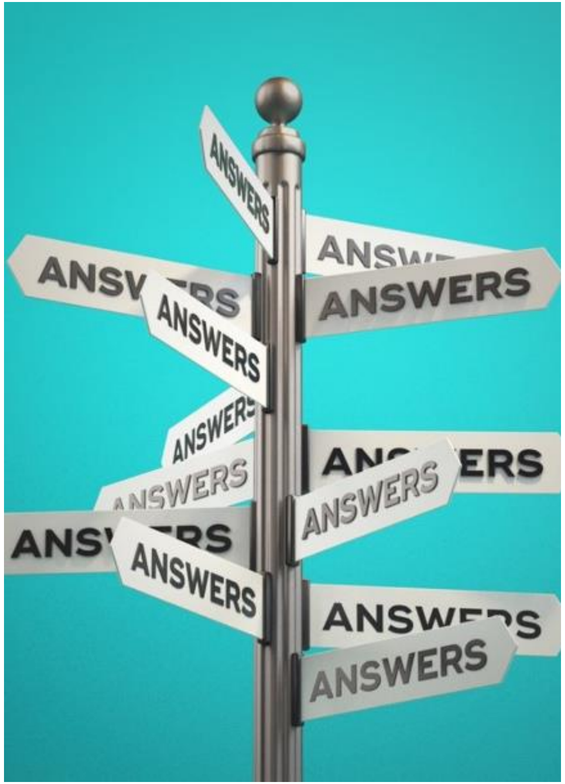
Vantagecour Interactive

Top 10 most populous countries

Continent	Country
Asia	
Asia	
North America	
Asia	
South America	
Asia	
Africa	
Asia	
Asia	
North America	

Current Topic – Histogram

- Data Science Pipelines
- Outlier Filtering
- Normalization (Scale functions)
- **Histogram**
- Review & Summary



Description – Histogram

Histograms are useful for assessing the shape of a data distribution. The Histogram function calculates the frequency distribution of a data set using either the Sturges or Scott algorithm to compute binning (bin width and number of bins)

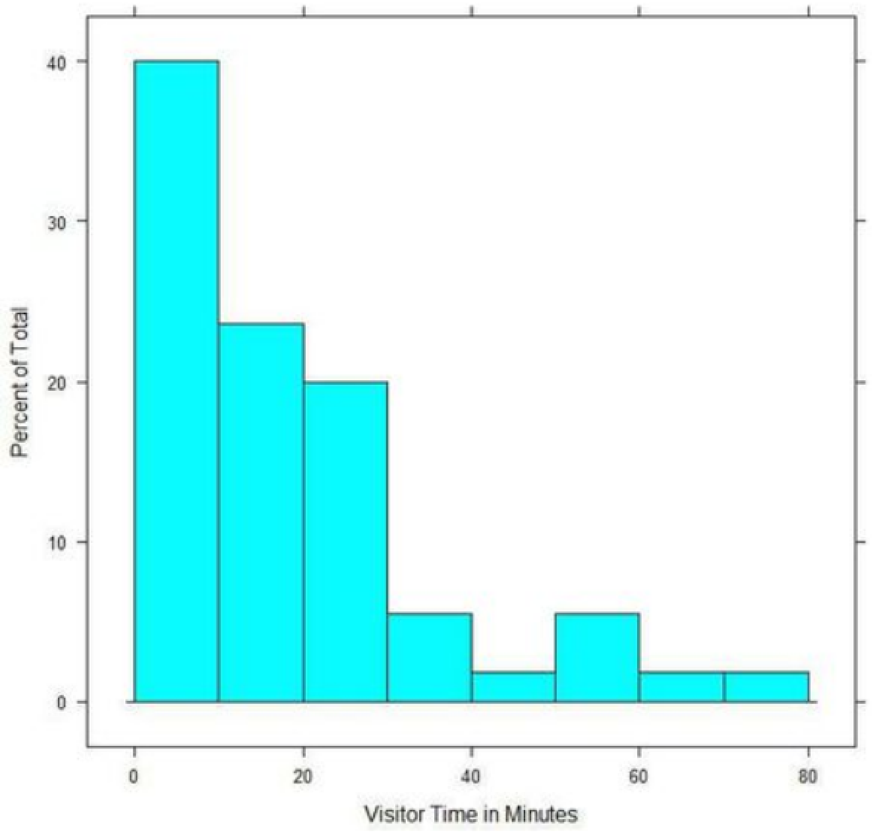
- User-selected or automatic bin determination
- User-selected left-inclusive or right-inclusive binning
- Multiple histograms for distinct groups

Histograms vs. Bar Charts

- Histograms show continuous data distributions while Bar charts compare variables
- Histograms plot quantitative data with ranges of the data grouped into bins or intervals while Bar charts plot categorical data
- Bars can be reordered in Bar charts but not in Histogram
- The bars of Bar charts typically have the same width. The widths of the bars in a Histogram need not be the same as long as the total area is one hundred percent (if percents are used) or the total count if counts are used. Therefore, values in Bar charts are given by the length of the bar while values in Histograms are given by areas (length and width).

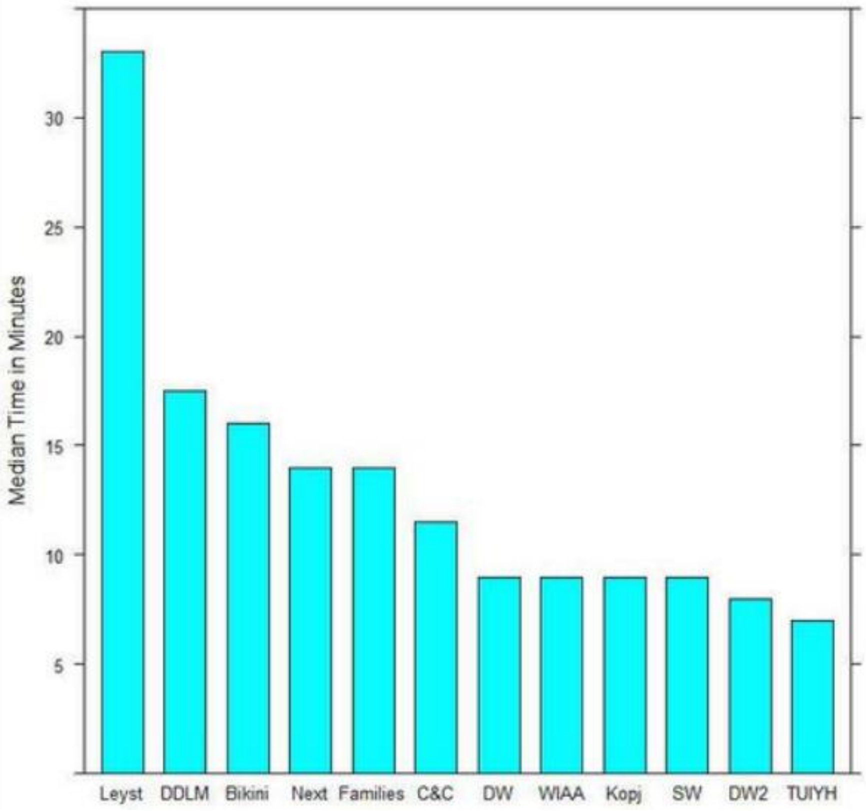
Histogram Chart

- The horizontal axis of consists of binned times: the first bin includes visits from 0 up to and including ten minutes, the second bin from 10 up to and including 20 minutes, and so on
- The vertical axis shows percentages. The area of each bar gives the percentage of all visitors who spent the amount of time shown in the corresponding bin. The sum of all areas equals 100%
- Note it does not make sense to rearrange the bars of a Histogram



Bar Chart

- The Bar chart compares median times visitors stayed at each of 12 exhibitions
- The variables on the horizontal axis are categorical; the names of the exhibitions
- The vertical axis indicates time in minutes. The height of each bar represents the median time for that exhibition
- Bars of a Bar chart can be rearranged at will. Many graph designers order the variables alphabetically while ordering by size is usually more informative



Why Use Histograms?

Suppose you have a set of numbers: 1, 23, 24, 25, 25, 25, 26, 27, 30, 32, 999

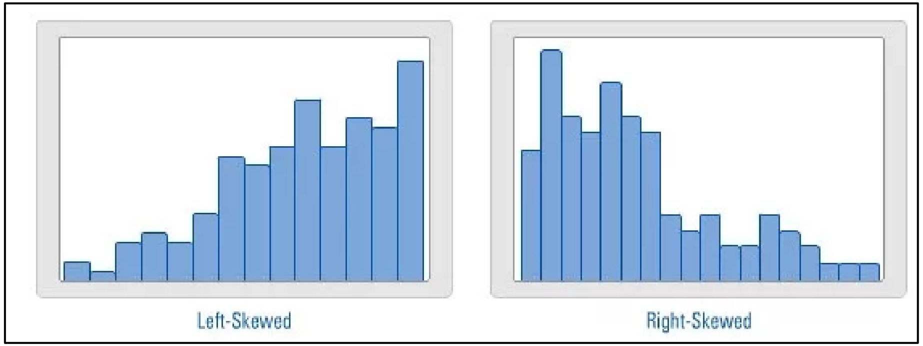
- The Mean value (112.45) is very sensitive to outliers. Almost all real-world data has outliers, so the mean value can be very misleading
- The Median value (25) does not tell you anything about the distribution
- The full Range (1 - 999) just shows the Outliers
- The Standard Deviation (294.1436) can be hard to be interpreted without a statistical background
- The Variance (86520.47) can be also hard to be interpreted without a statistical background
- Interquartile range (IQR) (24.5 – 28.5) is the central 50% of your values and does not tell you anything about the other 50%

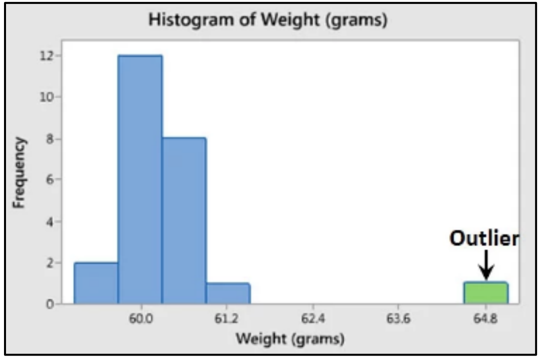
Which best describes the numbers? Only Histograms can display spikes and Distribution of data

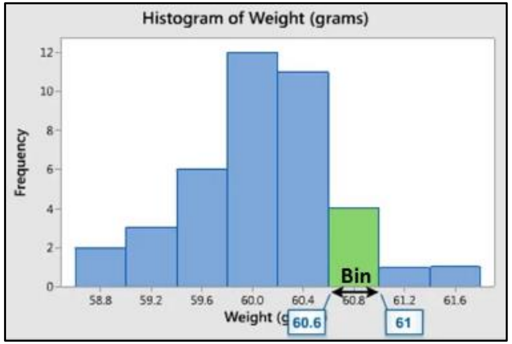
Why Use Histograms? (cont.)

Histograms can tell you:

1. Where the median and mean are
   - If Left-skewed, mean < median
   - If Right-skewed, mean > media
2. Bins allow you to see ranges and how spread out the data is
3. Outliers can be easily found using Histograms







Workflow – Histogram

- **Input Table:** Data is read from a specified input table, views, or query
- **Output table:** Data is written to an output table (or to Console)

Syntax – TD_Histogram

```
SELECT * TD_Histogram (
ON { table | view | (query) } AS InputTable
[ ON { table | view | (query) } AS MinMax DIMENSION ]
USING
MethodType ({ 'Sturges' | 'Scott' | 'Variable-Width' | 'Equal-Width' })
TargetColumn ('target_column')
[ NBins ('number_of_bins') ]
[ Inclusion ({ 'left' | 'right' }) ]
) AS alias;
```

Arguments – Histogram

- **MethodType:** Specify the method for calculating the frequency distribution of the data set
- **TargetColumn:** Specify the name of the InputTable column that contains the data set
- **Nbins:** [Required with methods Variable-Width and Equal-Width, otherwise ignored.] Specify the number of bins (number of data value ranges)
- **Inclusion:** [Optional] Specify where to put data points that are on bin boundaries—in the bin to the left of the boundary or the bin to the right of boundary.

  Default: left

MethodType

Methods	Description
Sturges	Algorithm for calculating bin width, *w*: $w = r/(1 + \log 2n)$ where: *w* = bin width *r* = data value range *n* = number of elements in data set Sturges algorithm performs best if data is normally distributed, and *n* is at least 30.
Scott	Algorithm for calculating bin width, *w*: $w = 3.49s/(n1/3)$ where: *w* = bin width *s* = standard deviation of data values *n* = number of elements in data set *r* = data value range Number of bins: *r*/*w* Scott algorithm performs best on normally distributed data.

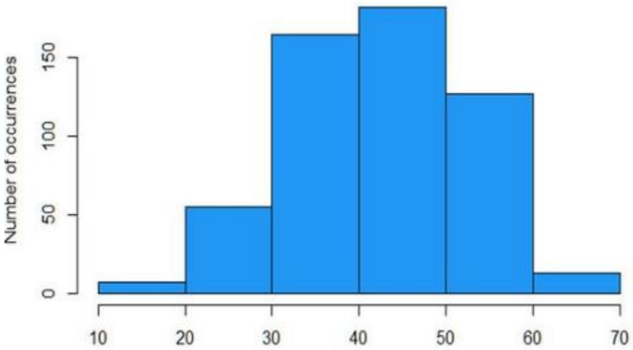
MethodType (cont.)

Methods	Description
Variable Width	Requires MinMax table, which specifies the minimum value and the maximum value of the bin in column1 and column2 respectively, and the label of the bin in column3. Maximum number of bins cannot exceed 3500
Equal Width	Requires MinMax table, which specifies the minimum value of the bins in column1 and the maximum value of the bins in column2. Algorithm for calculating bin width, *w*: $w = (max - min)/k$ where: *min* = minimum value of the bins *max* = maximum value of the bins *k* = number of intervals into which algorithm divides data set Interval boundaries: $min+w$, $min+2w$, …, $min+(k-1)w$

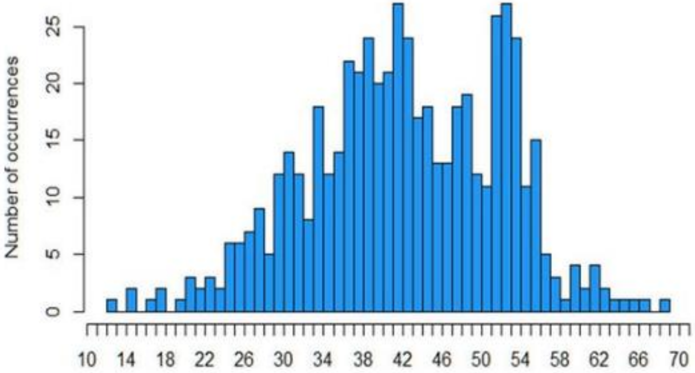
AutoBin (Sturges and Scott)

AutoBin: Specify either the algorithm for selecting bin boundaries (Sturges or Scott). These two techniques determines the number of bins and bin width.

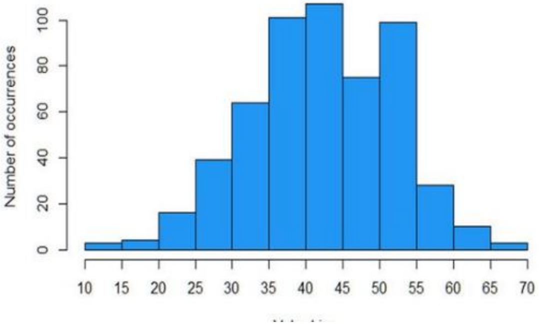
Too wide bins



Too narrow bins



Ideal bins



If you have a small amount of data, use wider bins to eliminate noise. If you have a lot of data, use narrower bins because the histogram will not be that noisy
https://www.answerminer.com/blog/binning-guide-ideal-histogram

What the Data Looks Like

```
SELECT * FROM cars_hist;
```

```
SELECT *
FROM bin_breaks;
```

break
50
100
150
200
250
300
350

id	name	cyl	hp
1	mazda rx4	6	110
2	mazda rx4 wag	6	110
3	datsun 710	4	93
4	hornet 4 drive	6	110
5	hornet sportabout	8	175
6	valiant	6	105
7	duster 360	8	245
8	merc 240d	4	62
9	merc 230	4	95
10	merc 280	6	123
11	merc 280c	6	123
12	merc 450se	8	180
13	merc 450sl	8	180
14	merc 450slc	8	180
15	cadillac fleetwood	8	205
16	lincoln continental	8	215
17	chrysler imperial	8	230
18	fiat 128	4	66
19	honda civic	4	52
20	toyota corolla	4	65
21	toyota corona	4	97
22	dodge challenger	8	150
23	amc javelin	8	150
24	camaro z28	8	245

Lab 8a: AutoBin (Sturges)

```
SELECT * FROM TD_Histogram (
ON TRNG_TDU_TD01.cars_hist AS InputTable
USING
TargetColumn ('hp')
MethodType ('Sturges')
) AS dt ORDER BY 1;
```

MinValue: >=
MaxValue: <

Label	MinValue	MaxValue	CountOfValues	bin_Percent
0	50	100	9	28.125
1	100	150	8	25
2	150	200	8	25
3	200	250	5	15.625
4	250	300	1	3.125
5	300	350	1	3.125

Lab 8b: AutoBin (Scott)


```
SELECT * FROM TD_Histogram (
ON TRNG_TDU_TD01.cars_hist AS
InputTable
USING
TargetColumn ('hp')
MethodType ('Scott')
) AS dt ORDER BY 1;
```

Label	MinValue	MaxValue	CountOfValues	bin_Percent
0	0	100	9	28.125
1	100	200	16	50
2	200	300	6	18.75
3	300	400	1	3.125

Lab 8c: Variable-Width Left-inclusion (default)


```
SELECT * FROM TD_Histogram (
ON TRNG_TDU_TD01.cars_hist AS
InputTable
ON cars_hist_minmax AS MinMax
DIMENSION
USING
TargetColumn ('hp')
MethodType ('Variable-Width')
NBins ('3')
Inclusion ('left')
) AS dt ORDER BY 2;

SELECT * FROM cars_hist_minmax;
```

Label	MinValue	MaxValue	CountOfValues	bin_Percent
Slow	0	105	9	28.125
Medium	105	200	16	50
Fast	200	999	7	21.875

minvalue	maxvalue	label
0	105	Slow
105	200	Medium
200	999	Fast

Lab 8c: Variable-Width Right-inclusion


```sql
SELECT * FROM TD_Histogram (
ON TRNG_TDU_TD01.cars_hist AS
InputTable
ON cars_hist_minmax AS MinMax
DIMENSION
USING
TargetColumn ('hp')
MethodType ('Variable-Width')
NBins ('3')
Inclusion ('right')
) AS dt ORDER BY 2;

SELECT * FROM cars_hist_minmax;
```

Label	MinValue	MaxValue	CountOfValues	bin_Percent
Slow	0	105	10	31.25
Medium	105	200	15	46.875
Fast	200	999	7	21.875

minvalue	maxvalue	label
0	105	Slow
105	200	Medium
200	999	Fast

Lab 8c: Equal-Width


```
SELECT * FROM TD_Histogram (
ON TRNG_TDU_TD01.cars_hist AS
InputTable
ON cars_hist_minmax AS MinMax
DIMENSION
USING
TargetColumn ('hp')
MethodType ('Equal-Width')
NBins ('8')
) AS dt ORDER BY 2;


SELECT * FROM cars_hist_minmax;
```

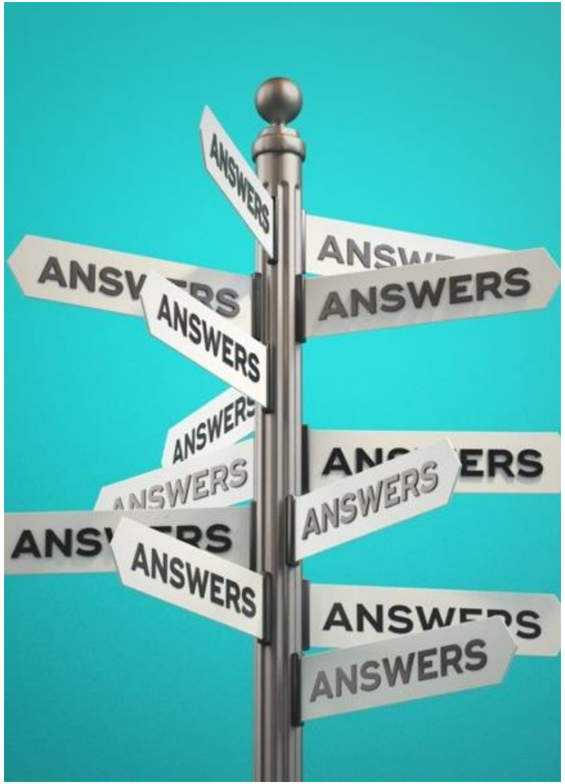
Label	MinValue	MaxValue	CountOfValues	bin_Percent
0	0	50	0	0
1	50	100	9	28.125
2	100	150	8	25
3	150	200	8	25
4	200	250	5	15.625
5	250	300	1	3.125
6	300	350	1	3.125
7	350	400	0	0

minvalue	maxvalue
0	400

Minmax table sets the range, Nbins creates bins

Current Topic – Review & Summary

- Data Science Pipelines
- Outlier Filtering
- Normalization (Scale functions)
- Histogram
- **Review & Summary**



Summary

In this module, you learned how to:

- Describe Data Science Pipelines
- Write queries using these Teradata Vantage Transformation functions:
  - **Outlier Filtering**
  - **Normalization (Scale Functions)**
  - **Histogram**

Thank you.

teradata.