Data Modeling with Power BI Desktop

Agenda

- Data Modeling with Power BI Desktop

- Understanding the DAX Evaluation Context

- Creating a Dynamic Calendar Table

- Designing Interactive Reports

- Understanding Row-level Security (RLS)

- Publishing PBIX Projects to PowerBI.com

Data Modeling with Power BI Desktop

- Steps to create a data model with Power Pivot
  - Create relationships between tables
  - Modify columns (rename, set formatting, convert type)
  - Create calculated columns
  - Create measures
  - Add column metadata
  - Create dimensional hierarchies
  - Add Calendar table(s)

Table Relationships

- Tables in data model associated with relationships
  - Relationships based on single columns
  - Tabular model supports [1-to-1] and [1-to-many] relationships
  - Relationships based on single column in each table
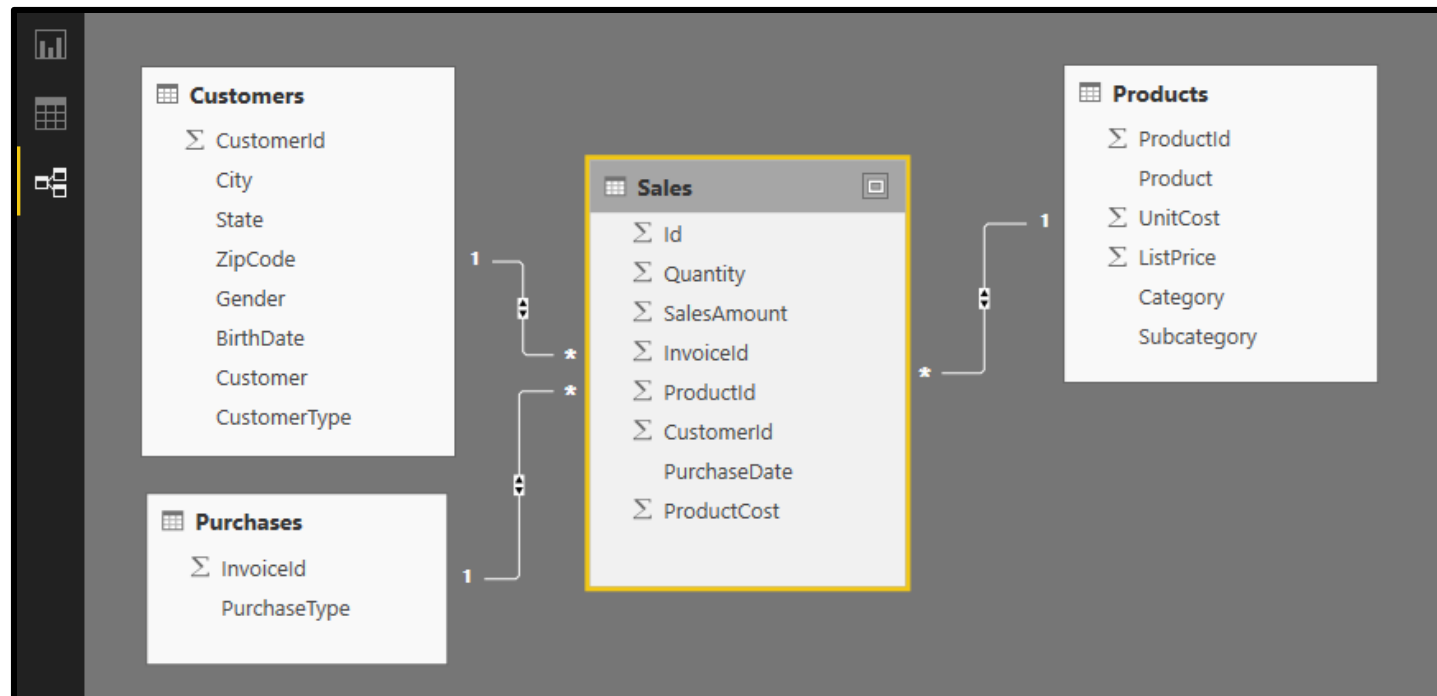
Calculated Columns vs Measures

- ## Calculated Columns (aka Columns)
  - Evaluated based on context of a single row
  - Evaluated when data is loaded into memory

  ```
  Column1 = <DAX expression>
  ```

- ## Measures
  - Evaluated at query time based on current filter context
  - Commonly used for aggregations (e.g. SUM, AVG, etc.)
  - Used more frequently than calculated columns
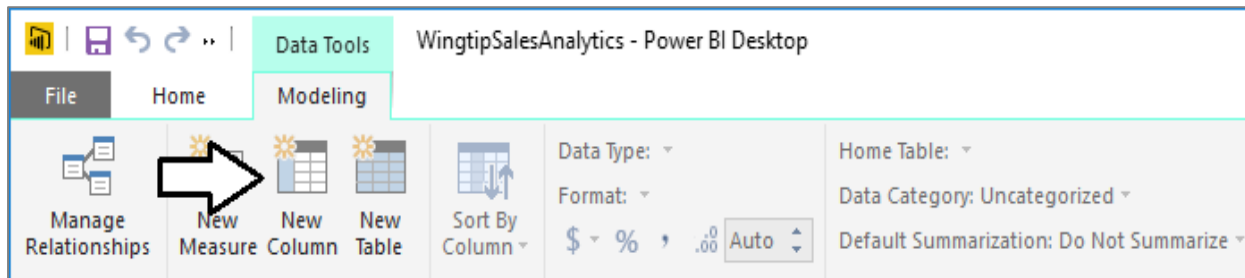
  ```
  Measure1 = <DAX expression>
  ```

When to Create Calculated Columns

- Measures often better choice than calculate columns
  - Don't create calculated column when you need a measure
  - Prefer to create calculated columns only in specific scenarios

- When should you create calculated columns?
  - To create headers for row labels or column labels
  - To place calculated results in a slicer for filtering
  - Define an expression strictly bound to current row
  - Categories text or numbers (e.g. customer age groups)

Creating Calculated Columns

- Edited in formula bar of Power Pivot data view
  - Start with name and then equals (=) sign
  - Enter a valid DAX expression
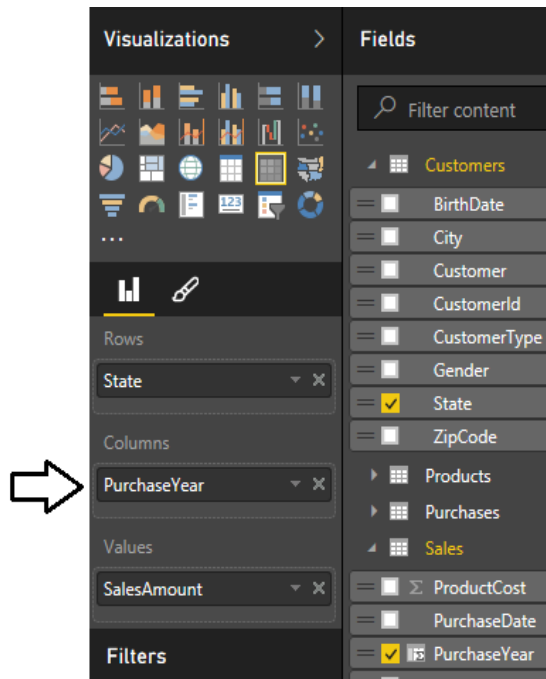 - Clicking on column adds it into expression

Calculated Column as a Column Label

- Calculate column can serve as...
  - Row labels
  - Column labels

State	2012	2013	2014	2015	Total
CA	$270,926.32	$550,160.02	$737,878.53	$770,402.11	$2,329,366.98
TX	$212,085.08	$490,643.98	$683,079.11	$919,030.36	$2,304,838.53
FL	$51,730.85	$300,866.87	$535,693.94	$891,344.92	$1,779,636.58
NC	$11,018.02	$164,804.24	$315,139.92	$448,638.72	$939,600.90
NY	$24,207.43	$165,046.23	$256,294.27	$430,971.24	$876,519.17
GA	$40,305.80	$152,807.51	$239,451.05	$417,037.28	$849,601.64

Calculated Column for Customer Age Group

1. Calculate customer age from birthdate

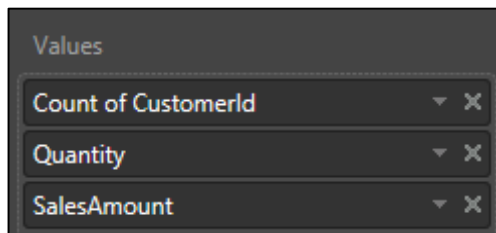2. Calculate age groups using calculated column

Calculated Column used in a Slicer

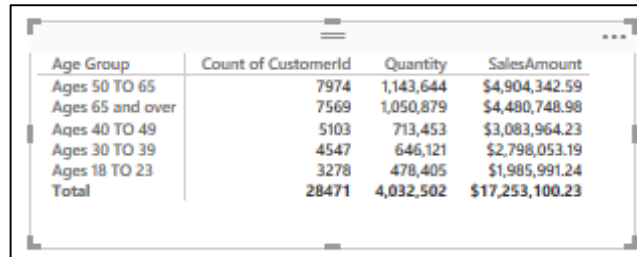- Calculated column can populate slicer values
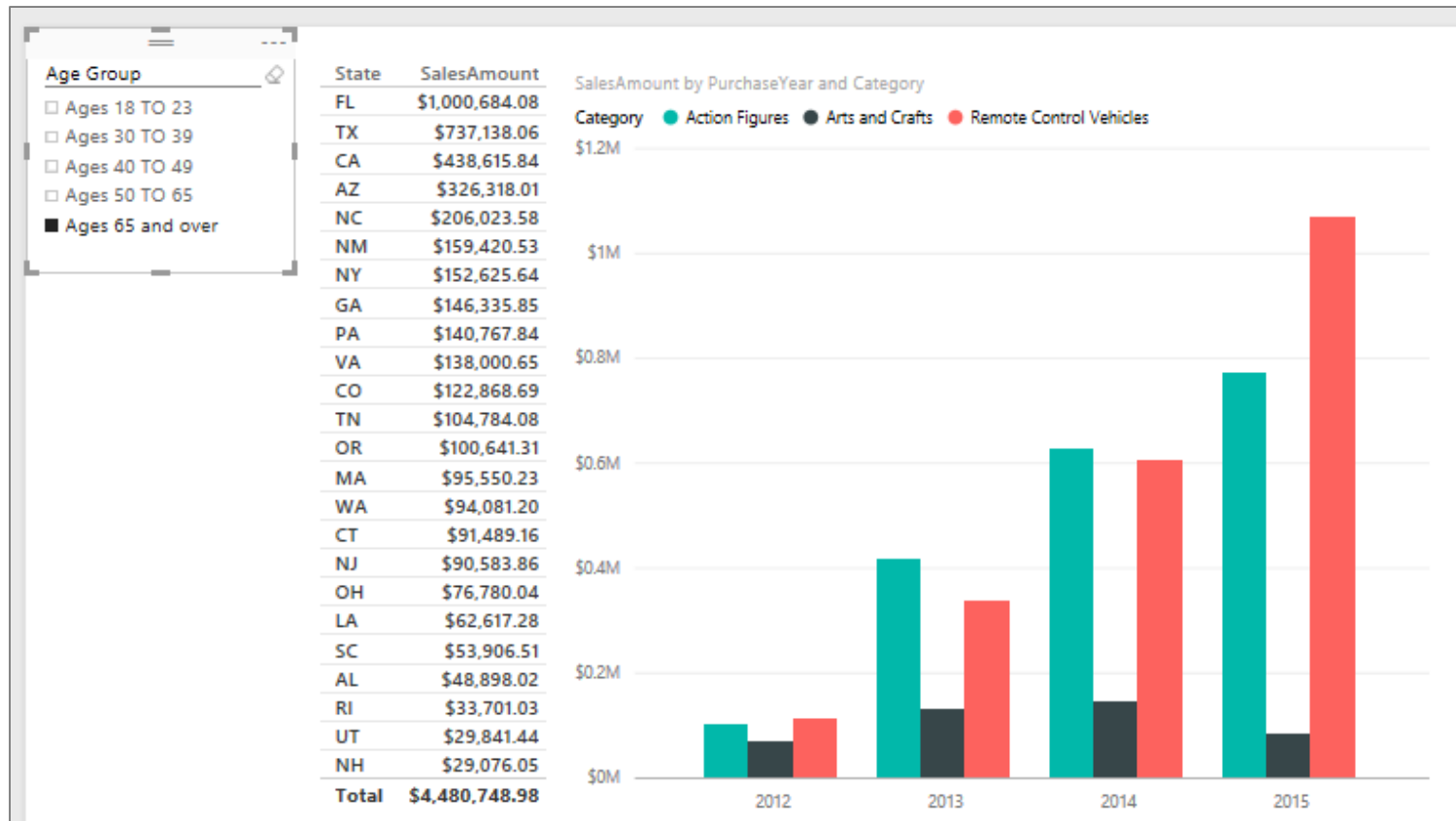
Benefits of Measures over Calculated Columns

- Calculated columns can be aggregated in visual
  - However, aggregation details are stored in visual
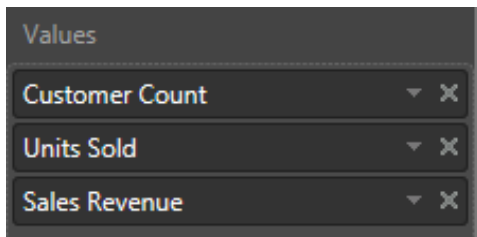 - Visual doesn't offer control over name and formatting
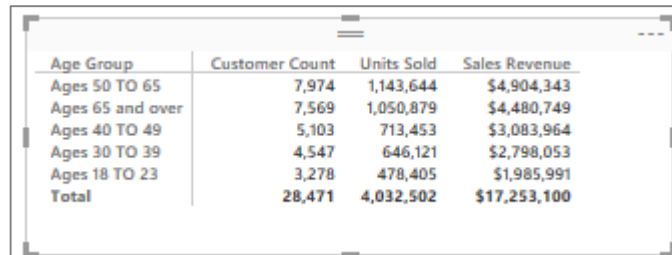

Age Group	Count of CustomerId	Quantity	SalesAmount
Ages 50 TO 65	7974	1,143,644	$4,904,342.59
Ages 65 and over	7569	1,050,879	$4,480,748.98
Ages 40 TO 49	5103	713,453	$3,083,964.23
Ages 30 TO 39	4547	646,121	$2,798,053.19
Ages 18 TO 23	3278	478,405	$1,985,991.24
Total	28471	4,032,502	$17,253,100.23

- Measure defines name, aggregation and formatting
  - Work is done once and reused across many visuals
  - Makes data model more fool-proof for report designers

Age Group	Customer Count	Units Sold	Sales Revenue
Ages 50 TO 65	7,974	1,143,644	$4,904,343
Ages 65 and over	7,569	1,050,879	$4,480,749
Ages 40 TO 49	5,103	713,453	$3,083,964
Ages 30 TO 39	4,547	646,121	$2,798,053
Ages 18 TO 23	3,278	478,405	$1,985,991
Total	28,471	4,032,502	$17,253,100

Creating Measures

- Measures have advantage over calculated columns
  - They are evaluated based on the current evaluation context

- Creating a measure with Power BI Desktop
  1. Click New Measure button
  2. Give measure a name and write DAX expressions
  3. Configure formatting

Geographic Field Metadata

- Fields in data model have metadata properties
  - Metadata used by visuals and reporting tools
  - Used as hints to Bing Mapping service
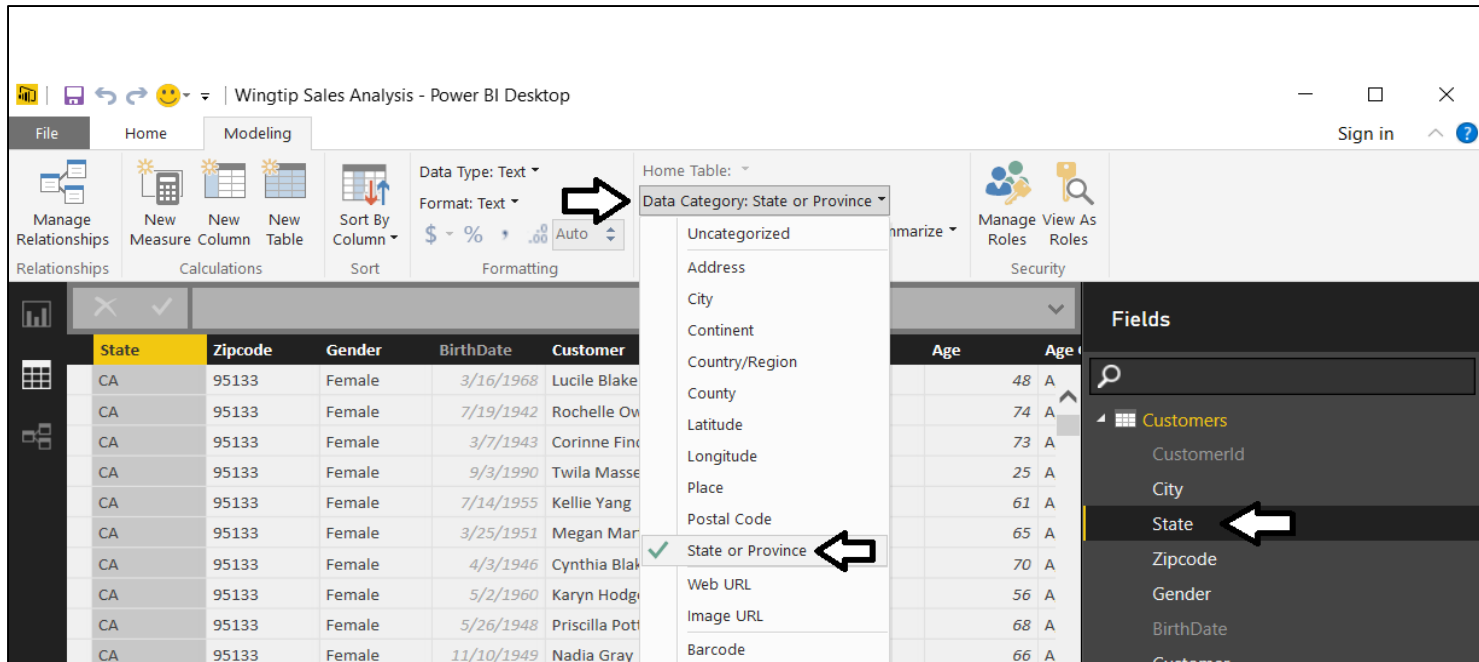
Eliminate Geographic Ambiguity

- ## City name alone is ambiguous
  - "Athens" defaults to Greece not Georgia
  - Concatenate city name with state to disambiguate

	Age Group	Sales Region	State Name	SalesRegionSort	City
48	Ages 40 TO 49	Western Region	California	1	San Jose, CA
74	Ages 65 and over	Western Region	California	1	San Jose, CA
73	Ages 65 and over	Western Region	California	1	San Jose, CA
25	Ages 18 TO 23	Western Region	California	1	San Jose, CA
61	Ages 50 TO 65	Western Region	California	1	San Jose, CA
65	Ages 65 and over	Western Region	California	1	San Jose, CA

City = [City Name] & ", " & [State]

Using Map Visual with a Geographic Field

- Map Visual shows distribution over geographic area
  - Visual automatically updates when filtered
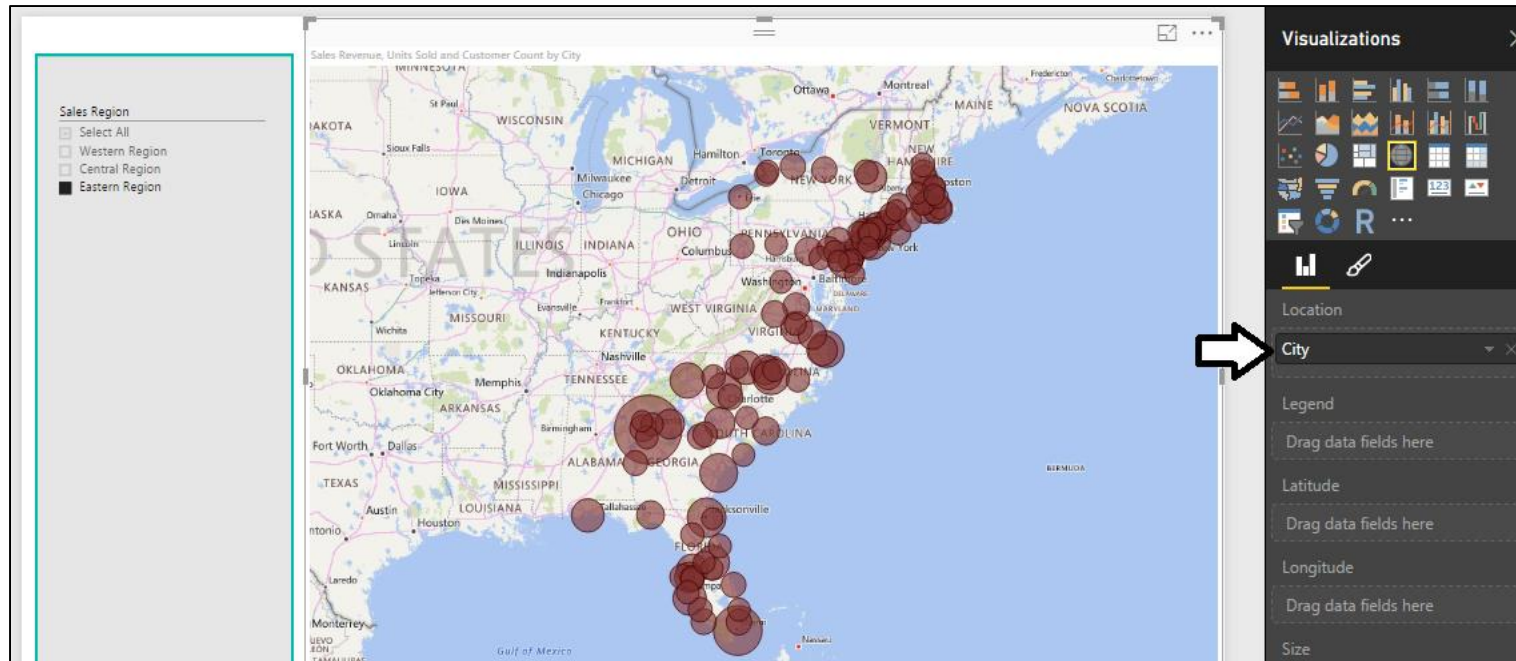
Dimensional Hierarchies

- Hierarchy created from two or more columns
  - All columns in hierarchy must be from the same table
  - Defines parent-child relationship between columns
  - Provides path to navigate through data
  - Provides path to drill down into greater level of detail
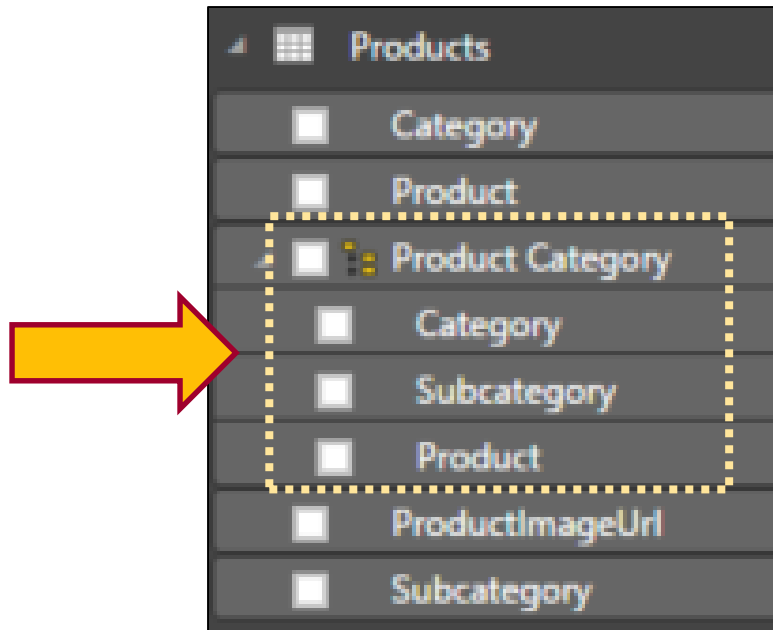
Pulling Columns for Hierarchy into Single Table

- Sometimes hierarchy columns are spread across tables
  - Use RELATED function from DAX to pull columns into single table

  - Then create hierarchy in the table with all the columns

Agenda

- ✓ Data Modeling with Power BI Desktop
- ➢ Understanding the DAX Evaluation Context
- Creating a Dynamic Calendar Table
- Designing Interactive Reports
- Understanding Row-level Security (RLS)
- Publishing PBIX Projects to PowerBI.com

A Tale of Two Evaluation Contexts

- ## Row Context
  - Context includes all columns in iteration of current row
  - Used to evaluate DAX expression in calculated column
  - Only available in measures with iterator function (e.g. SUMX)

- ## Filter Context
  - Context includes filter(s) defining current set of rows
  - Used by default to evaluate DAX expressions in measures
  - Can be fully ignored or partially ignored using DAX code
  - Not used to evaluate DAX in calculated columns

Understanding Row Context

- Row context used to evaluate calculated columns

City = [City Name] & ", " & [State]

	Age Group	Sales Region	State Name	SalesRegionSort	City
48	Ages 40 TO 49	Western Region	California	1	San Jose, CA
74	Ages 65 and over	Western Region	California	1	San Jose, CA
73	Ages 65 and over	Western Region	California	1	San Jose, CA
25	Ages 18 TO 23	Western Region	California	1	San Jose, CA
61	Ages 50 TO 65	Western Region	California	1	San Jose, CA
65	Ages 65 and over	Western Region	California	1	San Jose, CA

Age = Floor((TODAY()-Customers[BirthDate])/365, 1)

Customer	Customer Type	Age	Age Group	Sales Region	State Name
Lucile Blake	One-time Customer	48	Ages 40 TO 49	Western Region	California
Rochelle Owen	One-time Customer	74	Ages 65 and over	Western Region	California
Corinne Finch	One-time Customer	73	Ages 65 and over	Western Region	California

Understanding Iterators Like SUMX

- Standard aggregation functions (e.g. SUM) have no row context
  - You can use SUM to sum values of a single column
  - You cannot use SUM to sum results of an expressions

```
Gross Margin = SUM( Sales[SalesAmount]-Sales[ProductCost] )
```
⚠ The SUM function only accepts a column reference as an argument.
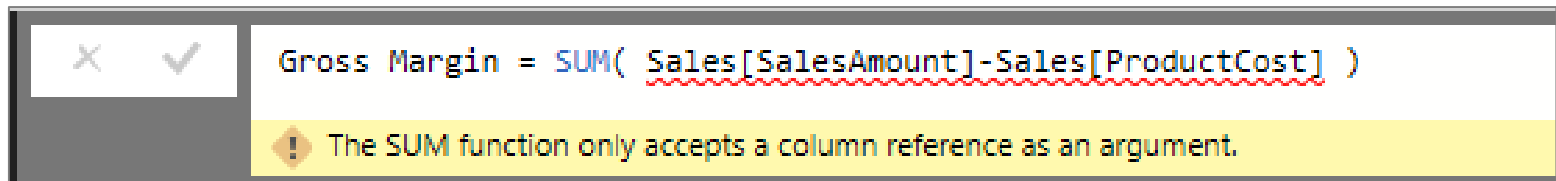
- Iterator functions (e.g. SUMX) iterate through rows in target table

```
Gross Margin = SUMX(Sales, Sales[SalesAmount]-Sales[ProductCost] )
```
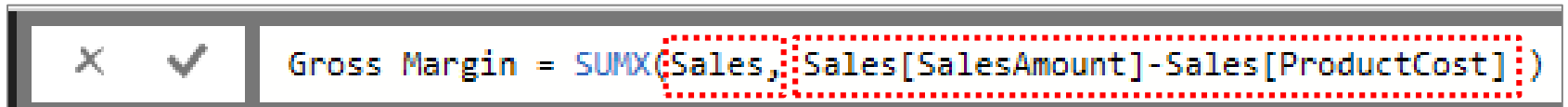
  - First argument accepts expressions that evaluates to table of rows
  - Second argument accepts expression that is evaluated for each row

DAX Table Iterator Functions

- The following DAX functions create row context
  - AVERAGEX
  - COUNTAX
  - COUNTX
  - MAXX
  - MINX
  - SUMX

Understanding Filter Context

- Visuals apply various filters in different evaluation contexts

Month in Year	2012	2013	2014	2015	Total
January	$6,306	$164,334	$385,275	$512,822	$1,068,737
February	$48,815	$126,501	$358,244	$597,684	$1,131,244
March	$53,958	$243,676	$381,309	$532,123	$1,211,067
April	$52,601	$300,872	$381,157	$602,751	$1,337,381
May	$61,756	$334,948	$438,261	$647,276	$1,482,241
June	$76,756	$321,715	$378,749	$608,448	$1,385,668
July	$104,408	$287,800	$359,744	$620,316	$1,372,268
August	$111,167	$298,483	$457,312	$678,499	$1,545,461
September	$110,716	$376,207	$505,332	$613,974	$1,606,229
October	$145,999	$362,943	$602,448	$620,735	$1,732,125
November	$156,751	$340,228	$545,572	$590,220	$1,632,770
December	$147,593	$331,526	$581,977	$686,814	$1,747,910
Total	$1,076,826	$3,489,234	$5,375,379	$7,311,660	$17,253,100

Filters on this evaluation

[Year] = 2015

[Month in Year] = "October"

- Filter context also affected by slicers and other filters

Sales Region

☐ Select All
☐ Central Region
☐ Eastern Region
■ Western Region

Customer Type

☐ One-time customer
■ Repeat Customer

Month in Year	2012	2013	2014	2015	Total
January	$425	$50,169	$61,295	$76,614	$188,503
February	$13,891	$40,133	$63,670	$101,542	$219,236
March	$19,121	$58,411	$73,839	$84,180	$235,551
April	$19,128	$53,711	$67,919	$91,762	$232,520
May	$22,939	$64,259	$78,668	$109,689	$275,555
June	$29,082	$50,564	$73,504	$88,047	$241,197
July	$34,809	$62,971	$69,053	$80,749	$247,582
August	$36,096	$61,217	$76,009	$94,719	$268,041
September	$39,415	$68,653	$82,697	$94,805	$285,570
October	$51,994	$69,122	$99,344	$84,177	$304,637
November	$47,020	$52,548	$85,924	$74,611	$260,102
December	$50,580	$66,260	$102,088	$94,877	$313,804
Total	$364,500	$698,018	$934,009	$1,075,771	$3,072,298

Filters on this evaluation

[Year] = 2015

[Month in Year] = "October"

[Sales Region] = "Western Region"

[Customer Type] = "Repeat Customer"

Using the CALCULATE Function

- CALCULATE function provides greatest amount of control
  - First argument defines expression to evaluate
  - Second argument defines table on which to evaluate expression
  - You can evaluate expressions with or without current filter context

```
Pct of All Products =
DIVIDE(
    SUM( Sales[SalesAmount] ),
    CALCULATE(
        Sum (Sales[SalesAmount] ),
        ALL(Products[Category], Products[Subcategory], Products[Product])
    )
)
```

```
Pct of Product Category =
DIVIDE(
    SUM( Sales[SalesAmount] ),
    CALCULATE(
        Sum (Sales[SalesAmount] ),
        ALL( Products[Subcategory], Products[Product] )
    )
)
```

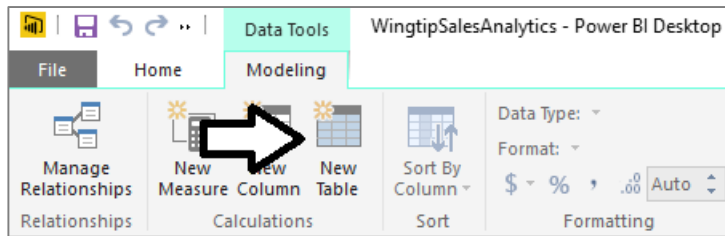# Agenda

- ✓ Data Modeling with Power BI Desktop
- ✓ Understanding the DAX Evaluation Context
- ➢ Creating a Dynamic Calendar Table
- • Designing Interactive Reports
- • Understanding Row-level Security (RLS)
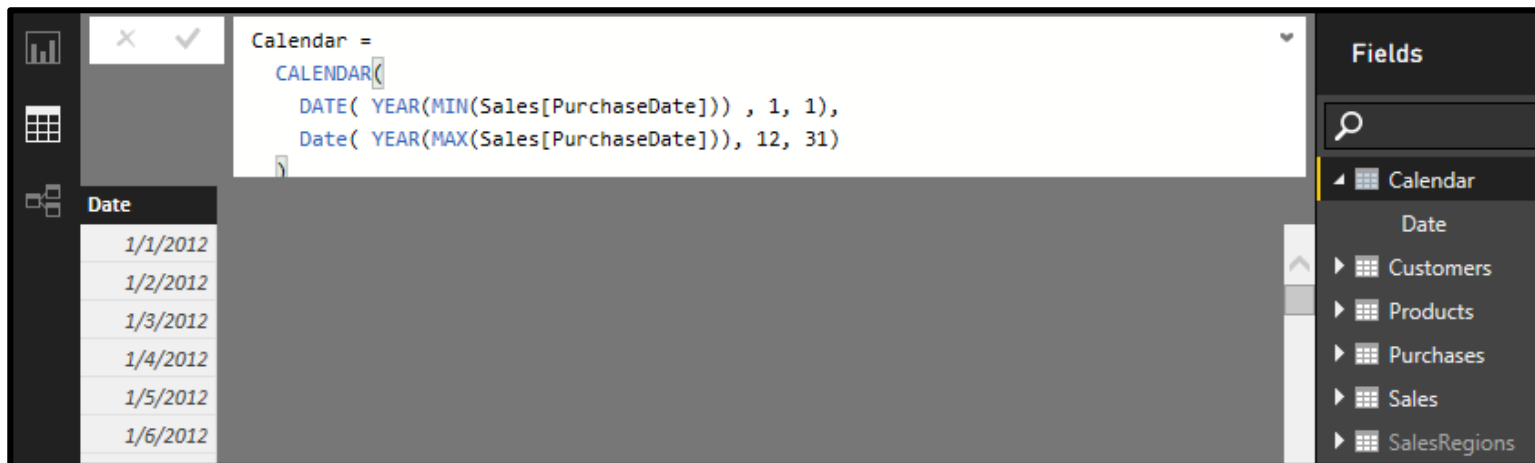- • Publishing PBIX Projects to PowerBI.com

Creating Calendar Table as Calculated Table

- Use **New Table** command in ribbon

- Create calendar table using DAX **CALENDAR** function

Adding Columns to Calendar Table

- Creating the **Year** column

✕ ✓	Year = YEAR('Calendar'[Date])

Date	Year
1/1/2012	2012
1/2/2012	2012
1/3/2012	2012

- Creating the **Quarter** column

✕ ✓	Quarter = YEAR('Calendar'[Date]) & "-Q" & FORMAT('Calendar'[Date], "q")

Date	Year	Quarter
01/01/2012	2012	2012-Q1
01/02/2012	2012	2012-Q1
01/03/2012	2012	2012-Q1
01/04/2012	2012	2012-Q1
01/05/2012	2012	2012-Q1

- Creating the **Month** column

✕ ✓	Month = FORMAT('Calendar'[Date], "MMM yyyy")

Date	Year	Quarter	Month
1/1/2012	2012	2012-Q1	Jan 2012
1/2/2012	2012	2012-Q1	Jan 2012
1/3/2012	2012	2012-Q1	Jan 2012

Configuring Sort Columns

- Month column will not sort in desired fashion by default
  - For example, April will sort before January, February and March

- Creating a sort column for the **Month** column
  - **MonthSort** sorts alphabetically & chronologically at same time

  - Configure **Month** column with **MonthSort** as sort column

Columns for Month in Year and Day in week

- Creating the **Month in Year** column

× ✓	Month in Year = FORMAT('Calendar'[Date], "MMMM")				
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January

- Creating the **MonthInYearSort** column

× ✓	MonthInYearSort = MONTH('Calendar'[Date])					
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**	**MonthInYearSort**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January	1
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January	1

- Creating the **Day of Week** column

× ✓	Day of Week = FORMAT('Calendar'[Date], "dddd")						
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**	**MonthInYearSort**	**Day of Week**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Sunday
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Monday

- Creating the **DayOfWeekSort** column

|								
× ✓	DayOfWeekSort = WEEKDAY('Calendar'[Date], 2)							
Date	**Year**	**Quarter**	**Month**	**MonthSort**	**Month in Year**	**MonthInYearSort**	**Day of Week**	**DayOfWeekSort**
1/1/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Sunday	7
1/2/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Monday	1
1/3/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Tuesday	2
1/4/2012	2012	2012-Q1	Jan 2012	2012-01	January	1	Wednesday	3

Integrating Calendar Table into Data Model

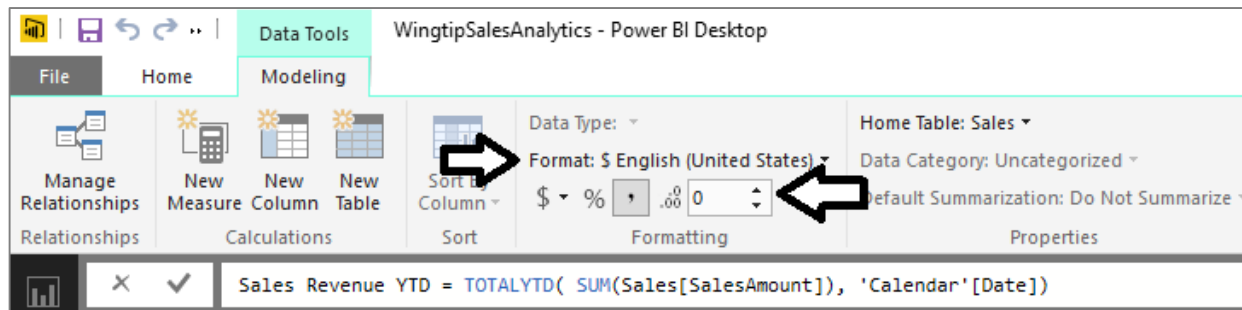- Calendar table needs relationship to one or more tables

Calculated Fields for QTD and YTD Sales

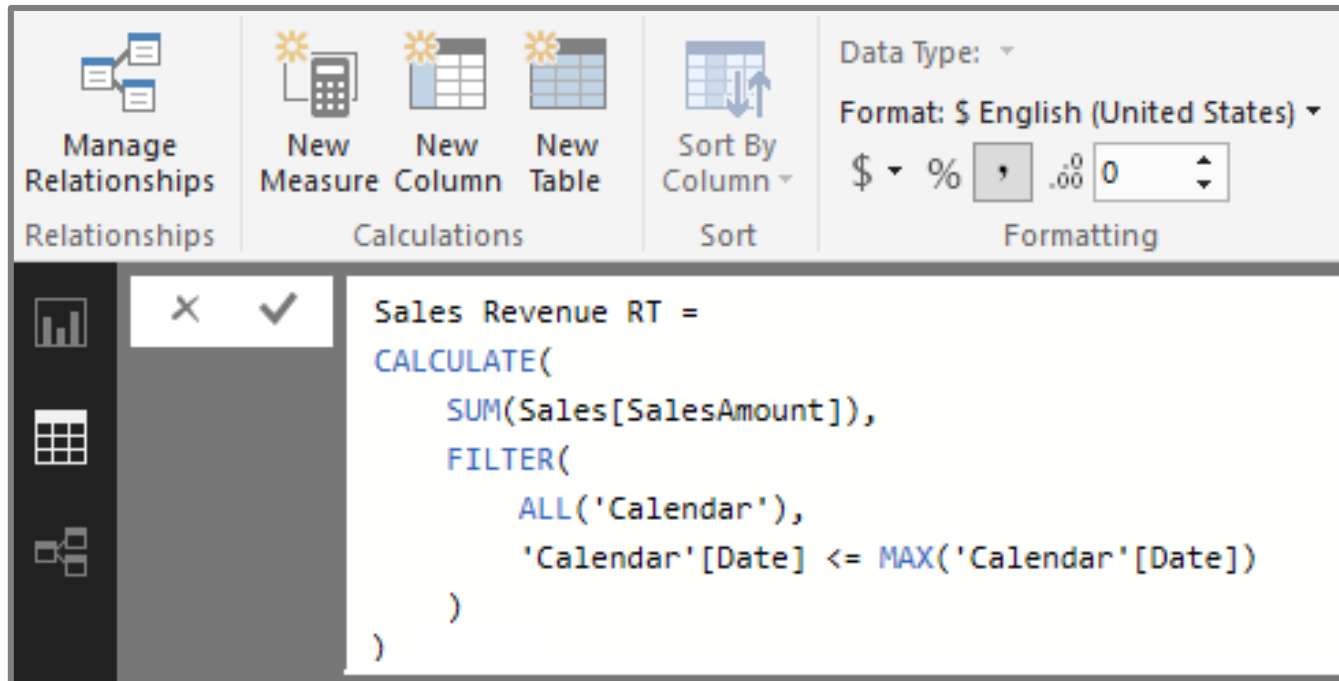- TOTALQTD function calculates quarter-to-date totals
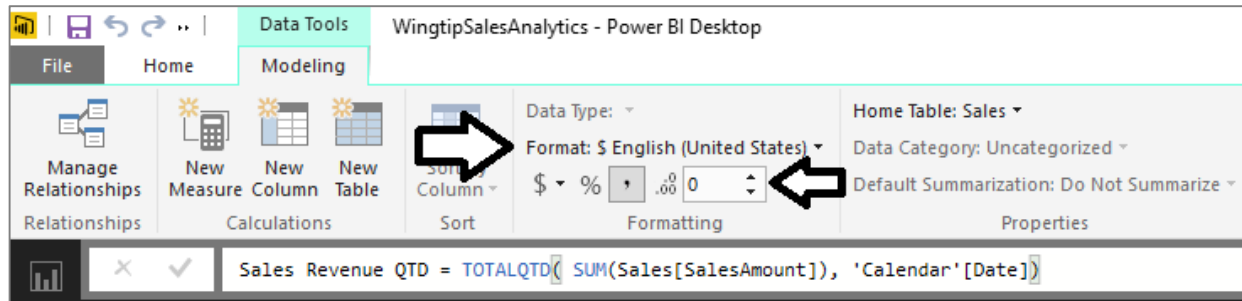
- TOTALYTD function calculates year-to-date totals

Creating Running Total using CALCULATE

- Calculate a running total of sales revenue across years
  - This must be done using **CALCULATE** function


```
Sales Revenue RT =
CALCULATE(
    SUM(Sales[SalesAmount]),
    FILTER(
        ALL('Calendar'),
        'Calendar'[Date] <= MAX('Calendar'[Date])
    )
)
```

Sales Growth PM Measure - First Attempt

- Create a measure named Sales Growth PM

```
Sales Growth PM =
DIVIDE(
  SUM(Sales[SalesAmount]) -
  CALCULATE(
    SUM(Sales[SalesAmount]),
    PREVIOUSMONTH(Calendar[Date])
  ),
  CALCULATE(
    SUM(Sales[SalesAmount]),
    PREVIOUSMONTH(Calendar[Date])
  )
)
```

- Use measure in matrix evaluating month and quarter
  - Measure returns correct value when filtered by Month
  - Measure returns large, erroneous value when filtered by Quarter

Year	Quarter	Month	Sales Revenue	Sales Growth PM
2014	2014-Q1	Jan 2014	$629,969	-18.13 %
		Feb 2014	$609,637	-3.23 %
		Mar 2014	$628,618	3.11 %
		Total	**$1,868,225**	**142.79 %** ⇐
	2014-Q2	Apr 2014	$661,588	5.24 %
		May 2014	$748,193	13.09 %
		Jun 2014	$814,333	8.84 %
		Total	**$2,224,114**	**253.81 %** ⇐
	2014-Q3	Jul 2014	$788,469	-3.18 %

Using the ISFILTERED Function

- ISFILTERED function used to determine when perform evaluation

```
Sales Growth PM =
IF(
    ( ISFILTERED(Calendar[Month]) && NOT(ISFILTERED(Calendar[Date])) ),
    DIVIDE(
        SUM(Sales[SalesAmount]) -
        CALCULATE(
            SUM(Sales[SalesAmount]),
            PREVIOUSMONTH(Calendar[Date])
        ),
        CALCULATE(
            SUM(Sales[SalesAmount]),
            PREVIOUSMONTH(Calendar[Date])
        )
    ),
    BLANK()
)
```

- Expression returns Blank value when evaluation context is invalid

| Year	Quarter	Month	Sales Revenue	Sales Growth PM
2014	2014-Q1	Jan 2014	$629,969	-18.13 %
		Feb 2014	$609,637	-3.23 %
		Mar 2014	$628,618	3.11 %
		Total	**$1,868,225**	
	2014-Q2	Apr 2014	$661,588	5.24 %
		May 2014	$748,193	13.09 %
		Jun 2014	$814,333	8.84 %
		Total	**$2,224,114**	
	2014-Q3	Jul 2014	$788,469	-3.18 %
		Aug 2014	$869,143	10.23 %

Agenda

- ✓ Data Modeling with Power BI Desktop
- ✓ Understanding the DAX Evaluation Context
- ✓ Creating a Dynamic Calendar Table
- ➢ Designing Interactive Reports
- • Understanding Row-level Security (RLS)
- • Publishing PBIX Projects to PowerBI.com

Built-in Visualization Types

- Table and Matrix
- Bar charts and Column charts
- Pie charts and Doughnut chart
- Line chart and Area chart
- Scatter chart and Combo charts
- Card and Multi-row Card
- Treemap
- Waterfall charts
- Funnel charts
- Gauge charts
- Map and Filled Map
- Slicer
- R script visual
- Shape map (in preview)
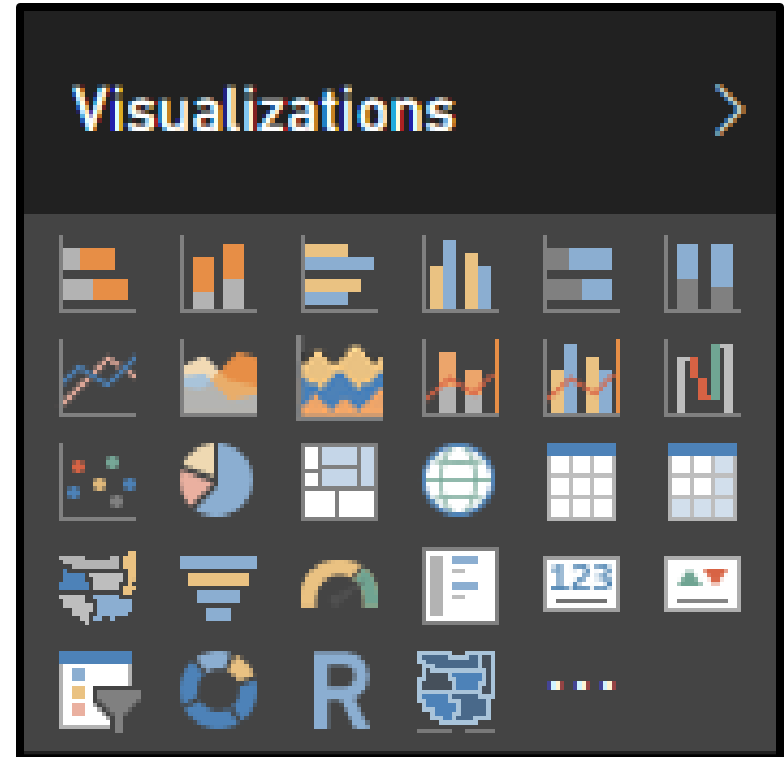
User Interaction with Slicers & Highlighting

- Provides user with interactive filtering control

User Interaction using Drill Actions

- Drill Actions supported when using hierarchies
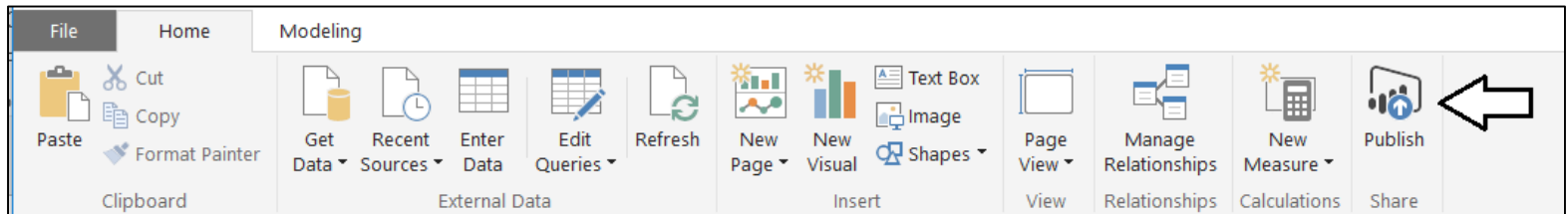 - You must enabled drilldown mode in visual
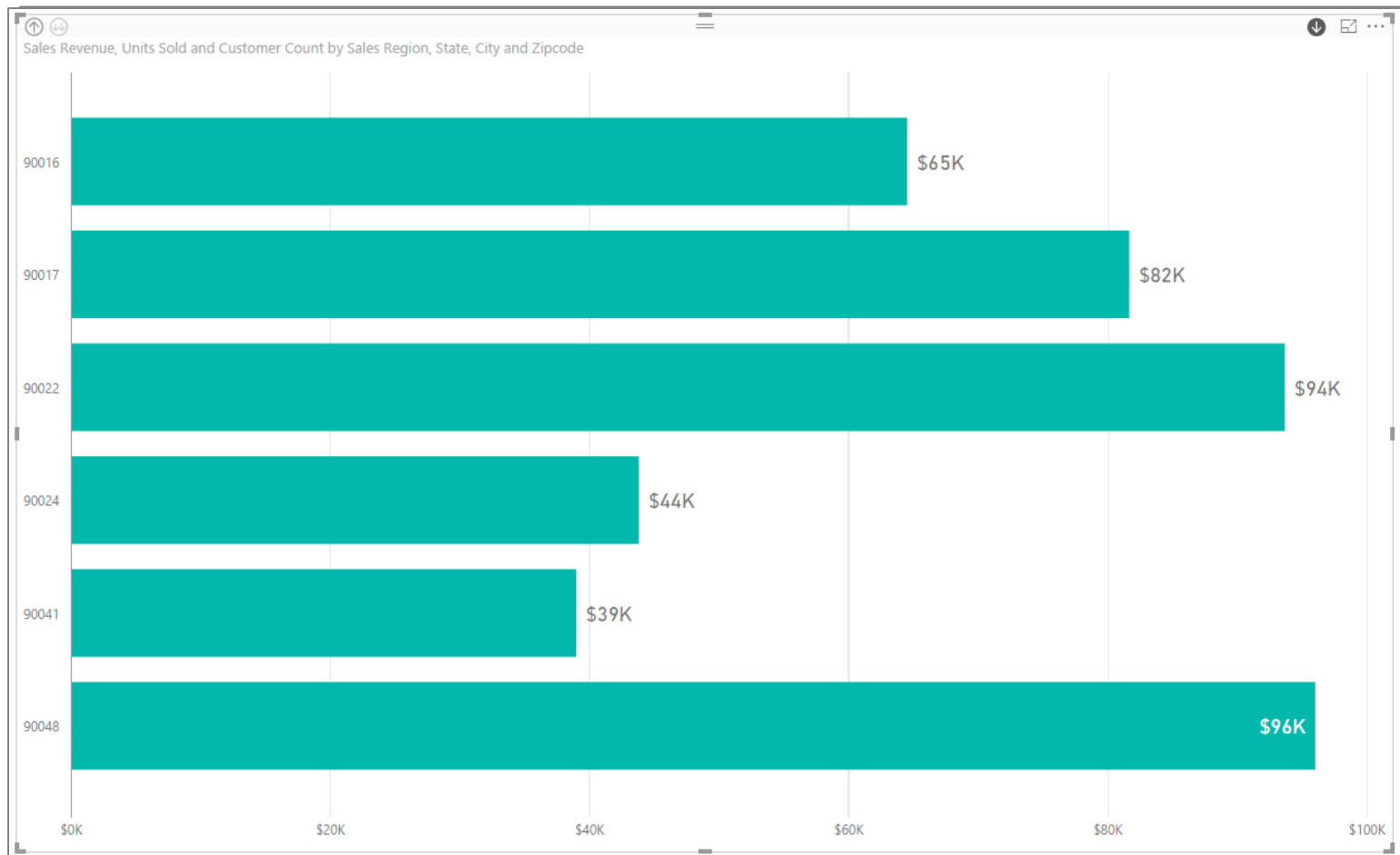
Agenda

- ✓ Data Modeling with Power BI Desktop
- ✓ Understanding the DAX Evaluation Context
- ✓ Creating a Dynamic Calendar Table
- ✓ Designing Interactive Reports
- ➢ Understanding Row-level Security (RLS)
- • Publishing PBIX Projects to PowerBI.com

What Is Row-level Security (RLS)

- Security features for restricting user access
  - Introduced into preview in February of 2016
  - RLS feature set changed in summer of 2016
  - Configuring RLS now requires Power BI Desktop
  - RLS requires all users to have Power BI Pro license
  - This course covers RLS in Module 6 and Module 7
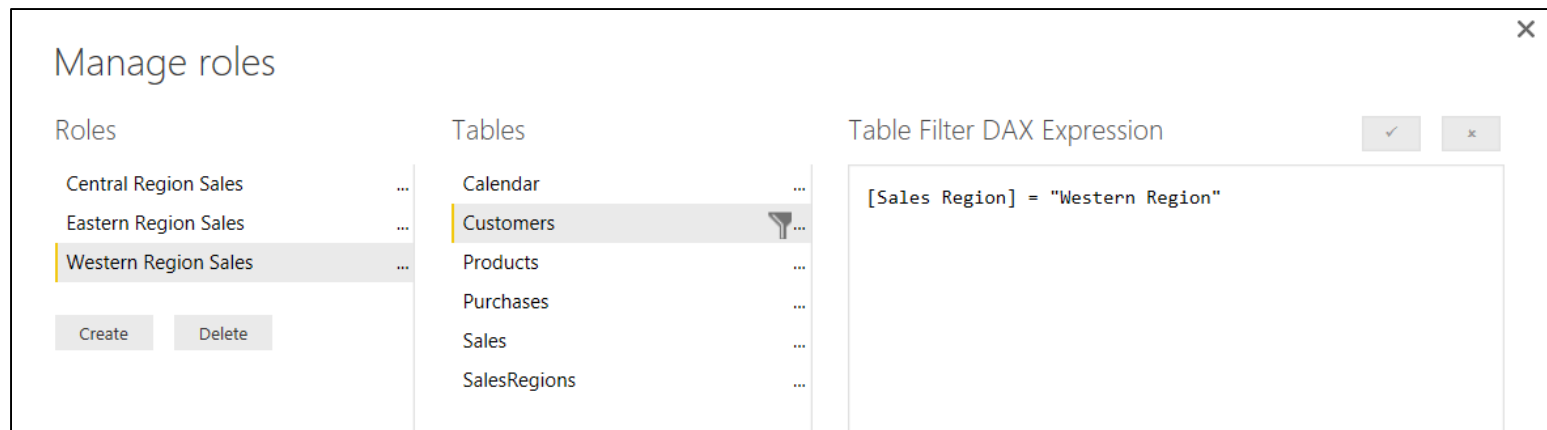
Agenda

- ✓ Data Modeling with Power BI Desktop
- ✓ Understanding the DAX Evaluation Context
- ✓ Creating a Dynamic Calendar Table
- ✓ Designing Interactive Reports
- ✓ Understanding Row-level Security (RLS)
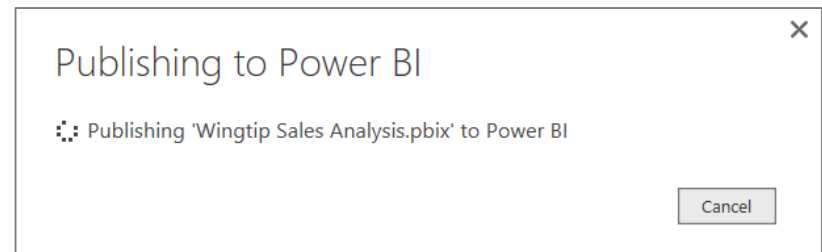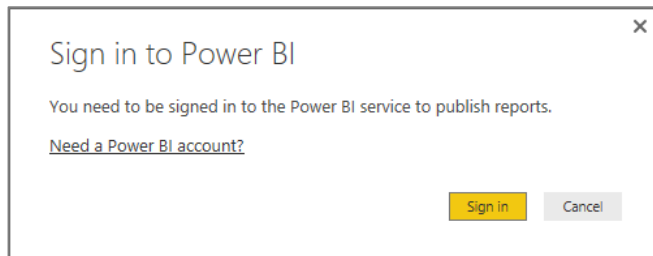- ➢ Publishing PBIX Projects to PowerBI.com

Publishing a Power BI Desktop Project

- Power BI Desktop provides **Publish** command
  - Used to publish project to Power BI service

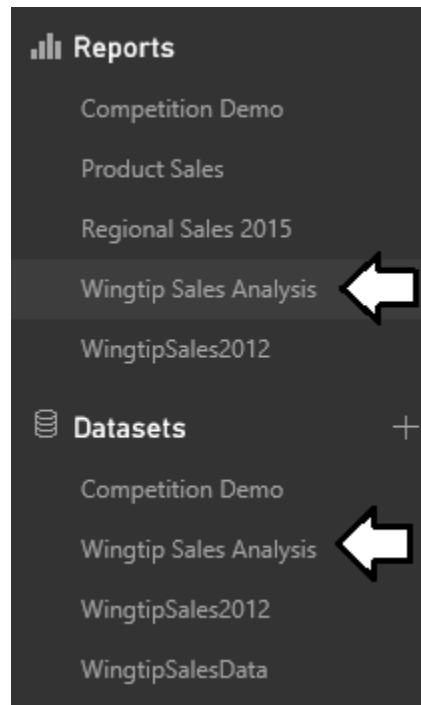 - Requires logging into your Office 365 account

 - Published articles added to a specific workspace

Examining What's Been Published

- What does project publishing add to workspace?
  - One dataset with same name as project
  - One report with same name as project

Dataset Configuration

- You can configure Dataset after its been published
  - Configure data source credentials
  - Configure refresh schedule
  - Configure Row-level Security
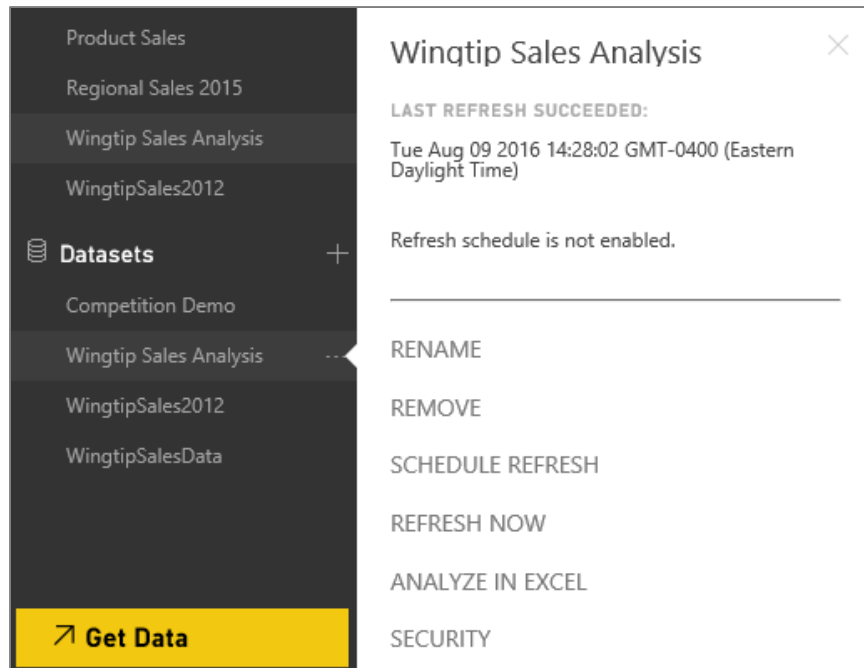
Summary

- ✓ Data Modeling with Power BI Desktop
- ✓ Understanding the DAX Evaluation Context
- ✓ Creating a Dynamic Calendar Table
- ✓ Designing Interactive Reports
- ✓ Understanding Row-level Security (RLS)
- ✓ Publishing PBIX Projects to PowerBI.com