

Lab 03A

Developing a Power BI Custom Visual

Overview

The estimated time to complete the lab is 90 minutes

*You must successfully complete **Lab01A** before commencing this lab. If you did not successfully complete **Lab02A**, you can work with the provided Power BI Desktop report.*

In this lab, you will develop a Power BI custom visual named Circle Card to display a formatted measure value inside a circle. The Circle Card will support customization of fill color and thickness of its outline.

When imported into your Power BI Desktop report, the cards will be modified to become Circle Cards.



You will learn how to:

- Create a Power BI custom visual
- Develop the custom visual with D3 visual elements
- Configure data binding with the visual elements
- Format data values
- Add visual properties
- Package the visual
- Import the custom visual into a Power BI Desktop report

Creating a Custom Visual

In this exercise, you will create a custom visual, and also setup the development environment in the Power BI service.

Creating a Custom Visual

In this task, you will create a custom visual.

1. Open Windows PowerShell.



2. First, verify that the Power BI Visual Tools package has been installed.

Command

```
pbviz
```

3. Review the output, including the list of supported commands.

```
Commands:
new [name]      Create a new visual
info           Display info about the current visual
start          Start the current visual
package        Package the current visual into a pbviz file
update [version] Updates the api definitions and schemas in the current visual.
help [cmd]     display help for [cmd]
```

4. Navigate to your **MySolution** folder, using the folder path you used to install the course files.

Command

```
cd <CourseFolder>\PowerBIDev\MySolution
```

5. To create a custom visual project, enter the following command.

Command

```
pbviz new CircleCard
```

CircleCard is the name of the project.

6. Navigate to the project folder.

Command

```
cd CircleCard
```

7. Start the custom visual.

Command

```
pbviz start
```

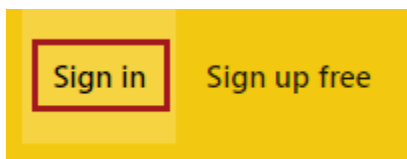
The visual is now running, hosted on your computer, and is ready for testing.

8. Leave PowerShell open.

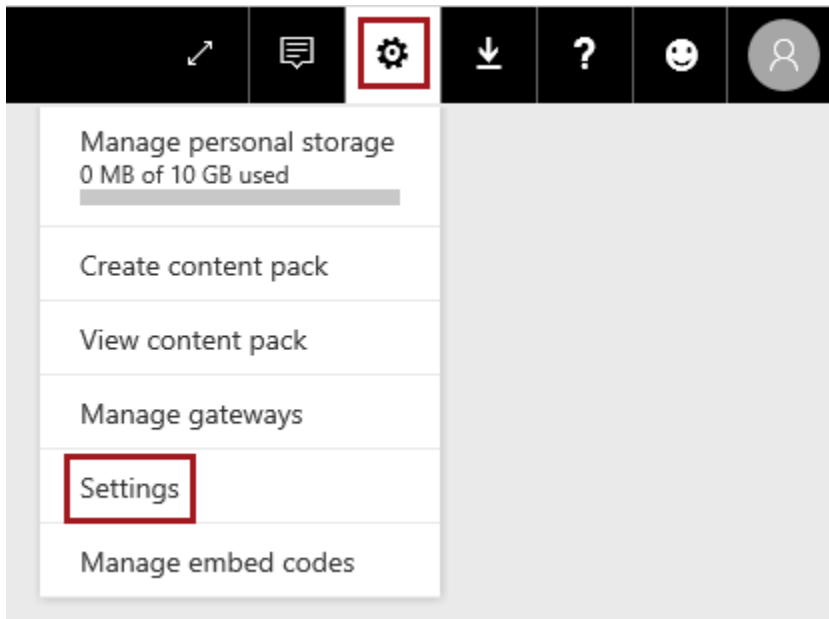
Setting Up the Development Environment

In this task, you will sign in to Power BI, and then enable the developer settings. You will then upload a Power BI Desktop report, and then edit the report to display the custom visual.

1. Open a web browser, and then navigate to <https://powerbi.com/>.
2. At the top-right corner, click **Sign In**, and then complete the sign in process.



3. Once signed in, at the top-right corner, click **Settings** (cog icon), and then select **Settings**.



- At the left, select the **Developer** page.

Preview features

Privacy

Language

Close account

Developer

ArcGIS Maps for Power BI (Preview)

- In the right pane, check the **Enable Developer Visual for Testing** checkbox.

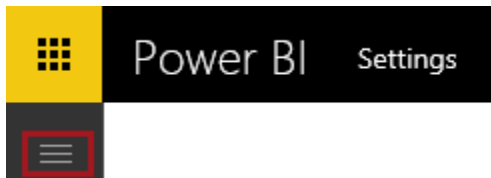
Developer Settings



Enable developer visual for testing

[Learn more](#)

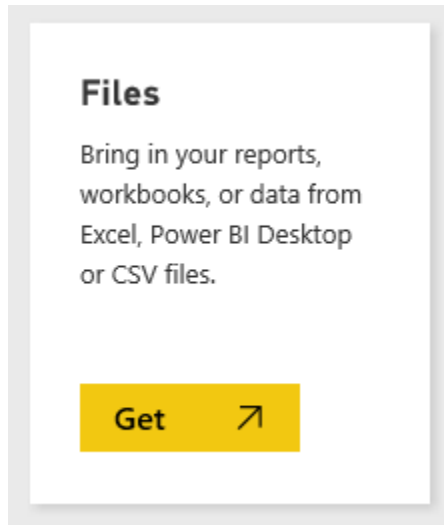
- To upload a Power BI Desktop report, first open the **Navigation** pane.



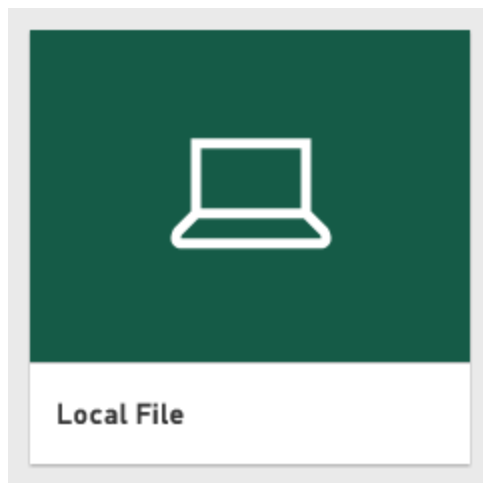
- In the **Navigation** pane, at the bottom-left corner, click **Get Data**.



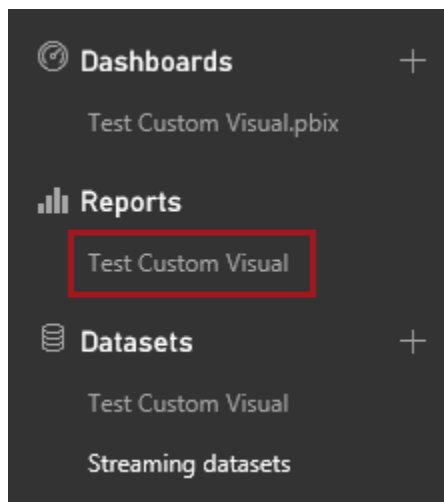
8. Select the **Files** tile.



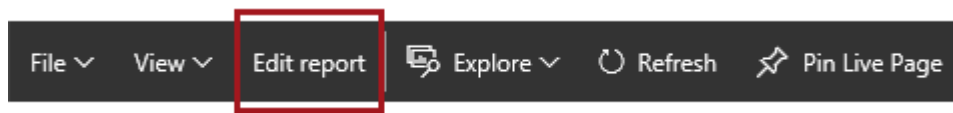
9. Select the **Local File** tile.



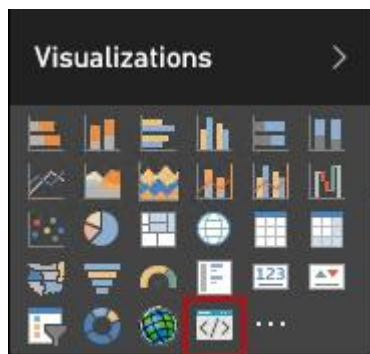
10. In the **Choose File to Upload** window, navigate to **<CourseFolder>\PowerBIDev\Lab03A\Assets** folder.
11. Select **Test Custom Visual.pbix**, and then click **Open**.
This Power BI Desktop report is a light-weight solution containing just two fields to support the testing of the custom visual.
12. Once uploaded, in the **Navigation** pane, select the **Test Custom Visual** report.



13. To edit the report, on the toolbar, click **Edit Report**.

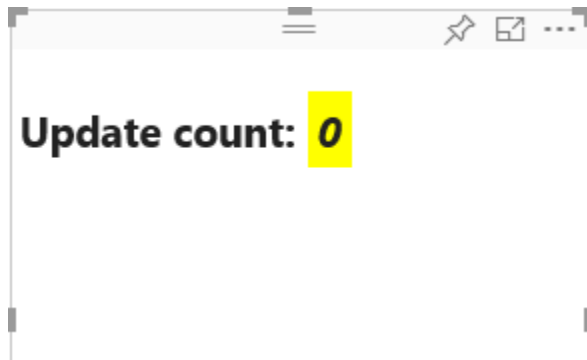


14. In the **Visualizations** pane, select the **Developer Visual**.



This visualization represents the custom visual that you started on your computer. It is only available when the developer settings have been enabled.

15. Notice that a visualization was added to the report canvas.



*This is a very simple visual that displays the number of times its **Update** method has been called. At this stage the visual does not yet retrieve any data.*

16. Resize the visual, and notice that the update value increments.
17. In PowerShell, to stop the custom visual running, enter **Ctrl+C**, and when prompted to terminate the batch job, enter **Y**, and then press **Enter**.

Tip: During the development and testing of your custom visual, if at any stage you receive errors or unexpected behavior, you should stop and restart the custom visual. Issues can be due to an inconsistent project build, often due to dependencies being added—or, because project files have not yet been saved.

You have now setup the development environment. In the next exercise, you will commence developing the custom visual by using D3 visual elements.

Adding Visual Elements

In this exercise, you will install the D3 JavaScript library and typings, configure file dependencies, and then develop the custom visual to display a circle and text.

D3 is a JavaScript library for producing dynamic, interactive data visualizations in web browsers. It makes use of widely implemented SVG, HTML5, and CSS standards.

Adding D3 to the Project

In this task, you will add the D3 JavaScript library and typings.

1. To install D3, in PowerShell, enter the following command.

*For your convenience, many text entries in this lab can be copied from the **<CourseFolder>PowerBIDev\Lab03A\Assets\Snippets.txt** file.*

To paste the clipboard content into PowerShell, simply right-click at the command prompt.

Command

```
npm i d3@3.5.5 --save
```

2. To install typings, enter the following command.

Command

```
typings install --save --global  
d3=github:DefinitelyTyped/DefinitelyTyped/d3/d3.d.ts#6e2f2280ef16ef277049d0ce8583af167d  
586c59 dt~jquery#1.10.0+20160929162922 dt~lodash#4.14.0+20161110215204
```

This command installs TypeScript definitions based on JavaScript files, enabling you to develop the custom visual in TypeScript (which is a superset of JavaScript). Visual Studio Code is an ideal IDE for developing TypeScript applications.

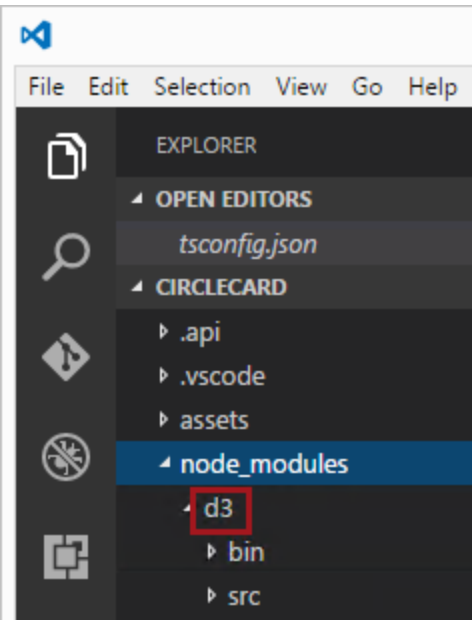
3. To launch Visual Studio Code, in PowerShell, enter the following command.

Command

```
code .
```

4. If you are redirected to the Visual Studio Getting Started page, set the focus back to Visual Studio Code.

5. In Visual Studio Code, in the **Explorer** pane (located at the left), expand the **node_modules** folder, and then verify that **d3** was installed.



6. In the **Explorer** pane, expand **typings** ► **globals** ► **d3**, and notice the TypeScript file **index.d.ts**.
7. To open the project file, in the **Explorer** pane, select **tsconfig.json**.
8. To register d3, enter the following two file references into the **files** array, and be sure to add a comma after the **visual.ts** file reference.

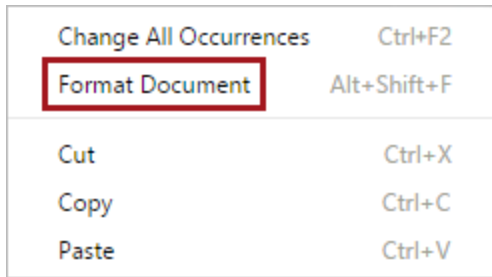
*Tip: Copy the file references from the **Snippets.txt** file.*

JSON

```
"typings/index.d.ts",  
"node_modules/d3/d3.min.js"
```

```
9      },  
10     "files": [  
11       ".api/v1.4.0/PowerBI-visuals.d.ts",  
12       "src/visual.ts",  
13       "typings/index.d.ts",  
14       "node_modules/d3/d3.min.js"  
15     ]  
16   }
```

9. To format the document, right-click anywhere in the document, and then select **Format Document**.

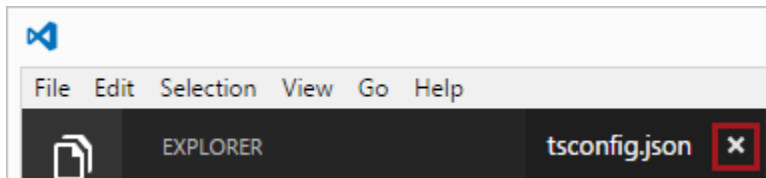


To improve readability, it is recommended that you format the document every time that paste in code snippets.

10. To save the file change, enter **Ctrl+S**.

It is important to be disciplined and save files as you make changes. Unsaved files may result in an inconsistent and unexpected build of the project when the visual project is running.

11. Close the **tsconfig.json** file.



Developing the Visual Elements

In this task, you will develop the custom visual to display a circle and sample text.

1. In the **Explorer** pane, expand the **src** folder, and then select **visual.ts**.
2. Notice the comments at the top of the file.

Permission to use the Power BI custom visual packages is granted free of charge under the terms of the MIT License. As part of the agreement, you must leave the comments at the top of the file.

3. To remove the default custom visual logic from the **Visual** class, remove:
 - The two class-level private variable declarations
 - The three lines of code from the constructor
 - The two lines of code from the **update** method

4. Verify that the module code looks like the following.

```
27 module powerbi.extensibility.visual {
28     export class Visual implements IVisual {
29
30         constructor(options: VisualConstructorOptions) {
31         }
32
33         public update(options: VisualUpdateOptions) {
34         }
35     }
36 }
```

5. Beneath the **Visual** class declaration, insert the following class-level properties.

*Tip: Copy the code from the **Snippets.txt** file.*

TypeScript

```
private host: IVisualHost;
private svg: d3.Selection<SVGElement>;
private container: d3.Selection<SVGElement>;
private circle: d3.Selection<SVGElement>;
private textValue: d3.Selection<SVGElement>;
private textLabel: d3.Selection<SVGElement>;
```

*These declarations are for properties that will reference the visual control itself (**host**), and each of the D3 (SVG) elements.*

6. Verify that the code looks like the following.

```
27 module powerbi.extensibility.visual {
28     export class Visual implements IVisual {
29         private host: IVisualHost;
30         private svg: d3.Selection<SVGElement>;
31         private container: d3.Selection<SVGElement>;
32         private circle: d3.Selection<SVGElement>;
33         private textValue: d3.Selection<SVGElement>;
34         private textLabel: d3.Selection<SVGElement>;
35     }
```

7. Add the following code to the constructor.

TypeScript

```
this.svg = d3.select(options.element)
    .append('svg')
    .classed('circleCard', true);

this.container = this.svg.append("g")
    .classed('container', true);

this.circle = this.container.append("circle")
    .classed('circle', true);

this.textValue = this.container.append("text")
    .classed("textValue", true);

this.textLabel = this.container.append("text")
    .classed("textLabel", true);
```

This code adds an SVG group inside the visual, and then adds three shapes: a circle and two text elements.

8. Add the following code to the **update** method.

TypeScript

```
let width: number = options.viewport.width;
let height: number = options.viewport.height;

this.svg.attr({
    width: width,
    height: height
});

let radius: number = Math.min(width, height) / 2.2;

this.circle
    .style("fill", "white")
    .style("fill-opacity", 0.5)
    .style("stroke", "black")
    .style("stroke-width", 2)
    .attr({
        r: radius,
        cx: width / 2,
        cy: height / 2
    });

let fontSizeValue: number = Math.min(width, height) / 5;

this.textValue
    .text("Value")
    .attr({
        x: "50%",
```

```

        y: "50%",
        dy: "0.35em",
        "text-anchor": "middle"
    }).style("font-size", fontSizeValue + "px");

let fontSizeLabel: number = fontSizeValue / 4;

this.textLabel
    .text("Label")
    .attr({
        x: "50%",
        y: height / 2,
        dy: fontSizeValue / 1.2,
        "text-anchor": "middle"
    }).style("font-size", fontSizeLabel + "px");

```

This code sets the width and height of the visual, and then initializes the attributes and styles of the visual elements.

9. Save the **visual.ts** file.
10. In PowerShell, start the custom visual.

Command

```
pbviz start
```

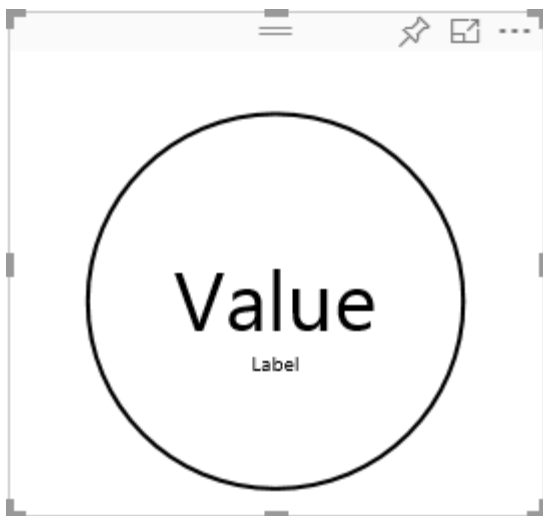
If you encounter errors, use the error notifications (file and line numbers) to help guide you to determine the location of the error.

11. In Power BI, in the toolbar floating above the visual, click **Toggle Auto Reload**.



This option ensures that the visual is automatically reloaded each time you save project changes.

12. Verify that the visual looks like the following.



13. Resize the visual, and notice that the circle and text value scales to fit the available dimension of the visual.

*The **update** method is called continuously with each resize, and this results in the fluid rescaling of the visual elements.*

You have now developed the visual elements. In the next exercise, you will configure data binding.

Configuring Data Binding

In this exercise, you will first define the data roles and data view mappings, and then modify the custom visual logic to display the value and display name of a measure.

Configuring the Capabilities

In this task, you will modify the **capabilities.json** file to define the data role and data view mappings.

1. In Visual Studio code, open the **capabilities.json** file.
2. From inside the **dataRoles** array, remove all content (lines 3-12).
3. Inside the **dataRoles** array, insert the following content.

JSON

```
{
  "displayName": "Measure",
  "name": "measure",
  "kind": "Measure"
}
```

The **dataRoles** array now defines a single data role of type **measure**, that is named **measure**, and will be displayed as **Measure**. This data role will allow passing in either a measure field, or a field that is summarized.

4. From inside the **dataViewMappings** array, remove all content.
5. Inside the **dataViewMappings** array, insert the following content.

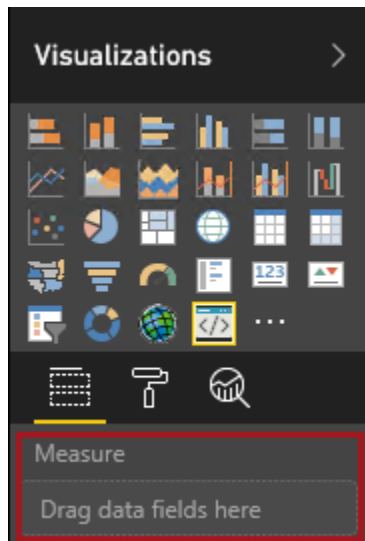
JSON

```
{
  "conditions": [
    { "measure": { "max": 1 } }
  ],
  "single": {
    "role": "measure"
  }
}
```

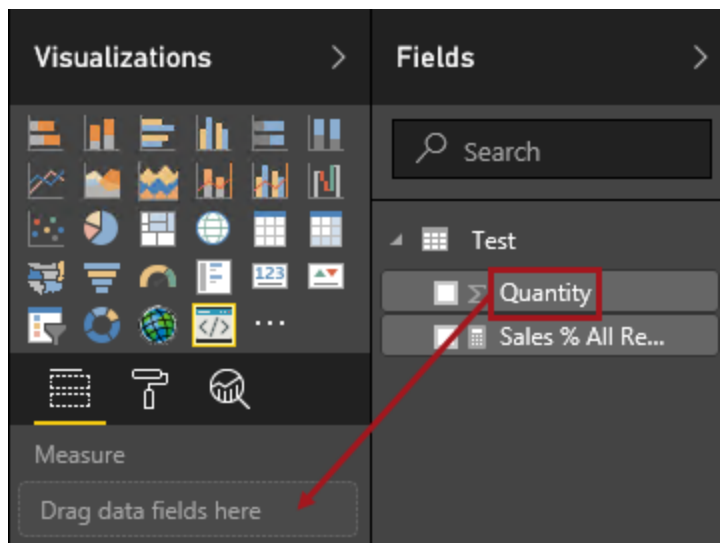
The **dataViewMappings** array now defines that at most, one field can be passed to the data role named **measure**.

6. Save the **capabilities.json** file.

7. In Power BI, notice that the that the visual now can be configured with **Measure**.



8. From the **Fields** pane, drag the **Quantity** field into **Measures**.



The visual project does not yet include data binding logic, and you will develop this logic later in the exercise.

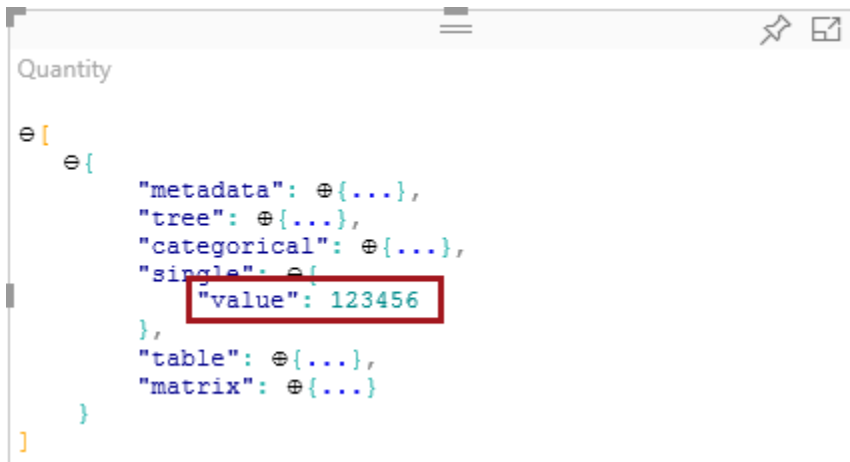
Exploring the Dataview

In this task, you will explore the dataview.

1. In the toolbar floating above the visual, click **Show Dataview**.



2. Expand down into **single**, and then notice the **value** value.



You will update the text in the visual to display the **value** value.

3. Expand down into **metadata**, and then into the **columns** array, and in particular notice the **format** and **displayName** values.

It may be necessary to widen the visual.



You will configure formatting based on the **format** value in the next exercise. And you will update the text in the visual to display the **displayName** value.

4. To toggle back to the visual, in the toolbar floating above the visual, click **Show Dataview**.



Configuring Data Binding

In this task, you will modify the visual logic to display the measure value and display name.

1. In Visual Studio Code, in the **visual.ts** file, add the following statement as the first statement of the **update** method.

TypeScript

```
let dataView: DataView = options.dataViews[0];
```

This statement assigns the dataview to a variable for easy access.

```
54 public update(options: VisualUpdateOptions) {  
55     let dataView: DataView = options.dataViews[0];  
56     let width: number = options.viewport.width;  
57     let height: number = options.viewport.height;
```

This statement declares a variable, and sets it to reference the dataview object.

2. In the **update** method, for **this.textValue**, replace **.text("Value")** with the following.

TypeScript

```
.text(dataView.single.value as string)
```

```
79 this.textValue  
80     .text(dataView.single.value as string)  
81     .attr({  
82         x: "50%",  
83         y: "50%",  
84         dy: "0.35em",  
85         "text-anchor": "middle"  
86     }).style("font-size", fontSizeValue + "px");
```

3. In the **update** method, for **this.textLabel**, replace **.text("Label")** with the following.

TypeScript

```
.text(dataView.metadata.columns[0].displayName)
```

```
90  this.textLabel
91      .text(dataView.metadata.columns[0].displayName)
92      .attr({
93          x: "50%",
94          y: height / 2,
95          dy: fontSizeValue / 1.2,
96          "text-anchor": "middle"
97      }).style("font-size", fontSizeLabel + "px");
```

4. Save the **visual.ts** file.
5. In Power BI, review the visual, which now displays the value and display name.



You have now configured the data roles, and bound the visual to the dataview. In a later exercise, you will format the value according to the format defined in the Power BI Desktop report.

Installing Power BI Utilities

In this exercise, you will install Power BI utilities to help with working with the dataview and also with numeric formatting.

Installing Power BI Utilities

In this task, you will install Power BI utilities.

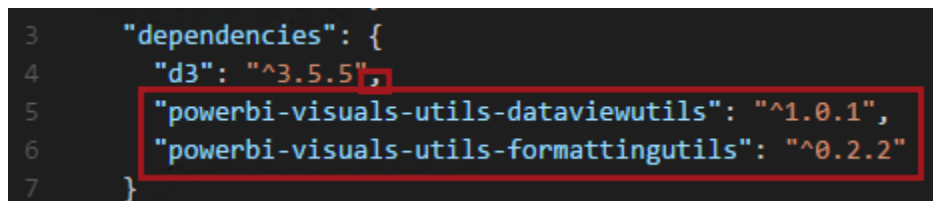
1. In PowerShell, stop the visual.

Whenever you add additional dependencies to the project, you must stop and restart the visual.

2. In Visual Studio Code, open the **package.json** file.
3. Enter the following two dependencies into the **dependencies** array, and be sure to add a comma after the **d3** reference.

JSON

```
"powerbi-visuals-utils-dataviewutils": "^1.0.1",  
"powerbi-visuals-utils-formattingutils": "^0.2.2"
```



```
3  "dependencies": {  
4    "d3": "^3.5.5",  
5    "powerbi-visuals-utils-dataviewutils": "^1.0.1",  
6    "powerbi-visuals-utils-formattingutils": "^0.2.2"  
7  }
```

4. Save the **package.json** file.
5. In PowerShell, to install the dependencies, enter the following command.

Command

```
npm install
```

6. Open the **tsconfig.json** file.
7. Enter the following four file references into the **files** array, and be sure to add a comma after the previous file.

JSON

```
"node_modules/powerbi-visuals-utils-typeutils/lib/index.d.ts",  
"node_modules/powerbi-visuals-utils-svgutils/lib/index.d.ts",  
"node_modules/powerbi-visuals-utils-dataviewutils/lib/index.d.ts",  
"node_modules/powerbi-visuals-utils-formattingutils/lib/index.d.ts"
```

These register the TypeScript files for the project.

```

10  "files": [
11    ".api/v1.4.0/PowerBI-visuals.d.ts",
12    "src/visual.ts",
13    "typings/index.d.ts",
14    "node_modules/d3/d3.min.js",
15    "node_modules/powerbi-visuals-utils-typeutils/lib/index.d.ts",
16    "node_modules/powerbi-visuals-utils-svgutils/lib/index.d.ts",
17    "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.d.ts",
18    "node_modules/powerbi-visuals-utils-formattingutils/lib/index.d.ts"
19  ]

```

8. Save the **tsconfig.json** file.
9. Open the **pbviz.json** file.
10. Enter the following nine file references into the **externalJS** array.

*Tip: Copy the file references from the **Snippets.txt** file.*

JSON

```

"node_modules/jquery/dist/jquery.min.js",
"node_modules/d3/d3.min.js",
"node_modules/lodash/lodash.min.js",
"node_modules/globalize/lib/globalize.js",
"node_modules/globalize/lib/cultures/globalize.culture.en-US.js",
"node_modules/powerbi-visuals-utils-typeutils/lib/index.js",
"node_modules/powerbi-visuals-utils-svgutils/lib/index.js",
"node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js",
"node_modules/powerbi-visuals-utils-formattingutils/lib/index.js"

```

This registers the JavaScript files for the visual.

```

20  "externalJS": [
21    "node_modules/jquery/dist/jquery.min.js",
22    "node_modules/d3/d3.min.js",
23    "node_modules/lodash/lodash.min.js",
24    "node_modules/globalize/lib/globalize.js",
25    "node_modules/globalize/lib/cultures/globalize.culture.en-US.js",
26    "node_modules/powerbi-visuals-utils-typeutils/lib/index.js",
27    "node_modules/powerbi-visuals-utils-svgutils/lib/index.js",
28    "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js",
29    "node_modules/powerbi-visuals-utils-formattingutils/lib/index.js"
30  ],

```

11. Save the **pbviz.json** file.
12. In the **Explorer** pane, expand the **style** folder, and then select the **visual.less** file.

13. Replace the entire content with the following CSS at-rule.

The styles already defined supported the starter visualization.

CSS

```
@import (less) "node_modules/powerbi-visuals-utils-formattingutils/lib/index.css";
```

The @import CSS at-rule is used to import style rules supplied with the formatting utility.

14. Save the **visual.less** file.
15. In PowerShell, start the visual.

Command

```
pbviz start
```

16. In Power BI, in the toolbar floating above the visual, click **Toggle Auto Reload**.



17. Verify that the visual displays.



You have now added utilities that will be helpful in the next two exercises to format values, and also add visual properties.

Formatting Data Values

In this exercise, you will leverage the formatting utility to format the measure value.

Formatting Data Values

In this task, you will add logic to the custom visual to use the formatting utility to format the measure value.

1. In Visual Studio Code, open the **visual.ts** file.
2. Directly after the module declaration, insert the following statement.

TypeScript

```
import valueFormatter = powerbi.extensibility.utils.formatting.valueFormatter;
```

```
27 module powerbi.extensibility.visual {  
28     import valueFormatter = powerbi.extensibility.utils.formatting.valueFormatter;  
29 }
```

*This statement imports the **valueFormatter** module.*

3. In the **update** method, after the **fontSizeValue** variable declaration, insert the following code.

TypeScript

```
let formatString: string = ►  
    valueFormatter.getFormatStringByColumn(dataView.metadata.columns[0]);  
const formatter = valueFormatter.create({  
    format: formatString  
});
```

This code retrieves the format for the measure, and creates a formatter object.

```
79     let fontSizeValue: number = Math.min(width, height) / 5;  
80  
81     let formatString: string = valueFormatter.getFormatStringByColumn(dataView.metadata.columns[0]);  
82     const formatter = valueFormatter.create({  
83         format: formatString  
84     });
```

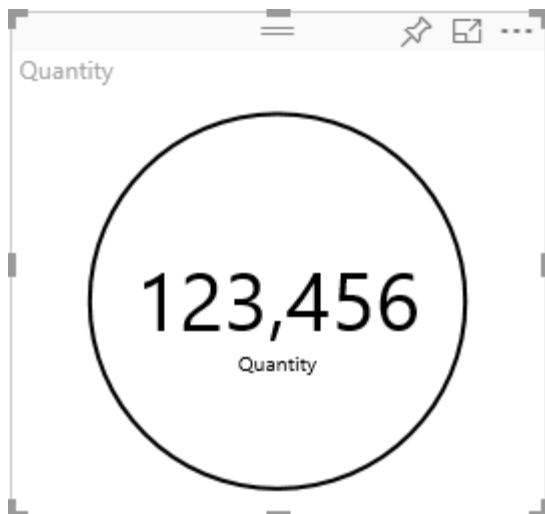

4. In the **update** method, for **this.textValue**, replace **.text(dataView.single.value as string)** with the following code,

TypeScript

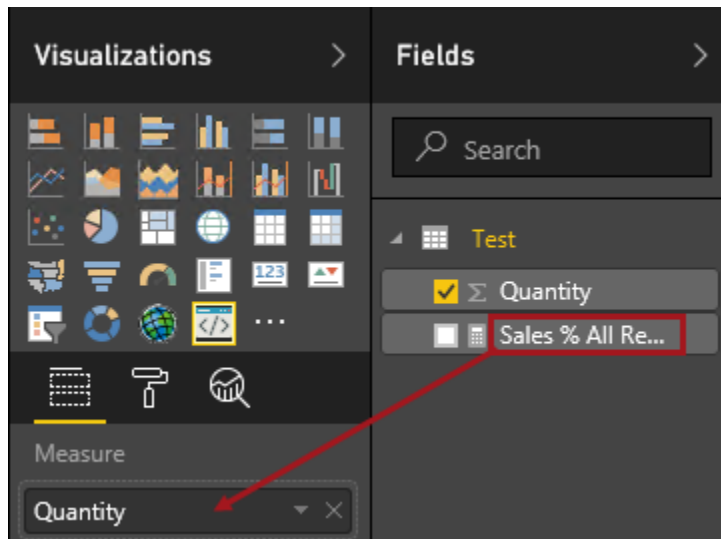
```
.text(formatter.format(dataView.single.value))
```

```
79  this.textValue
80      .text(dataView.single.value as string)
81  .attr({
82      x: "50%",
83      y: "50%",
84      dy: "0.35em",
85      "text-anchor": "middle"
86  }).style("font-size", fontSizeValue + "px");
```

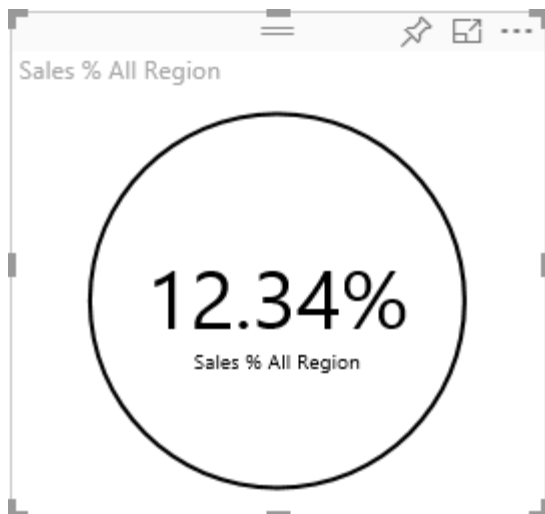
5. Save the **visual.ts** file.
6. In Power BI, verify that the value is now formatted with the thousands separator.



7. Modify the visual to now use the **Sales % All Region** field.



8. Verify that the visual displays the measure with a percentage format.



You have now leveraged the formatting utility to format measure values. In the next exercise, you will add formatting options for the visual.

Adding Formatting Options

In this exercise, you will add formatting options to the visual.

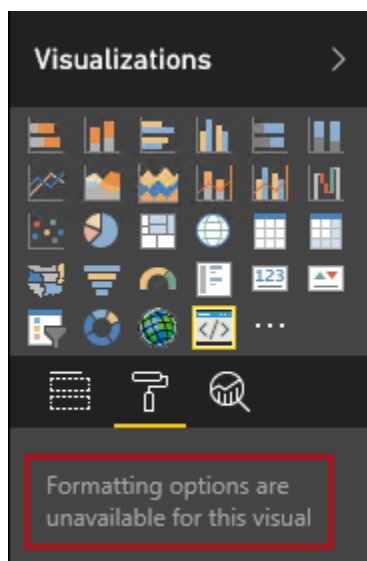
Adding Common Formatting Options

In this task, you will add common properties to the visual.

1. In Power BI, select the **Format** page.



2. Notice that formatting options are not available.

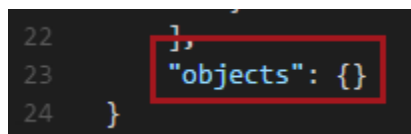


3. In Visual Studio Code, open the **capabilities.json** file.
4. After the **dataViewMappings** array, and the **objects** object (after line 22).

TypeScript

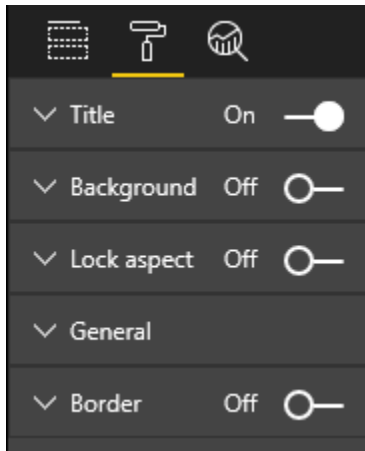
```
,"objects": {}
```

Common formatting options will be added simply adding this object.



5. Save the **capabilities.json** file.

6. In Power BI, review the formatting options now available.



7. Set the **Title** option to **Off**.



8. Notice that the visual no longer displays the measure name at the top-left corner.



Adding Custom Formatting Options

In this task, you will add custom properties to enable configuring the color of the circle, and also the border width.

1. In PowerShell, stop the custom visual.
2. In Visual Studio Code, in the **capabilities.json** file, insert the following JSON fragment into the **objects** object.

JSON

```
"circle": {
  "displayName": "Circle",
  "properties": {
    "circleColor": {
      "displayName": "Color",
      "description": "The fill color of the circle.",
      "type": {
        "fill": {
          "solid": {
            "color": true
          }
        }
      }
    },
    "circleThickness": {
      "displayName": "Thickness",
      "description": "The circle thickness.",
      "type": {
        "numeric": true
      }
    }
  }
}
```

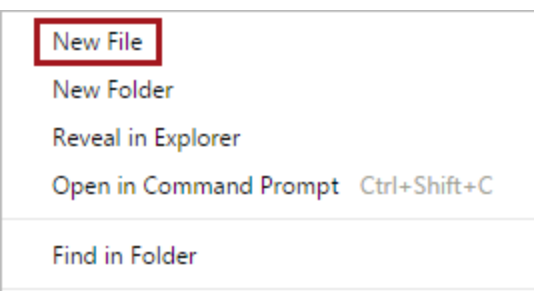
*The JSON fragment describes a group named circle which consists of two options named **circleColor** and **circleThickness**.*

```

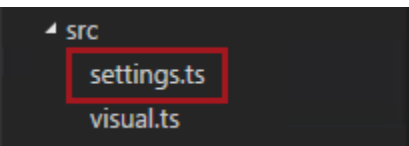
23     "objects": {
24         "circle": {
25             "displayName": "Circle",
26             "properties": {
27                 "circleColor": {
28                     "displayName": "Color",
29                     "description": "The fill color of the circle.",
30                     "type": {
31                         "fill": {
32                             "solid": {
33                                 "color": true
34                             }
35                         }
36                     }
37                 },
38                 "circleThickness": {
39                     "displayName": "Thickness",
40                     "description": "The circle thickness.",
41                     "type": {
42                         "numeric": true
43                     }
44                 }
45             }
46         }
47     }

```

3. Save the **capabilities.json** file.
4. In the **Explorer** pane, right-click the **src** folder, and then select **New File**.



5. Set the name of the file to **settings.ts**.



6. In the **settings.ts** file, insert the following code.

TypeScript

```
module powerbi.extensibility.visual {
    import DataViewObjectParser = ►
    powerbi.extensibility.utils.dataview.DataViewObjectsParser;

    export class CircleSettings {
        public circleColor: string = "white";
        public circleThickness: number = 2;
    }

    export class VisualSettings extends DataViewObjectParser {
        public circle: CircleSettings = new CircleSettings();
    }
}
```

This module defines the two classes.

*The **CircleSettings** class defines two properties with names that match the objects defined in the **capabilities.json** file (**circleColor** and **circleThickness**), and also sets default values.*

*The **VisualSettings** class inherits the **DataViewObjectParser** class, and adds a property named **circle** which matches the object defined in the **capabilities.json** file, and which returns an instance of **CircleSettings**.*

7. Save the **settings.ts** file.
8. Open the **tsconfig.json** file.
9. Enter the following file reference into the **files** array, and be sure to add a comma after the previous file.

JSON

```
"src/settings.ts"
```

```
10  "files": [
11    ".api/v1.4.0/PowerBI-visuals.d.ts",
12    "src/visual.ts",
13    "typings/index.d.ts",
14    "node_modules/d3/d3.min.js",
15    "node_modules/powerbi-visuals-utils-typeutils/lib/index.d.ts",
16    "node_modules/powerbi-visuals-utils-svgutils/lib/index.d.ts",
17    "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.d.ts",
18    "node_modules/powerbi-visuals-utils-formattingutils/lib/index.d.ts",
19    "src/settings.ts",
20  ]
```

10. Save the **tsconfig.json** file.
11. Open the **visual.ts** file.

12. In the **Visual** class, add the following property, immediately below the others.

TypeScript

```
private visualSettings: VisualSettings;
```

*This property will store a reference to the **VisualSettings** object, describing the visual settings.*

```
30     export class Visual implements IVisual {
31         private host: IVisualHost;
32         private svg: d3.Selection<SVGElement>;
33         private container: d3.Selection<SVGElement>;
34         private circle: d3.Selection<SVGElement>;
35         private textValue: d3.Selection<SVGElement>;
36         private textLabel: d3.Selection<SVGElement>;
37         private visualSettings: VisualSettings;
38     }
```

13. In the **Visual** class, add the following method before the **update** method.

TypeScript

```
public enumerateObjectInstances(options: EnumerateVisualObjectInstancesOptions): ►
    VisualObjectInstanceEnumeration {
    const settings: VisualSettings = this.visualSettings || ►
        VisualSettings.getDefault() as VisualSettings;

    return VisualSettings.enumerateObjectInstances(settings, options);
}
```

This method is used to populate the formatting options.

14. In the **update** method, after the declaration of the **radius** variable, add the following code.

TypeScript

```
this.visualSettings = VisualSettings.parse<VisualSettings>(dataView);

this.visualSettings.circle.circleThickness = ►
    Math.max(0, this.visualSettings.circle.circleThickness);
this.visualSettings.circle.circleThickness = ►
    Math.min(10, this.visualSettings.circle.circleThickness);
```

*This code retrieves the format options, and also adjusts any value passed into the **circleThickness** property, converting it to 0 if negative, or 10 if a value greater than 10.*

```
73         let radius: number = Math.min(width, height) / 2.2;
74
75         this.visualSettings = VisualSettings.parse<VisualSettings>(dataView);
76
77         this.visualSettings.circle.circleThickness = Math.max(0, this.visualSettings.circle.circleThickness);
78         this.visualSettings.circle.circleThickness = Math.min(10, this.visualSettings.circle.circleThickness);
79     }
```


15. For the **circle** element, modify the value passed to the **fill** style to the following expression.

TypeScript

```
this.visualSettings.circle.circleColor
```

```
80     this.circle
81         .style("fill", this.visualSettings.circle.circleColor)
82         .style("fill-opacity", 0.5)
83         .style("stroke", "black")
84         .style("stroke-width", 2)
85         .attr({
86             r: radius,
87             cx: width / 2,
88             cy: height / 2
89         });
```

16. For the **circle** element, modify the value passed to the **stroke-width** style to the following expression.

TypeScript

```
this.visualSettings.circle.circleThickness
```

```
80     this.circle
81         .style("fill", this.visualSettings.circle.circleColor)
82         .style("fill-opacity", 0.5)
83         .style("stroke", "black")
84         .style("stroke-width", this.visualSettings.circle.circleThickness)
85         .attr({
86             r: radius,
87             cx: width / 2,
88             cy: height / 2
89         });
```

17. Save the **visual.ts** file.
18. In PowerShell, start the visual.

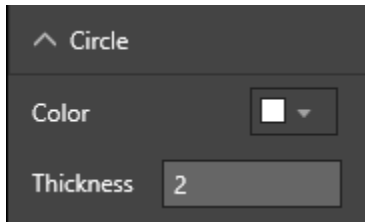
Command

```
pbviz start
```

19. In Power BI, in the toolbar floating above the visual, click **Toggle Auto Reload**.



20. In the visual format options, expand **Circle**.



21. Modify the **Color** option, and also the **Thickness** option.

22. Modify the **Thickness** option to a value less than zero, and also a value greater than 10, and notice that the visual updates the value to a tolerable minimum or maximum.

Having added custom formatting options, you have now completed the custom visual development. In the next exercise, you will package the custom visual.

Packaging the Custom Visual

In this exercise, you will enter property values for the custom visual project, update the icon file, and then package the custom visual.

Packaging the Custom Visual

In this task, you will enter property values for the custom visual project, update the icon file, and then package the custom visual.

1. In PowerShell, stop the custom visual.
2. In Visual Studio Code, open the **pbviz.json** file.
3. In the visual object, modify the **displayName** property to **Circle Card**.

```
1 {  
2   "visual": {  
3     "name": "circleCard",  
4     "displayName": "Circle Card",  
5     "guid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
6     "visualClassName": "Visual",
```

In the **Visualizations** pane, hovering over the icon will reveal the display name.

4. For the description property, enter the following text.

*Tip: Copy the text from the **Snippets.txt** file.*

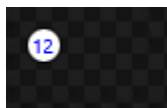
Text

Displays a formatted measure value inside a circle

5. Optionally, in the **author** object, enter your details.
6. Save the **pbviz.json** file.
7. In the **assets** object, notice that the document defines a path to an icon.

*The icon is the image that appears in the **Visualizations** pane. It must be a PNG file, 20 pixels by 20 pixels.*

8. In Windows Explorer, copy the **<CourseFolder>\PowerBIDev\Lab03A\Assets\icon.png** file, and then paste it to replace the default file located at **<CourseFolder>\PowerBIDev\MySolution\circleCard\assets**.
9. In Visual Studio Code, in the **Explorer** pane, expand the **assets** folder, and then select the **icon.png** file.
10. Review the icon.



11. In Visual Studio Code, ensure that all files are saved.
12. To package the custom visual, in PowerShell, enter the following command.

Command

```
pbiviz package
```

*The package is output to the **dist** folder of the project. The package contain everything required to import the custom visual into either the Power BI service, or a Power BI Desktop report.*

You have now packaged the custom visual, and it is now ready for use. In the next exercise, you will import the custom visual into your Power BI Desktop report.

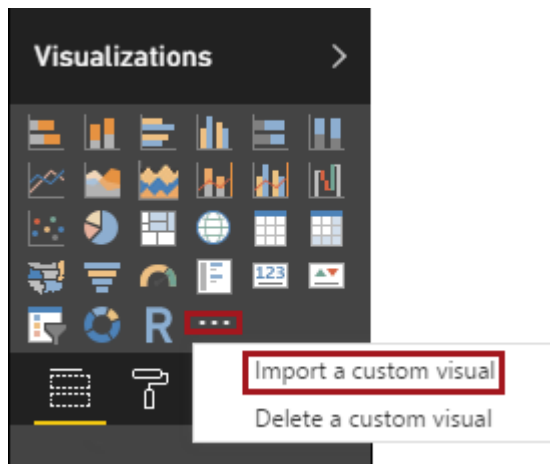
Importing the Custom Visual

In this exercise, you will open the Power BI Desktop report, and then import the Circle Card custom visual. You will then modify the two card visualizations to become Circle Cards, and then modify the circle format options.

Importing the Custom Visual

In this task, you will open the Power BI Desktop report, and import the Circle Card custom visual.

1. Open Power BI Desktop, and the analytic solution you developed, located at **<CourseFolder>\PowerBIDev\MySolution\US Sales Analysis.pbix**.
2. In the **Visualizations** pane, click the ellipsis, and then select **Import a Custom Visual**.

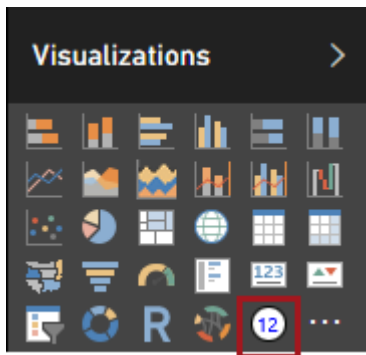


3. In the import window, click **Import**.



1. In the **Open** window, navigate to the **<CourseFolder>\PowerBIDev\MySolution\circleCard\dist** folder.
2. Select the **circleCard.pbiviz** file, and then select **Open**.
3. When the visual has successfully imported, click **OK**.

4. Verify that the visual has been added to the **Visualizations** pane.

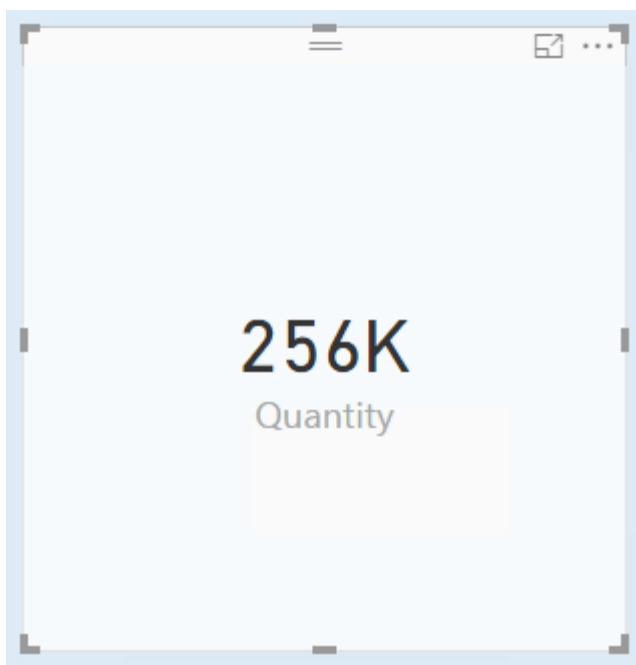


5. Hover over the **Circle Card** icon, and notice the tooltip that appears.

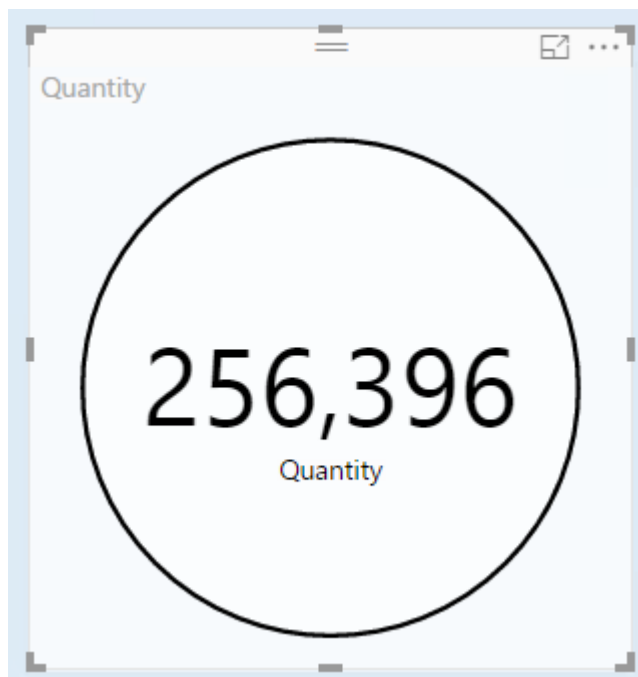
Modify the Card Visualizations

In this task, you will modify the two card visualizations to become Circle Cards, and then modify the circle format options.

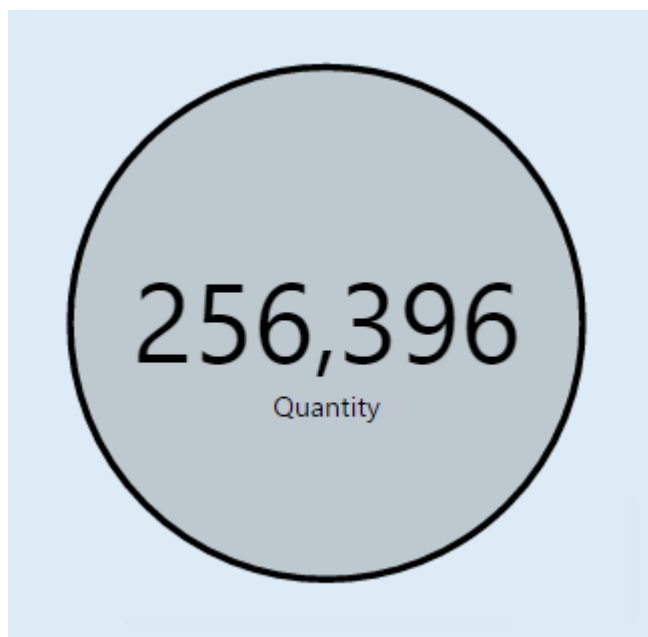
1. In the report page, select the **Quantity** card.



2. In the **Visualizations** pane, select the **Circle Card** visualization.



3. Set the following formatting options.
 - Circle color: Light grey
 - Thickness: 3
 - Title: Off
 - Background: Off
4. Verify that the **Quantity** Circle Card looks like the following.



5. Modify the **Sales % All Regions** card also to a Circle Card, and then set the following options.
 - Circle color: Light blue
 - Thickness: 3
 - Title: Off
 - Background: Off
6. Verify that the **Sales % All Regions** Circle Card looks like the following.



7. Save the Power BI Desktop report.
8. Close Power BI Desktop.

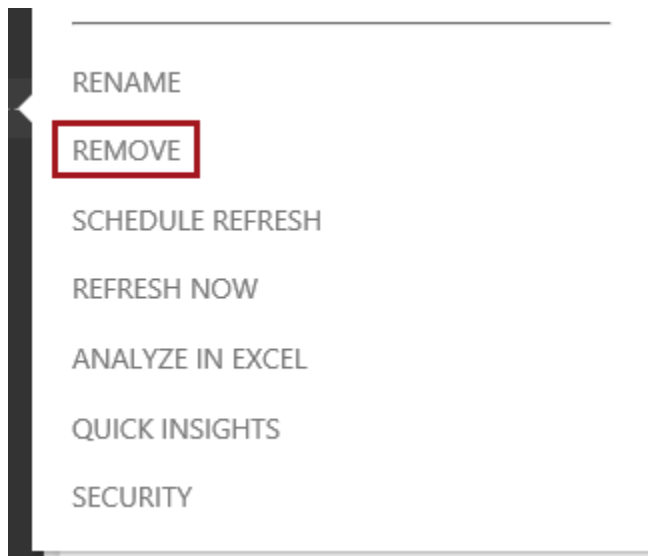
*The Power BI Desktop report is now ready to embed into an application. You will achieve this in **Lab 04A**.*

Finishing Up

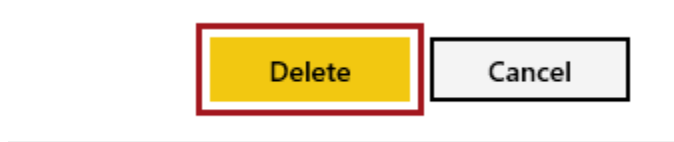
In this exercise, you will finish up by closing application, and removing the dataset and dashboard added to your Power BI workspace.

Finishing Up

1. Close PowerShell.
2. Close Visual Studio Code.
3. In Power BI, in the **Navigation** pane, inside the **Datasets** group, right-click **Test Custom Visual**, and then select **Remove**.



4. When prompted to delete the dataset, click **Delete**.



5. When prompted to save changes to the report, click **Don't Save**.



The report will be deleted as a consequence of deleting the dataset.

6. Delete the **Test Custom Visual** dashboard also.
7. Sign out of Power BI.

There is no further requirement to work with the Power BI service in this course.

Summary

In this lab, you developed a Power BI custom visual named Circle Card to display a formatted measure value inside a circle. The Circle Card supports customization of fill color and thickness of its outline.

When imported into your Power BI Desktop report, the cards were modified to become Circle Cards.

Terms of Use

© 2017 Microsoft Corporation. All rights reserved.

By using this hands-on lab, you agree to the following terms:

The technology/functionality described in this hands-on lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the hands-on lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. Without written permission, you may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this hands-on lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION WITHOUT WRITTEN PERMISSION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCTIONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK If you give feedback about the technology features, functionality and/or concepts described in this hands-on lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

DISCLAIMER This lab contains only a portion of new features and enhancements in Microsoft Power BI. Some of the features might change in future releases of the product.

Document Version

#	Date	Author	Comments
1	17-JAN-2017	Peter Myers	Power BI Desktop v2.42.4611.701 64-bit Visual Studio Code v1.8.1
2	23-FEB-2017	Peter Myers	Power BI Desktop v2.43.4647.541 64-bit Node.js v7.4.0 typings v2.1.0 pbviz v1.5.0