Data Visualization

(A) Viewing Data from Static Google Charts (external html files)

You may see a gallery of charts in the following website
https://developers.google.com/chart/interactive/docs/gallery

Click on the chart type you want. There is 3 pieces of information you need to provide for a chart to be rendered. The below example will create a "ComboChart"

(i) In the <head > region of your page (or a link to an external .js file) place the following:

```
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"> </script>
<script type="text/javascript">
    google.charts.load('current', {'packages':['corechart']});
    google.charts.setOnLoadCallback(drawVisualization);
// From this point embed the function (drawVisualization) that will render this chart of your choice
function drawVisualization() {
    // Some raw data (not necessarily accurate)
//1. Define your data …stored in the variable array "data". The first column holds the categories on which the numerical data series is based. The first row holds the names of each data column
    var data = google.visualization.arrayToDataTable([
     ['Month', 'Bolivia', 'Ecuador', 'Madagascar', 'Papua New Guinea', 'Rwanda', 'Average'],
     ['2004/05',  165,      938,       522,          998,          450,      614.6],
     ['2005/06',  135,      1120,      599,          1268,         288,      682],
     ['2006/07',  157,      1167,      587,          807,          397,      623],
     ['2007/08',  139,      1110,      615,          968,          215,      609.4],
     ['2008/09',  136,      691,       629,          1026,         366,      569.6]
    ]);
//  2. define the options for this particular chart
    var options = {
     title : 'Monthly Coffee Production by Country',
     vAxis: {title: 'Cups'},
     hAxis: {title: 'Month'},
     seriesType: 'bars',
     series: {5: {type: 'line'}}
```

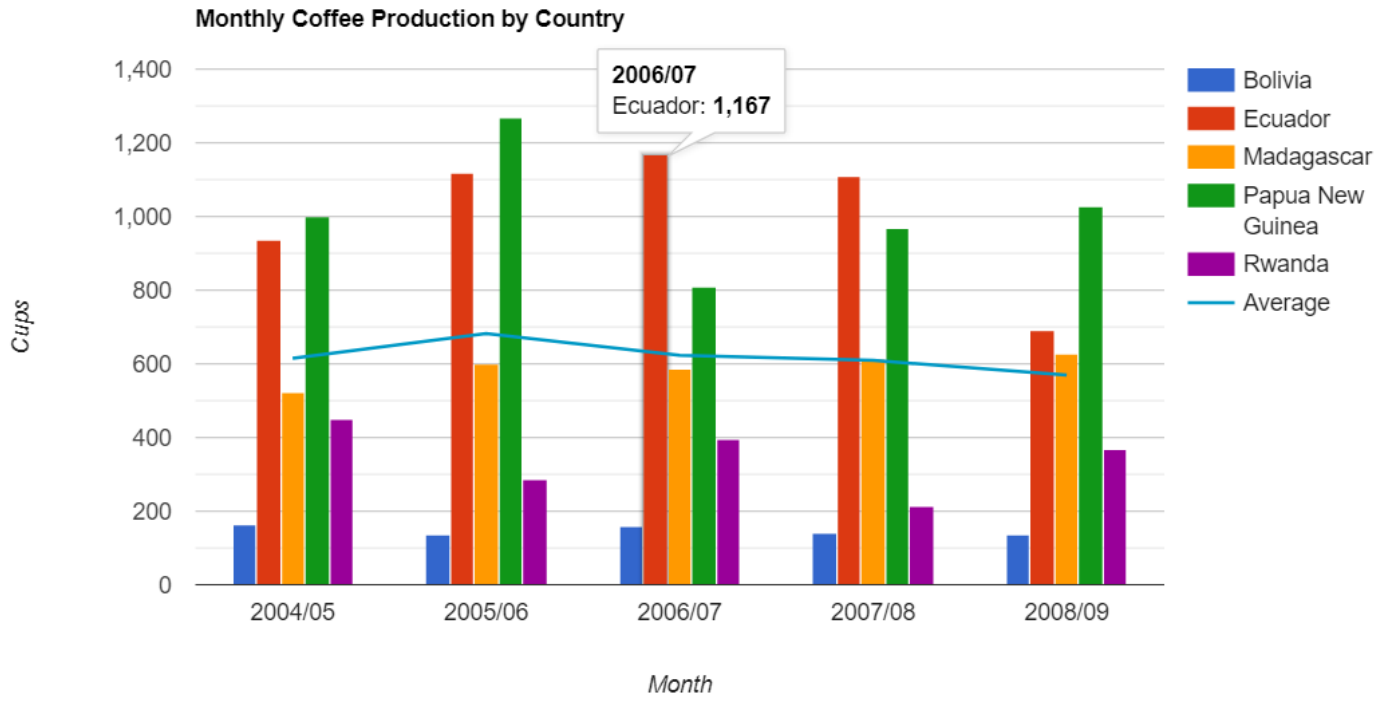
```
    };
// 3. Invoke the code to draw the chart. Be sure to supply the name of the <div> element where the chart is to be rendered.
    var chart = new google.visualization.ComboChart(document.getElementById('chart_div'));
    chart.draw(data, options);
  }
</script>
```

//4. Don't forget to terminate the script with </script>

//5. In the body of the page be sure to create a pair with the name you mentioned in #3 (e.g. chart_div)

e.g.

The above example will create a chart that looks like

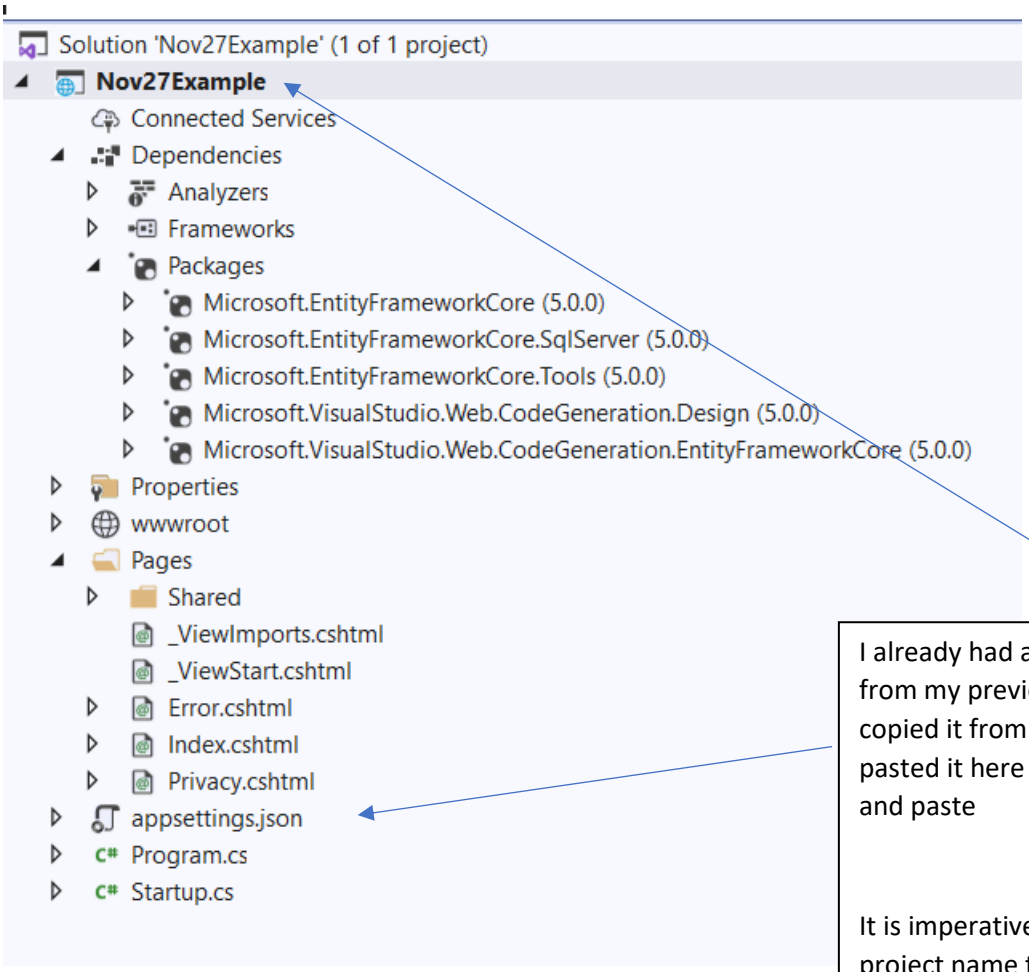


Practice doing this outside of your project on a simple html file, when you find the chart you like and data then incorporate examples of these in html documents within your project that could be reached from your layout page navigation links

For the rest of the visualization elements discussed in this document, be sure to create all your views, stored procedures (without parameters), stored procedures (with parameters).

Have all this ready along with your tables **before you scaffold your database** and create your context.cs file.

(B) GET YOUR DATA READY



I already had a functional appsettings.json from my previous project, so I selected it, copied it from the previous project and pasted it here by selecting the project name, and paste

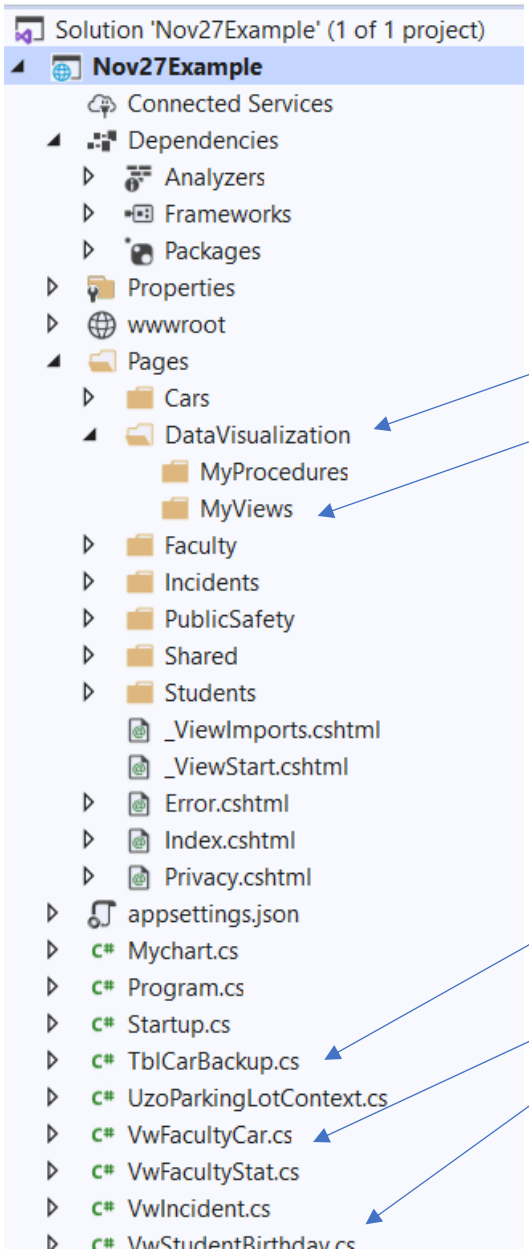
It is imperative to paste AFTER clicking on the project name to ensure that the apsetting files paste at the right level (directly in the main project folder)

After pasting your appsettings.json files, ensure you have the correct NuGet packages installed. See image above. Adjust your DatabaseContext.cs and Startup.cs file as your other projects. Th starup.cs may be identical from your previous projects BUT the databaseContext.cs file may have changed due to new objects such as views, stored procedures in your database. So make the manual changes in these as you did in prior projects.

Then connect to SQL SERVER, make your connections trusted, as before,

Then run your Scaffold command from the NuGet PM console window,

All your tables, and views will be scaffolded. You will have o create model.cs files for each stored procedure, and amend your databaseContext.cs file to accommodate these stored procedure model files.



Create new folders, one for your views and another for your stored procedures, or however you choose to navigate. Create your CRUD pages as before in the appropriate folders.
Move your view.cs files into the folder you made for your views.
I made a DataVisualization folder in which I placed a folder for views and another for procedures. This is how I choose to navigate from my main page

Remove any scaffolded file you have no use for. Which were made from tables in your database irrelevant to this project

These view vw.cs files will be moved into my MyViews folder

Your typical view.cs file looks like this

```
using System;
using System.Collections.Generic;

#nullable disable

namespace Nov27Example
{
    2 references
    public partial class VwStudentMercede
    {
        1 reference
        public string Fname { get; set; }
        1 reference
        public string Lname { get; set; }
        1 reference
        public string Make { get; set; }
    }
}
```

If you choose to rename your column names as opposed to using the table column names, this is where you specify it. You can also specify the way you want the data displayed if you have a numeric value you want to display with so many decimal places.

To do this you need to add

using System.ComponentModel.DataAnnotations;

VwStudentMercede.cs

Nov27Example

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel.DataAnnotations;
4   #nullable disable
5
6   namespace Nov27Example
7   {
        2 references
8       public partial class VwStudentMercede
9       {
10          [Display(Name = "First Name")]
            1 reference
11          public string Fname { get; set; }
12          [Display(Name = "Last Name")]
            1 reference
13          public string Lname { get; set; }
14          [Display(Name = "Car Brand")]
            1 reference
15          public string Make { get; set; }
16      }
17  }
18
```

add

using System.ComponentModel.DataAnnotations;

put [Display(Name = " your preferred column name")]

immediately above specified field

Do this for each view display view.cs file for each view you want modified.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
#nullable disable

namespace Nov13PartB
{
    4 references
    public partial class VwStudentBirthday
    {
        [Display(Name = "First Name")]
        3 references
        public string FirstName { get; set; }
        [Display(Name = "Last Name")]
        3 references
        public string LastName { get; set; }

        [DisplayFormat(DataFormatString = "{0:dd MMM yyyy}")]
        3 references
        public DateTime? Birthday { get; set; }

        public static implicit operator VwStudentBirthday(TblCar v)
        {
            throw new NotImplementedException();
        }
    }
}
```

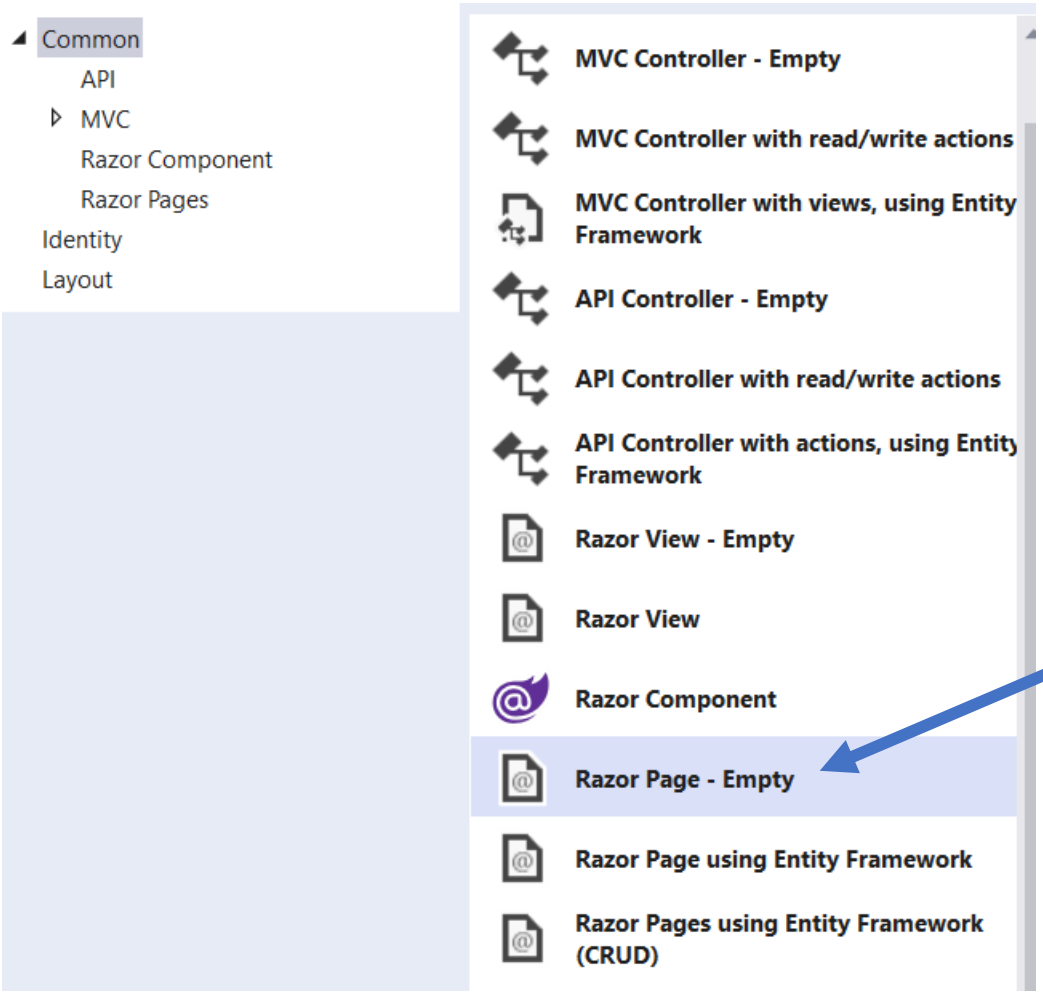
Another example of modification of a view.cs file to alter display name of a column, and also to change the way the data is displayed. Here I am controlling the date format display, as well as first name and last name displayed instead of fname and lname

(C) Viewing data in a grid (tabular structure) from a View

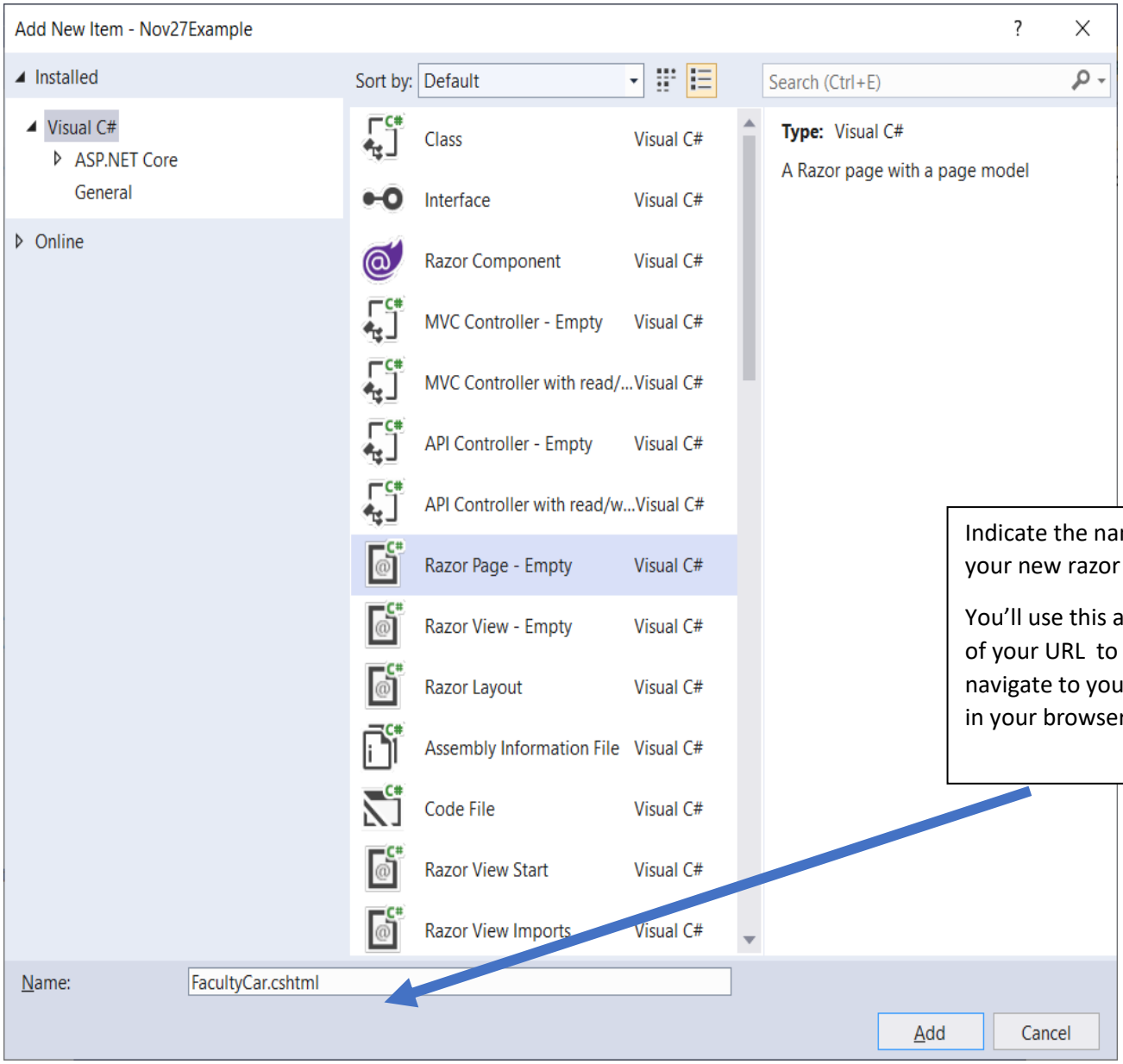
Create an empty Razor page based on your databaseContext.cs file

Click on the folder in which you want to create your .cshtml page for your view.

Add new item, and select



Be sure to select Razor PAGE **not** Razor View

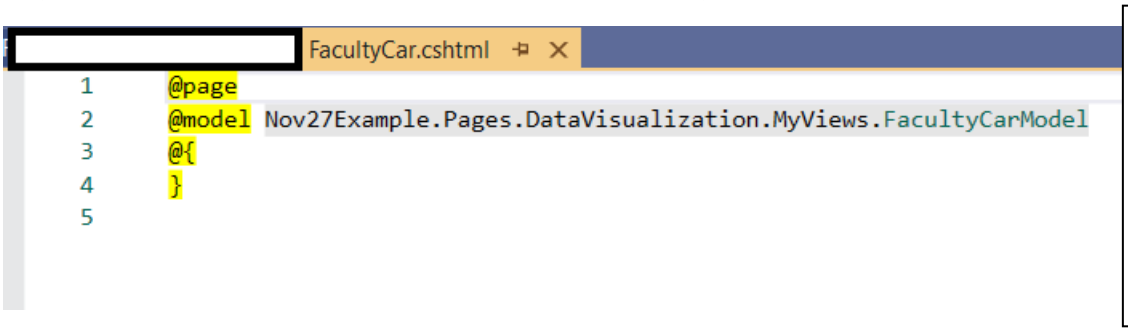


Indicate the name for your new razor page

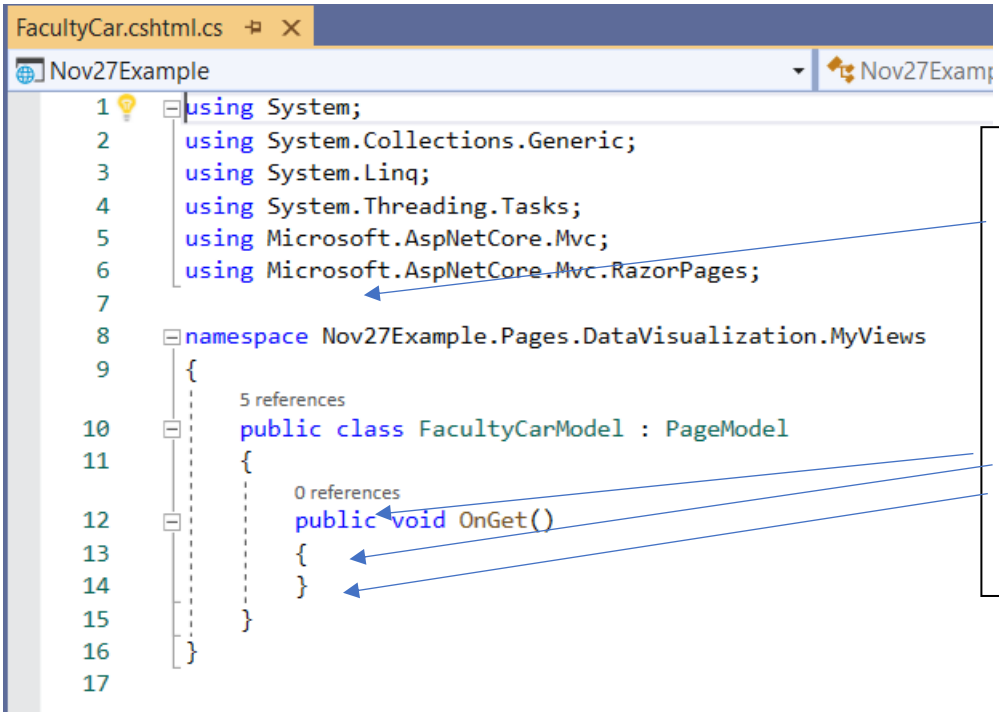
You'll use this as part of your URL to navigate to your page in your browser

Upon creation of your Empty Razor Page will have a .cshtml part and a .cs part.

E.g in the example above my page was named FacultyCar.cshtml so I will see that in my solution explorer and when I click on it I will see FacultyCar.cs file.



This is what the .cs file behind your newly created razor page. We will amend it later to control our page display



Add

using Microsoft.EntityFrameworkCore;

This is what the ..cshtml file of your newly created razor page. We will amend it next to reflect he model of your view.cs file

Eliminate these sentences including Braces, and replace them with these shown below

```
private readonly Nov27Example.UzoParkingLotContext _context;

    public FacultyCarGridModel(Nov27Example.UzoParkingLotContext context)
      {
         _context = context;
      }

      public IList<VwFacultyCar> VwFacultyCar { get; set; }

      public async Task OnGetAsync()
      {
      VwFacultyCar = await _context.VwFacultyCars.ToListAsync();
      }
```

Replace this with the name of your database and your databaseContext.cs file

Name of the Razor page with the word Model after it

Name of your VWview.cs model file. Made from the

See the next page to see screen shot of what your razor page looks like after these changes.

FacultyCar.cshtml.cs*

Nov27Example

Nov27Example.Pages.DataVisualizat

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5   using Microsoft.AspNetCore.Mvc;
6   using Microsoft.AspNetCore.Mvc.RazorPages;
7   using Microsoft.EntityFrameworkCore;
8   namespace Nov27Example.Pages.DataVisualization.MyViews
9   {
        6 references
10      public class FacultyCarModel : PageModel
11      {
12          private readonly Nov27Example.UzoParkingLotContext _context;
13
            0 references
14          public FacultyCarModel(Nov27Example.UzoParkingLotContext context)
15          {
16              _context = context;
17          }
18
            1 reference
19          public IList<VwFacultyCar> VwFacultyCar { get; set; }
20
            0 references
21          public async Task OnGetAsync()
22          {
23              VwFacultyCar = await _context.VwFacultyCars.ToListAsync();
24          }
25
26      }
27  }
28
```

Now to amend your Razor page.cshtml file which depicts the data displayed on your page and any other styling and embellishments.

I am choosing to display my data in a table which I can style later on.

Note how column names are displayed, and note how an iteration occurs for every row in the table, modeling the data collection. See diagram on next page. The field names used are the exact spelling and case specified in the view.cs model file.

FacultyCar.cshtml*

```
@page
@model Nov27Example.Pages.DataVisualization.MyViews.FacultyCarModel
@{
    ViewData["Title"] = "Faculty Cars";
    Layout = "~/Pages/Shared/_Layout.cshtml";
}

<table id="myTable" class="table-striped table-bordered" style="width:100%;">

    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.VwFacultyCar[0].FacultyName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.VwFacultyCar[0].Gender)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.VwFacultyCar[0].Car)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.VwFacultyCar[0].Color)
            </th>

        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.VwFacultyCar)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.FacultyName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Gender)

                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Car)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Color)

                </td>

            </tr>
        }
    </tbody>
</table>

```

The page when displayed looks like this:

Nov27Example Home Privacy

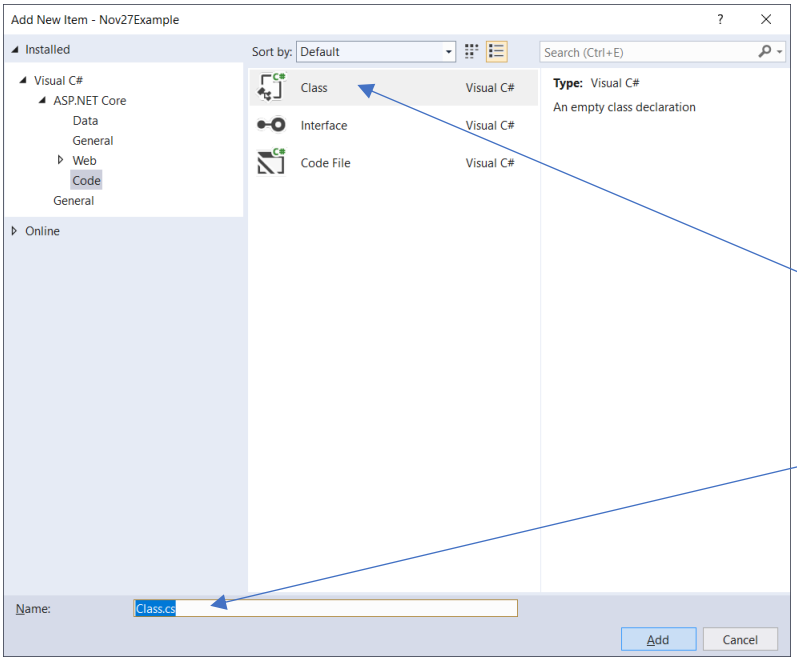
Faculty	Gender	Car	Color
Smith	M	Honda	Black
Jones	F	Honda	Red
Jones	M	Toyota	Blue
Williams	M	Toyota	White
Perez	F	BMW	Blue
Robinson	M	Mercedes	Red
Perez	M	BMW	Blue
Gardenia	M	Bugatti	Silver
Wellington	F	Bugatti	Red

(D) Viewing data from Stored Procedures (without parameters)

Unlike the Views which can be scaffolded into initial view.cs model files, we need to create our own model.cs files for each stored procedure, and include the models in our databaseContext.cs file.

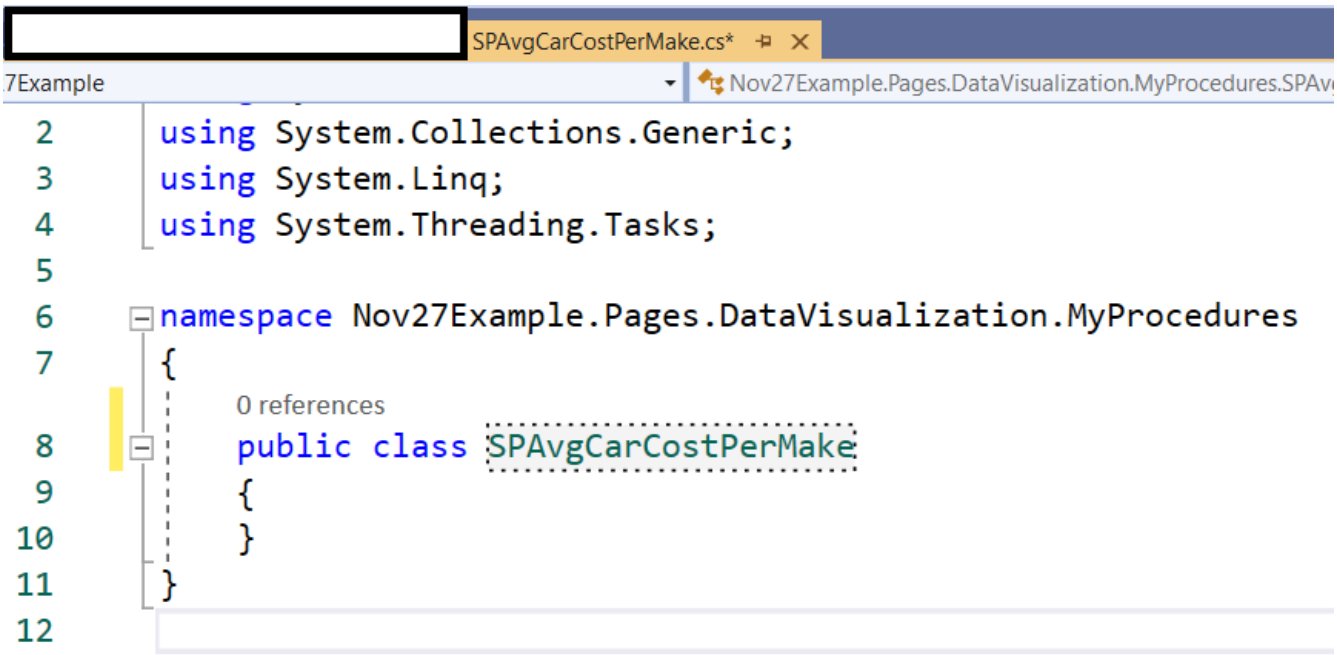
Then we can do what we did above in (C) and create an empty Razor page, modify the .cs file and display the .cshtml as we wish… in a table grid, or in a chart.

(1)Create a class declaration for a stored procedure. Select the folder where you want the razor page for this data, then Add new item, and select CLASS, when the window below pops open, put the name of your class



Write the name of your class. You may just use the name of your stored procedure

It will look like below



SPAvgCarCostPerMake.cs*

Nov27Example

Nov27Example.Pages.DataVisualization.MyProcedures.SPAvgC

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5   using System.ComponentModel.DataAnnotations;
6
7   namespace Nov27Example.Pages.DataVisualization.MyProcedures
8   {
        0 references
9       public class SPAvgCarCostPerMake
10      {
11
            0 references
12          public string Gender { get; set; }
13          [Display(Name = "Car Brand")]
            0 references
14          public string make { get; set; }
15
16          [DisplayFormat(DataFormatString = "{0:C}")]
17          [Display(Name = "Average Cost")]
            0 references
18          public string AverageCost { get; set; }
19      }
20  }
21
```

Remove.Pages.DataVisualization.MyPRocedurs

Which represent an absolute path to your class file

2) Add declarations to match the data generated by your store procedure in the order of occurrence. Keep the namespace area relative, by just using project name

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace Nov27Example
{
    2 references
    public partial class spAvgCarCostPerMake
    {

        [Display(Name = "Car Brand")]
        1 reference
        public string make { get; set; }

        [DisplayFormat(DataFormatString = "{0:C}")]
        [Display(Name = "Average Cost")]
        1 reference
        public double AverageCost { get; set; }
    }
}
```

```
namespace Nov27Example
{
    57 references
    public partial class UzoParkingLotContext : DbContext
    {

        0 references
        public UzoParkingLotContext(DbContextOptions<UzoParkingLotContext> options)
            : base(options)
        {
        }
        0 references
        public virtual DbSet<spAvgCarCostPerMake> spAvgCarCostPerMake { get; set; }
        0 references
        public virtual DbSet<Mychart> Mycharts { get; set; }
        14 references
        public virtual DbSet<TblCar> TblCars { get; set; }
```

In your DatabaseContext.cs file, add one line for each stored procedure, using the same name as your stored procedure to avoid confusion

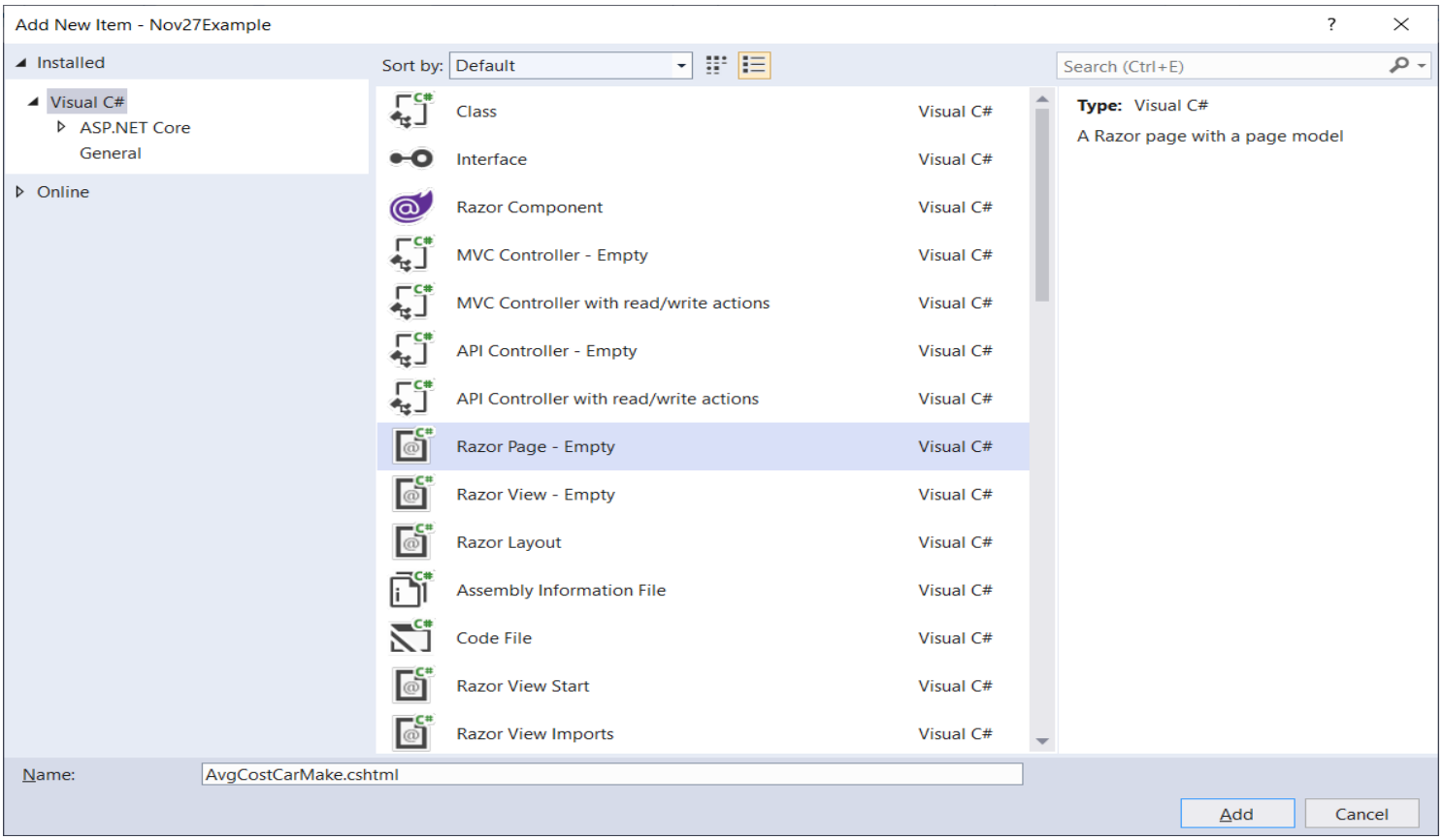
3) Adjust the DatabaseContext.cs file to account for this new class of your stored procedure

Then scroll down in your DatabaseContext.cs file to the part where the modelbuilders are defined. Insert a new modelbuilder for your stored procedure using the pattern below

Include an entity.Property for each column in your stored procedure.

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<spAvgCarCostPerMake>(entity =>
    {
        entity.HasNoKey();

        entity.ToView("spAvgCarCostPerMake");

        entity.Property(e => e.make);


        entity.Property(e => e.AverageCost).HasColumnName("AverageCost");


    });

    modelBuilder.Entity<Mychart>(entity =>
    {
        entity.HasNoKey();
```

3) Now to display data from the stored procedure on a razor.cshtml page, just like you did with the views. Click on the folder containing where the stored procedure.cs file is located, click Add new Item, and select empty Razor Page



Name the page. I usually name it with something declarative of what the stored procedure's function is. Do not use the same name as the stored procedure. (I removed the sp)
A .cshtml page will be created along with a .cs file behind it. Just like the views we will take care of the .cs file, then the display part on the .cshtml file.

4) Modifying the .cs file. It initially looks like the picture below

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5   using Microsoft.AspNetCore.Mvc;
6   using Microsoft.AspNetCore.Mvc.RazorPages;
7
8   namespace Nov27Example.Pages.DataVisualization.MyProcedures
9   {
        5 references
10      public class AvgCostCarMakeModel : PageModel
11      {
            0 references
12          public void OnGet()
13          {
14          }
15      }
16  }
```

5)Replace encircled portion above with the following: Also be sure to include at the top with other using statements

using Microsoft.EntityFrameworkCore;

```
            private  readonly  Nov27Example.UzoParkingLotContext _context;

      public AvgCarCostPerMakeModel(Nov27Example.UzoParkingLotContext context)
      {
         _context = context;
      }

      public IList<spAvgCarCostPerMake> spAvgCarCostPerMake { get; set; }

      public async Task OnGetAsync()
      {

         try
         {

spAvgCarCostPerMake = await _context.spAvgCarCostPerMake.FromSqlRaw("Exec spAvgCarCostPerMake").ToListAsync();
         } catch(Exception)
         {

         }
      }

      public async Task OnPostAsync()
      {
spAvgCarCostPerMake = await _context.spAvgCarCostPerMake.FromSqlRaw("Exec spAvgCarCostPerMake").ToListAsync();

      }
```

Your .cs file will look like the figure on the next page

AvgCarCostPerMake.cshtml.cs

Nov27Example

Nov27Example.Pages.DataVi

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5   using Microsoft.AspNetCore.Mvc;
6   using Microsoft.AspNetCore.Mvc.RazorPages;
7   using Microsoft.EntityFrameworkCore;
8
9   namespace Nov27Example.Pages.DataVisualization.MyProcedures
10  {
        - references
11      public class AvgCarCostPerMakeModel : PageModel
12      {
13          private readonly Nov27Example.UzoParkingLotContext _context;
14
            - references
15          public AvgCarCostPerMakeModel(Nov27Example.UzoParkingLotContext context)
16          {
17              _context = context;
18          }
19
            - references
20          public IList<spAvgCarCostPerMake> spAvgCarCostPerMake { get; set; }
21
            - references
22          public async Task OnGetAsync()
23          {
24
25              try
26              {
27
28                  spAvgCarCostPerMake = await _context.spAvgCarCostPerMake.FromSqlRaw("Exec spAvgCarCostPerMake").ToListAsync();
29              }
30              catch (Exception)
31              {
32
33              }
34          }
35
            - references
36          public async Task OnPostAsync()
37          {
38              spAvgCarCostPerMake = await _context.spAvgCarCostPerMake.FromSqlRaw("Exec spAvgCarCostPerMake").ToListAsync();
39
40          }
41      }
42  }
43
```

Add using EntityFrameworkCore Name ofyour cshtml file with Model after it

Add this line declaring your Database.Context file name with project preceding it indicating position in the directory structure

Instantiate context

Declare your storedprocedure model file (.cs) as a collection

Declare onGet and onPost actions
With the ability to Exec raw sql statement. Recall that in SQLserver we were able to execute our stored procedures with exec name of stored procedure

5) Now time to amend the .cshtml portion for the display. Initially it looks like this

AvgCostCarMake.cshtml

```
1   @page
2   @model Nov27Example.Pages.DataVisualization.MyProcedures.AvgCostCarMakeModel
3   @{
4   }
5
```

Include the ViewData statement which will Title the page, and include the Lyout statement indicating which page to use for your layout. The default is _Layout.cshtml

I chose to display my stored procedure results in a table. I used the first row of the Display Name for my column headings, and used a For loop to iterate through the rows to show collection content. (Just like with the Views earlier)

See next page

AvgCarCostPerMake.cshtml

```
@page
@model Nov27Example.Pages.DataVisualization.MyProcedures.AvgCarCostPerMakeModel

@{
    ViewData["Title"] = "Average Cost Per Make";
    Layout = "~/Pages/Shared/_Layout.cshtml";
}

<table id="myTable" class="table-striped table-bordered" style="width:100%;">


    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.spAvgCarCostPerMake[0].make)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.spAvgCarCostPerMake[0].AverageCost)
            </th>


        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.spAvgCarCostPerMake)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.make)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.AverageCost)

                </td>

            </tr>
        }
    </tbody>
</table>
```

(E) Viewing Data from Stored Procedures (**with parameters**)

In the case of Stored Procedures with parameters, everything is the same with (D) except in the .cs file of the top part of the .cshtml used to view it.

i.e. make your stored procedure model file as before, nameofstoredprocedure.cs from an empty Razor .cs page, include a new model builder in your DatabaseContextfile.cs for this new stored procedure.cs file, then create a Razor page in with which you will display the contents as above. At the beginning of this dislay.cshtm for your stored procedure with parameters, you must include a way to receive these parameter arguments. I use a form

```
<form class="form-horizontal" method="post" id="moi">
  <div class="form-group">
    <label for="myMake" class="col-sm-2 control-label">Brand desired:</label>
    <div class="col-sm-10">
      <input type="text" id="myMake" class="form-control" value="" name="myMake">
    </div>
  </div>
  <div class="form-group">
    <label for="myColor" class="col-sm-2 control-label">Cost:</label>
    <div class="col-sm-10">
      <input type="text" id="myColor" class="form-control" value="" name="myColor">
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="submit" class="btn btn-primary">Get Car Data</button>
    </div>
  </div>
</form>
```

Use the same id value for your parameters as you would use in the underlying .cs page

Mine are myMake and myColor

arColorChoice.cshtml.cs | CarColorChoice.cshtml*

```
@page
@model Nov27Example.Pages.DataVisualization.MyProcedures.CarColorChoiceModel
@{
    ViewData["Title"] = "Car Make and Color per Choice";
    Layout = "~/Pages/Shared/_Layout.cshtml";
}

<form class="form-horizontal" method="post" id="moi">
    <div class="form-group">
        <label for="myMake" class="col-sm-2 control-label">Brand desired:</label>
        <div class="col-sm-10">
            <input type="text" id="myMake" class="form-control" value="" name="myMake">
        </div>
    </div>
    <div class="form-group">
        <label for="myColor" class="col-sm-2 control-label">Color:</label>
        <div class="col-sm-10">
            <input type="text" id="myColor" class="form-control" value=0 name="myColor">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <button type="submit" class="btn btn-primary">Get Car Data</button>
        </div>
    </div>
</form>

<table id="myTable" class="table-striped table-bordered" style="width:100%;">


    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.carchoice[0].make)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.carchoice[0].color)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.carchoice[0].caryear)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.carchoice[0].fname)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.carchoice[0].lname)
            </th>

        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.carchoice)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.make)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.color)

                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.caryear)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.fname)

                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.lname)
                </td>

            </tr>
        }
    </tbody>
</table>
```

My .CSHTML file looks like this

I included a form, with which to retrieve the parameters for the stored procedure, when the user clicks the button then the data is displayed.

On the next page I will display my .cs file which supports this .cshtml display

After checking that the data displays fine, I will later come back to my .cshtml file and add a div below the table, where I will serialize the data for my charts. I will hide this display when I am sure the data is good

Next Section we will discuss charts derived from the .cshtml pages.

My .cs file is on the next page

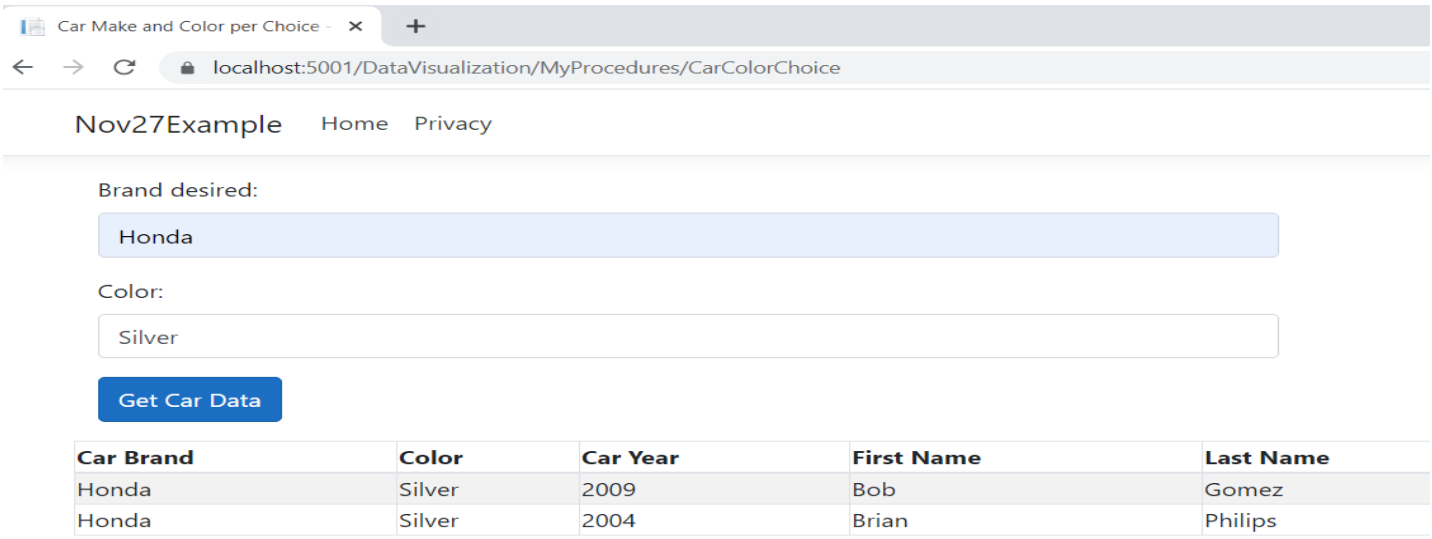
CarColorChoice.cshtml.cs

Nov27Example

Nov27Example.Pages.DataVisualizat

```
{
    6 references
    public class CarColorChoiceModel : PageModel
    {
        private readonly Nov27Example.UzoParkingLotContext _context;

        0 references
        public CarColorChoiceModel(Nov27Example.UzoParkingLotContext context)
        {
            _context = context;
        }

        8 references
        public IList<carchoice> carchoice { get; set; }  //THIS MAKES IT STATIC AND ONLY READABLE

        0 references
        public async Task OnGetAsync()
        {

            try
            {
                var carmakeSQLParam = new Microsoft.Data.SqlClient.SqlParameter("@make", "");
                var carcolorSQLParam = new Microsoft.Data.SqlClient.SqlParameter("@color", 1);
                carchoice = await _context.carchoice.FromSqlRaw("Exec carchoice @make={0}, @color={1}", carmakeSQLParam, carcolorSQLParam).ToListAsync();
            }
            catch (Exception)
            {

            }
        }

        0 references
        public async Task OnPostAsync()
        {
            try
            {

                string myMake = HttpContext.Request.Form["myMake"];
                string myColor = HttpContext.Request.Form["myColor"];

                var carmakeSQLParam = new Microsoft.Data.SqlClient.SqlParameter("@make", myMake);
                var carcolorSQLParam = new Microsoft.Data.SqlClient.SqlParameter("@color", myColor);
                carchoice = await _context.carchoice.FromSqlRaw("Exec carchoice @make={0}, @color={1}", carmakeSQLParam, carcolorSQLParam).ToListAsync();
            }
            catch (Exception)
            {

            }


        }
    }
}
```

My next display shows what happens after I enter my arguments of 'Honda' and 'Silver'



(F) Making Charts from your .cshtml Razor pages

First of al I would only be charting data that has a definite category and one or more numeric data series.
So a good example of where I might do this would be on my stored procedure which features average cost per Car brand. That gives me a column for Brand and one for Avg Cost

As demonstrated from the google charts demonstrated on the first couple of pages of this document,
Your chart information must be in three parts
-invoke the google chart engine
-specify data
-specify options
-draw chart on page (within a div)

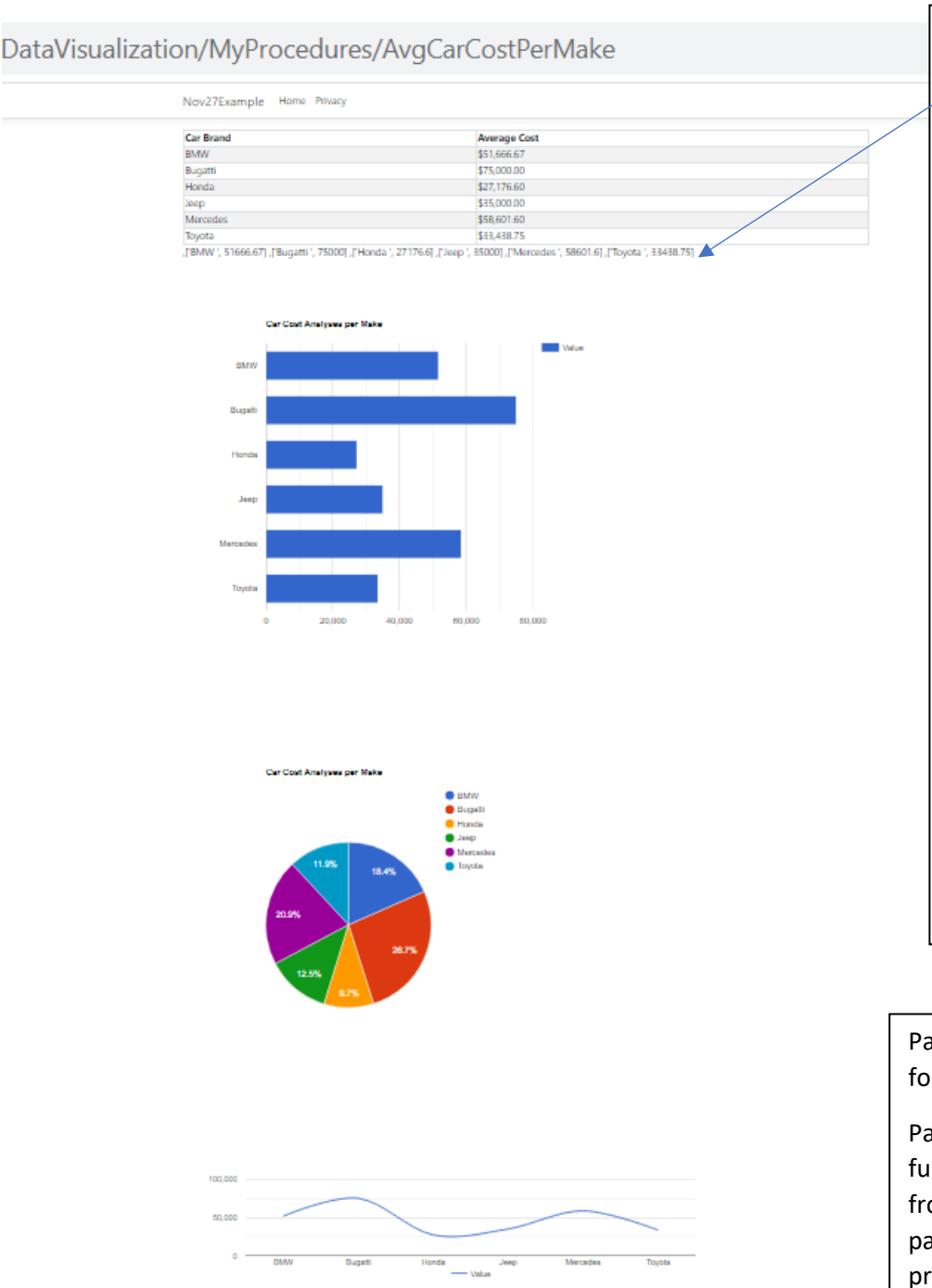
AvgCarCostPerMake.cshtml

```
@page
@model Nov27Example.Pages.DataVisualization.MyProcedures.AvgCarCostPerMakeModel

@{
    ViewData["Title"] = "Average Cost Per Make";
    Layout = "~/Pages/Shared/_Layout.cshtml";
    <script src="https://www.gstatic.com/charts/loader.js"></script>
    // for mychart.html</script>
    <script src="~/js/mychart.js"></script>
    <script>google.charts.setOnLoadCallback(myPieChartCostSumCount);</script>
}

<table id="myTable" class="table-striped table-bordered" style="width:100%;">


    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.spAvgCarCostPerMake[0].make)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.spAvgCarCostPerMake[0].AverageCost)
            </th>


        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.spAvgCarCostPerMake)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.make)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.AverageCost)
                </td>
            </tr>
        }
    </tbody>
</table>


<div id="mydata" height="400" style="display:block">

    @foreach (var item in Model.spAvgCarCostPerMake)
    {
        @string.Concat(" ,['", @item.make, " ', ", Math.Round(@item.AverageCost, 2), "]")


    }


</div>

<div id="myBarChart"></div>
<br />
<div id="myPieChart"></div>
<br />
<div id="myLineChart"></div>
```

I reference the google chart engine.
I reference the javascript file with my script for manipulating my data array and setting chart options
You must also reference the function you want invoked to generate your file. You will see examples of these in pages 22-25.
The data is usually expected as an array of arrays, so I use the display name, and model items to build up this array.

I used the same data to create three different charts

The created charts are on the next page.



I purposely let the raw data so you could see how the generated data looks. I normally would set the style attribute to display:none, so that it remains invisible.

My id is mydata, and you will see how this is referenced in my javascript file mychart.js

(PAGES 22 AND 23)

Which contains all my functions for each chart.

The function being used here for all three charts is myPieChartCostSumCount()

Which grabs the data from the mydata, does some apostrophe replacements, concatenates the data into the form expected (array of arrays) then draws the same data three times specifying different options for each one.

Page 22 Shows my javascript file for generic charts

Page 23 shows my javascript function that generates chart from data from my razor .cshtml page obtained from the stored procedure

Be sure to include you .js files in your layout or at the top of relevant page. See page 20

This is mycharts.js

I'll point out what you need

Load google core chart packages

ALWAYS FIRST LINE

```
// Load packages ONCE as seen  below

   google.charts.load('current', {packages: ['corechart'] });


      function myDrawPieChart() {
         // Define the chart to be drawn.
         var data = new google.visualization.DataTable();
         data.addColumn('string', 'Element');
         data.addColumn('number', 'Percentage');
         data.addRows([
            ['Nitrogen', 0.78],
            ['Oxygen', 0.21],
            ['Other', 0.01]
         ]);


// options to usemap
      var options_pie = {title:'Pie Chart: Air Constitution',
                width:500, height:500};


         // Instantiate and draw the chart.
         var chart = new google.visualization.PieChart(document.getElementById('myPieChart'));
         chart.draw(data, options_pie);
          }

// for multiplecharts2.html   different chart styles, same data


function drawMultipleCharts() {

   var data = new google.visualization.DataTable();
   data.addColumn('string', 'Topping');
   data.addColumn('number', 'Slices');
   data.addRows([
      ['Mushrooms', 3],
      ['Onions', 1],
      ['Olives', 1],
      ['Zucchini', 1],
      ['Pepperoni', 2]
   ]);

   var piechart_options = {
      title: 'Pie Chart: How Much Pizza I Ate Last Night',
      width: 400,
      height: 300
   };
   var piechart = new google.visualization.PieChart(document.getElementById('piechart_div'));
   piechart.draw(data, piechart_options);     // draw pie chart

   var barchart_options = {
      title: 'Barchart: How Much Pizza I Ate Last Night',
      width: 400,
      height: 300,
      legend: 'none'
   };
   var barchart = new google.visualization.BarChart(document.getElementById('barchart_div'));
   barchart.draw(data, barchart_options);     // draw bar chart
}
```

Example of a typical Chart

The function first defines data

Then provides options

Then draws chart based on type specified and in the Identified by the document.getElementById(" ")

Example of multiple chart types drawn with the same data

The function first defines data

Then provides options

Then draws chart based on type specified and in the Identified by the document.getElementById(" ")

//TO CREATE DATA FROM STORED PROCEDURE

```
//Draw chart from stored procedure
function myPieChartCostSumCount() {
   // Define the chart to be drawn.
   var data = new google.visualization.DataTable();
   data.addColumn('string', 'Category');
   data.addColumn('number', 'Value');
   var x = document.getElementById("mydata").innerHTML;
   x = x.trim();
   if (x.charAt(0) == "," )
      {
      x =  x.substring(1) ;
       }
   //**************
   // alert(x);
   var z = x.replace(/"/g, "!");  // switch " to ' by first replacing "  with !
    x = z.replace(/'/g, "~");    // switch ' to " by first replacing with ~

   z = x.replace(/~/g, "'");      // then swith ~ to "   and ! to '
   x = z.replace(/!/g, "'");

   // alert(x);
   var myChartData = JSON.parse("[" + x + "]");

   data.addRows(myChartData);     // add parsed data (myChartData) to array called data



   // options for Pie chart
   var options_pie = {
      title: 'Car Cost Analyses per Make',
      width: 700, height: 700
   };


   // options for Bar chart
   var options_bar = {
      title: 'Car Cost Analyses per Make',
      width: 700, height: 700
   };


   // options for Line chart
   var options_line = {
      curveType: 'function',
      legend: { position: 'bottom' }
   };
   // Instantiate and draw the Line chart.
   var chart = new google.visualization.LineChart(document.getElementById('myLineChart'));
   chart.draw(data, options_line);

   // Instantiate and draw the Bar chart.
   var chartBar = new google.visualization.BarChart(document.getElementById('myBarChart'));
   chartBar.draw(data, options_bar);

   // Instantiate and draw Pie chart.
   var chartPie = new google.visualization.PieChart(document.getElementById('myPieChart'));
   chartPie.draw(data, options_pie);
}
```

Example of chart drawn from data retrieved from my .cshtml Razor Page.

I named a mydata, before starting.

The function first defines data, then retrieves its contents from my

I replace the apostrophes to “” and arrange the data as an array of arrays.

Then provides options

I use the same data for multiple chart types so I give each one its own set of options.

Then draws chart based on type specified and in the Identified by the document.getElementById(“ “)