## Writing PowerShell Scripts for Power BI

As you work through these lab exercises, you will gain experience writing PowerShell scripts to automate common tasks in a Power BI environment. You will begin by ensuring your Windows PC is configured for PowerShell script development and by installing the PowerShell library for Power BI named **MicrosoftPowerBIMgmt**. After that, you will write a few simple PowerShell scripts that connect to your Power BI test environment and execute commands to create workspaces, manage workspace users and import PBIX files. In later exercises, you will be required to write more advanced PowerShell code which calls the **Invoke-PowerBIRestMethod** cmdlet to perform other essential Power BI operations such as patching datasource credentials and updating dataset parameters.

You can complete these lab exercises using either Windows PowerShell 5 or PowerShell 7 (aka PowerShell Core). The lab instructions and screenshot in this document are based on Windows PowerShell 5 and writing and testing PowerShell scripts using the Windows PowerShell Integrated Scripting Environment (ISE). However, you should be able to complete any of these lab exercises using [PowerShell 7](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-core-on-windows?view=powershell-7) and [Visual Studio Code](https://code.visualstudio.com/download) with the [PowerShell extension for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=ms-vscode.PowerShell) provided by Microsoft.

In order to complete these lab exercises, you need a Power BI Pro license or Pro trial license in a Power BI test environment in which you have permissions to create new workspaces and upload content such as a PBIX file created with Power BI Desktop. The final exercises at the end of this lab will also require that you have Power BI administrator permissions so that you can run PowerShell commands for Power BI scoped to the organization level. If you want to create a trial Office 365 tenant to provide a Power BI development environment in which you have permissions as a Power BI administrator and a global tenant administrator, you can use the step-by-step instructions in [Create a Trial Environment for Power BI Development](https://github.com/PowerBiDevCamp/Camp-Sessions/raw/master/Create%20a%20Trial%20Environment%20for%20Power%20BI%20Development.pdf).

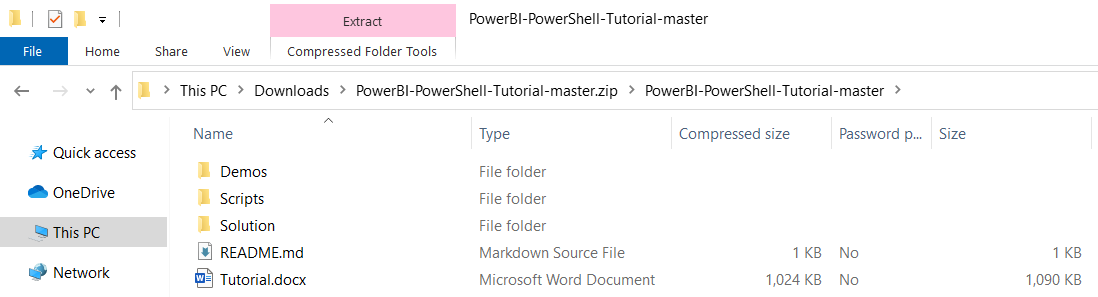
### Exercise 1: Configure PowerShell to Run Scripts on Your Computer

In this exercise, you will download the student files for this lab. You will also write and test a few simple PowerShell scripts to ensure your Windows PC is properly set up for writing and testing PowerShell scripts.

1. Download the student lab files to a local folder on your developer workstation.
   1. Create a new top-level folder on your workstation named **DevCamp** at a location such as **c:\DevCamp**.
   2. Download the ZIP archive with the student lab files from GitHub by clicking the following link.

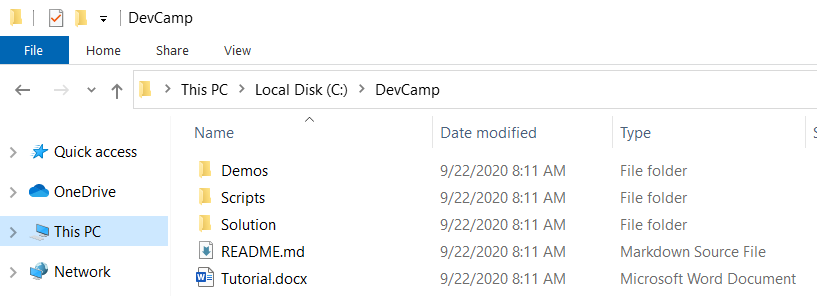
<https://github.com/PowerBiDevCamp/PowerBI-PowerShell-Tutorial/archive/master.zip>

* 1. Open the ZIP archive and locate the files inside the folder named **PowerBI-PowerShell-Tutorial-master**.

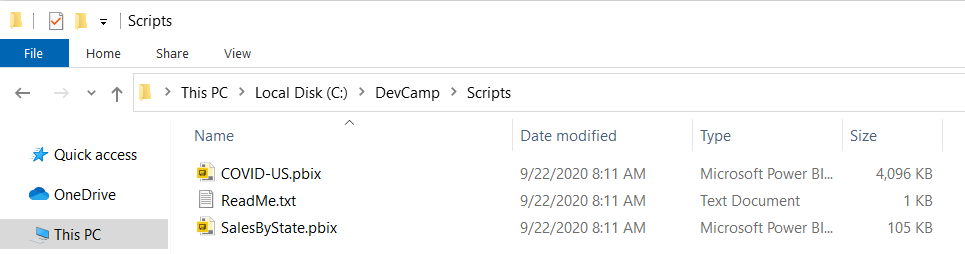


You will now copy the files out of this ZIP archive and paste them into a new folder on your local PC.

* 1. Copy the files from inside the **PowerBI-PowerShell-Tutorial-master** folder into a local folder at **C:\Devcamp**.
  2. The **C:\DevCamp** folder on your PC should now match the following screenshot.

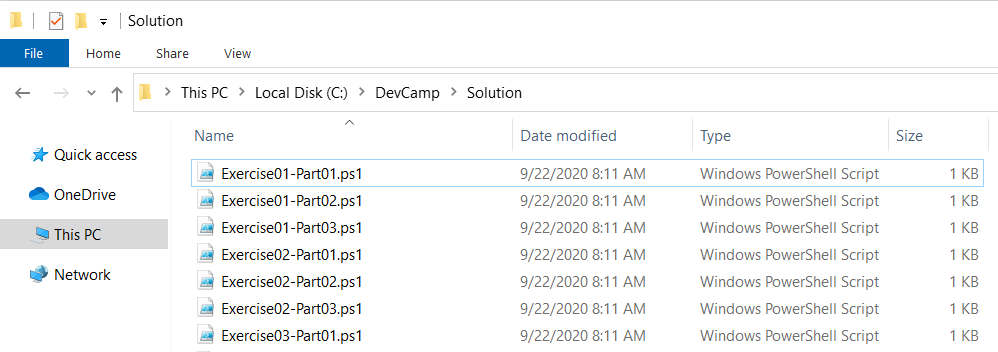


* 1. Look to see what is inside the **Scripts** folder.



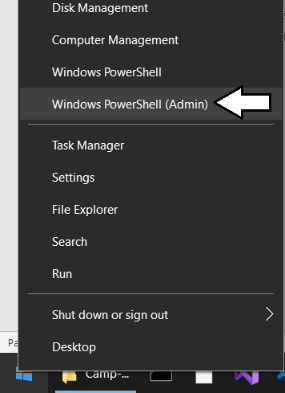
As you create new PowerShell scripts in the exercises of this lab, you will be instructed to create them in the **Scripts** folder.

* 1. Look to see what is inside the **Solution** folder.



As you can see, the **Solution** folder contains PowerShell scripts which provide solutions to all the exercises in this lab. Feel free to look at these files if you get stuck during any of the exercises.

1. Enable the execution of PowerShell scripts on your local PC if you have not already done so.
   1. Open a PowerShell command shell *running as Admin*.



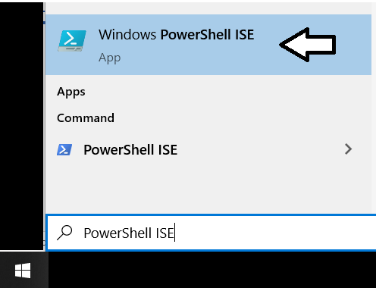
* 1. Type in and execute the following PowerShell command.

Set-ExecutionPolicy Bypass -Scope CurrentUser

* 1. When prompted to confirm to the operation, type Y and press ENTER to confirm that you want to enable script execution.

Now your PC should be configured for being able to write and testing PowerShell scripts using the Windows PowerShell ISE.

1. Create a new PowerShell script named **Exercise01.ps1**.
   1. Launch the Windows PowerShell ISE



* 1. Create a new PowerShell script and save it as **Exercise01.ps1** using the following path.

C:\DevCamp\Scripts\Exercise01.ps1

Before getting started with PowerShell for Power BI, you are going to warm up by writing and testing a few simple PowerShell scripts.

* 1. Add the following PowerShell code to **Exercise01.ps1** to create an array of strings and enumerate through it.

Clear-Host

$hobbies = @("Pilates", "Kick boxing", "Power BI Embedding")

Write-Host

Write-Host "My Hobbies"

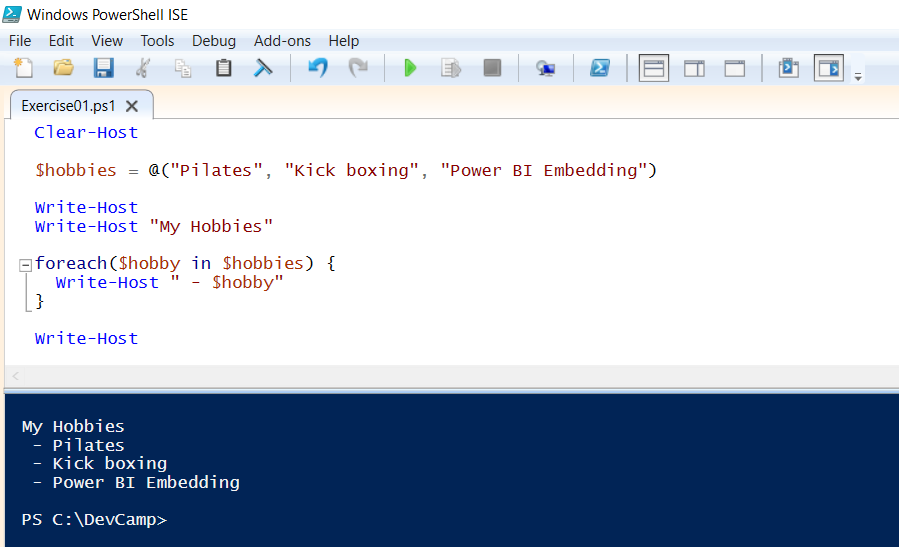
foreach($hobby in $hobbies) {

Write-Host " - $hobby"

}

Write-Host

* 1. Test the script by executing it by pressing the **{F5}** key or by pressing the **Execute** button with the green arrow.
  2. As the script executes, you should see it outputs the strings from the array in the console window.



* 1. Delete all the code in **Exercise01.ps1** except for the first line which calls **Clear-Host**.
  2. Add the following PowerShell code which creates an array of dictionaries where each dictionary contains data for a pet.

$pets = @(

@{ Name="Bob"; Type="Cat" }

@{ Name="Diggity"; Type="Dog" }

@{ Name="Larry"; Type="Lizard" }

@{ Name="Penny"; Type="Porcupine" }

)

* 1. Move below in in **Exercise01.ps1** and add the following code to output the heading **My Pets**.

Write-Host

Write-Host "My Pets"

* 1. Move below in in **Exercise01.ps1** and add the following code to enumerate the array and output information on each pet.

foreach($pet in $pets) {

$name = $pet.Name

$type = $pet.Type

Write-Host " - $name the $type"

}

* 1. Add one more call to Write-Host at the bottom of **Exercise01.ps1**.
  2. Your script should now match the following code listing.

Clear-Host

$pets = @(

@{ Name="Bob"; Type="Cat" }

@{ Name="Diggity"; Type="Dog" }

@{ Name="Larry"; Type="Lizard" }

@{ Name="Penny"; Type="Porcupine" }

)

Write-Host

Write-Host "My Pets"

foreach($pet in $pets) {

$name = $pet.Name

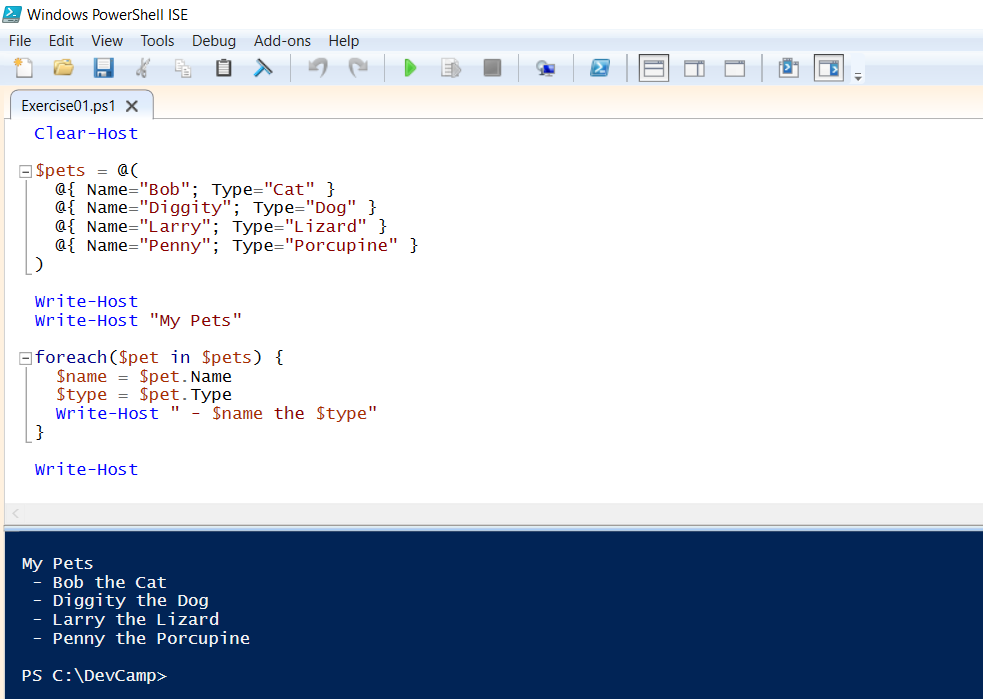
$type = $pet.Type

Write-Host " - $name the $type"

}

Write-Host

* 1. Press **{F5}** to execute the script. You should see output in the console that matches the following screenshot.



In the final step of this exercise, you will modify the PowerShell code to create a text file which contains information about the pets.

1. Write the information about pets to an output text file.
   1. Delete all the code in **Exercise01.ps1**.
   2. Add the following line to create a file path in the same folder as the script named **Pets.txt**.

$outputFilePath = "$PSScriptRoot/Pets.txt"

**$PSScriptRoot** is a variable built into PowerShell which returns the path to the folder which contains the hosting script. Therefore, the variable named **$outputFilePath** will hold a path to a file named **Pets.txt** in the same folder as the **Exercise01.ps1** script

* 1. Add the following code to create an array of dictionary objects for a collection of pets.

$pets = @(

@{ Name="Bob"; Type="Cat" }

@{ Name="Diggity"; Type="Dog" }

@{ Name="Larry"; Type="Lizard" }

@{ Name="Penny"; Type="Porcupine" }

)

* 1. Add the following line of code to write a heading of **My Pets** into the output file.

"My Pets" | Out-File $outputFilePath

* 1. Create a foreach loop to enumerate the dictionary objects and to output a line of text for each pet with its name and type.

foreach($pet in $pets) {

$name = $pet.Name

$type = $pet.Type

" - $name the $type" | Out-File $outputFilePath -Append

}

* 1. Add one more line of PowerShell code to open up the text file in notepad.

notepad.exe $outputFilePath

* 1. Your script should now match the following code listing.

$outputFilePath = "$PSScriptRoot/Pets.txt"

$pets = @(

@{ Name="Bob"; Type="Cat" }

@{ Name="Diggity"; Type="Dog" }

@{ Name="Larry"; Type="Lizard" }

@{ Name="Penny"; Type="Porcupine" }

)

"My Pets" | Out-File $outputFilePath

foreach($pet in $pets) {

$name = $pet.Name

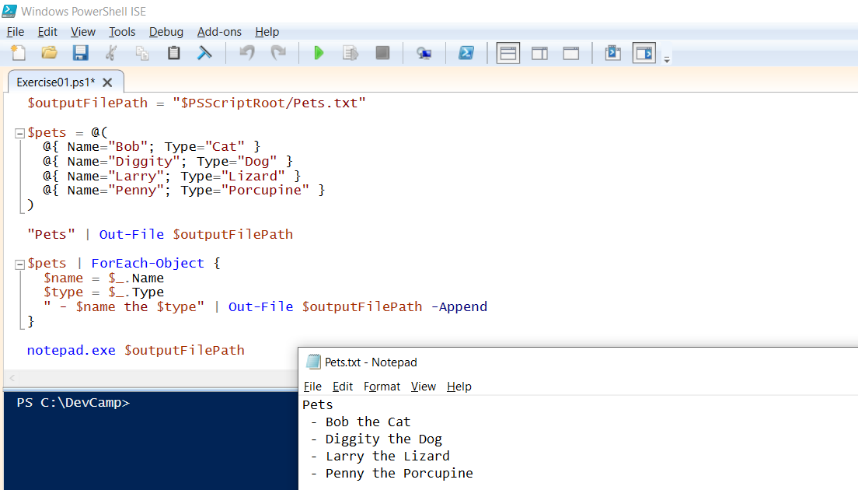
$type = $pet.Type

" - $name the $type" | Out-File $outputFilePath -Append

}

notepad.exe $outputFilePath

* 1. Test the script by executing it by pressing the **{F5}** key or by pressing the **Execute** button with the green arrow.
  2. As the script executes, you should see it generates and opens a next text file named Pets.txt with information about the pets..

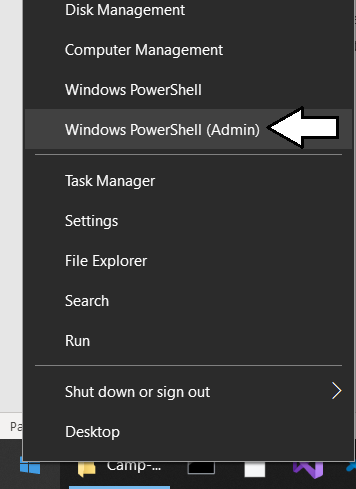


OK, now you have completed your warm up calisthenics. Now it's time to move on to writing PowerShell code for Power BI.

### Exercise 2: Install the Microsoft Power BI Cmdlets for Windows PowerShell

In this exercise, you will begin by install the PowerShell module named **MicrosoftPowerBIMgmt**.so you can access to the PowerShell cmdlets provided by the Power BI team. After that, you will write the PowerShell code to connect to you Power BI environment.

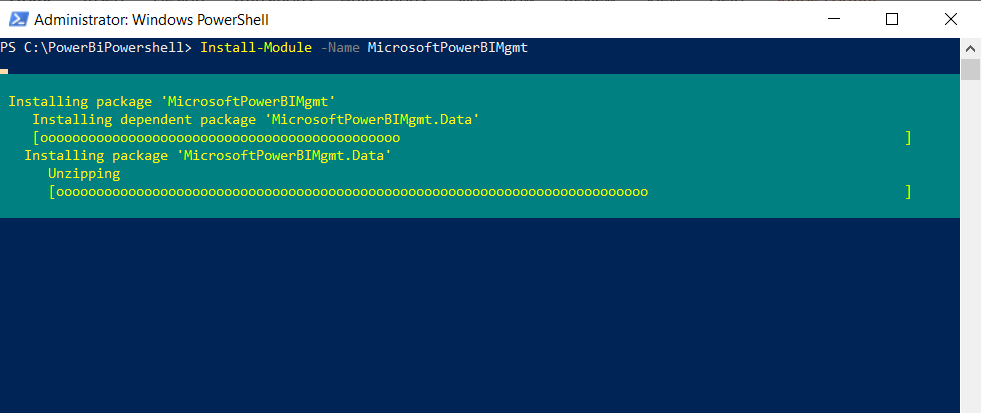
1. Install the PowerShell module named **MicrosoftPowerBIMgmt**. If you already installed **MicrosoftPowerBIMgmt** move to step 2.
   1. Right-click on the Windows Start menu and open a Windows PowerShell console as admin.



* 1. Type and execute the following code to install the PowerShell module named **MicrosoftPowerBIMgmt**.

Install-Module -Name MicrosoftPowerBIMgmt

* 1. Wait until the installation of **MicrosoftPowerBIMgmt** is complete.



Once you have installed the **MicrosoftPowerBIMgmt** module, there is no more need to use a Administrative PowerShell session. You can now return to the PowerShell ISE and use a standard PowerShell session.

1. Create a new PowerShell script named **Exercise02.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise02.ps1** using the following path.

C:\DevCamp\Scripts\Exercise02.ps1

1. Use the **Connect-PowerBIServiceAccount** cmdlet to connect to the Power BI Service.
   1. Copy and paste the following PowerShell code into **Exercise02.ps1**.

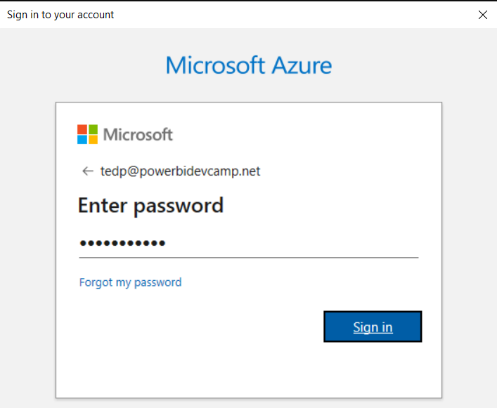
$user = Connect-PowerBIServiceAccount

$userName = $user.UserName

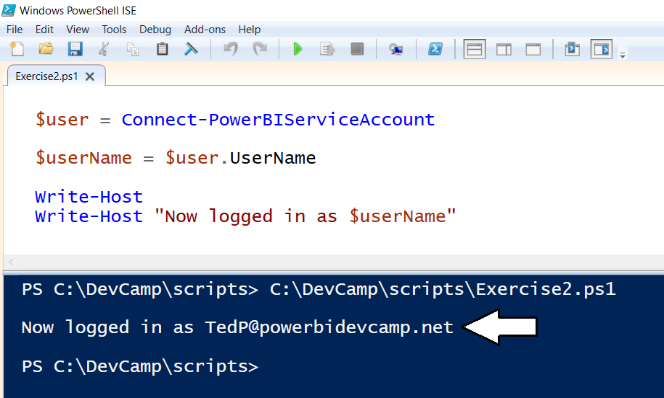
Write-Host

Write-Host "Now logged in as $userName"

* 1. Save your changes to **Exercise02.ps1**.
  2. Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
  3. When the script executes, you should be prompted to sign in.
  4. Sign in to your Power BI test environment using your user name and password.



* 1. After **Connect-PowerBIServiceAccount** executes, you script should display your user account in the console window.



As you can see, you can write scripts that does not contain any user names or passwords. This type of script can be run by a user interactively where the user is required to supply a user name and password interactively when the script begins to execute. In some scenarios such as PowerShell script development, it can convenient to hard-code the user name and password into the script so that it runs without any need for user interaction. In the next step you will modify the script with a hard-code user name and password.

1. Update **Exercise02.ps1** to log in without requiring interaction on the part of the user.
   1. Delete all the code in **Exercise02.ps1**.
   2. Copy and paste the following PowerShell code and paste it into **Exercise02.ps1** and update the variables named **$userName** and **$password** with your user credentials.

# update $userName and $password with your user credentials

$userName = "user1@tenant1.onMicrosoft.com"

$password = "myCat$rightLeg"

# convert password to secure string

$securePassword = ConvertTo-SecureString -String $password -AsPlainText -Force

# create PSCredential object to serve as login credentials

$credential = New-Object -TypeName System.Management.Automation.PSCredential `

-ArgumentList $userName, $securePassword

# log into Power BI unattended without any user interaction

$user = Connect-PowerBIServiceAccount -Credential $credential

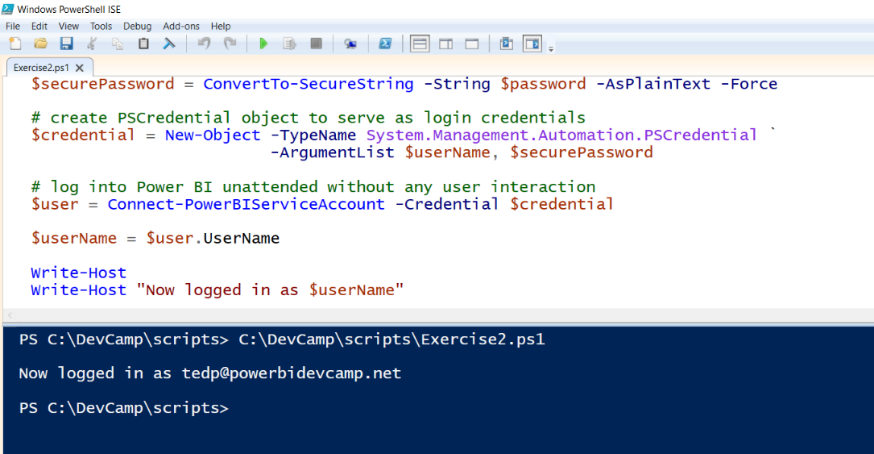
$userName = $user.UserName

Write-Host

Write-Host "Now logged in as $userName"

This script demonstrates a common technique of creating a **PSCredential** object using a secure string to include the password.

* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
  2. The script should now execute successfully without requiring you to sign in interactively.



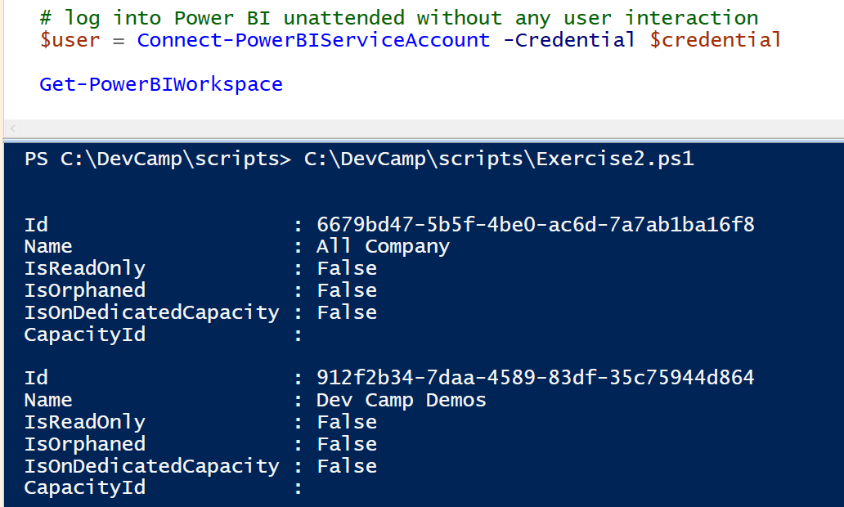
1. Add a call to **Get-PowerBIWorkspace**.
   1. Delete the lines of PowerShell code that appear after the call to **Connect-PowerBIServiceAccount**.
   2. Add a call to **Get-PowerBIWorkspace**.

# log into Power BI unattended without any user interaction

$user = Connect-PowerBIServiceAccount -Credential $credential

Get-PowerBIWorkspace

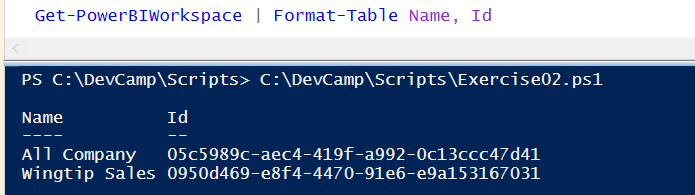
* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
  2. The script should display output for each Power BI workspace that your user account has permissions to view..



* 1. Reformat the output of **Get-PowerBIWorkspace** using the **Format-Table** cmdlet.

Get-PowerBIWorkspace | Format-Table Name, Id

* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
  2. The script should display the Power BI workspaces that your user account has permissions to view in a table format.



### Exercise 3: Write a Script to Create Workspaces and Add Workspace Users

In this exercise, you will write a PowerShell script to create a new app workspace and to add workspace users.

1. Create a new PowerShell script named **Exercise03.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise03.ps1** using the following path.

C:\DevCamp\Scripts\Exercise03.ps1

* 1. Begin by copying-and-pasting the following PowerShell code as the starting point for **Exercise03.ps1**.

Write-Host

Connect-PowerBIServiceAccount | Out-Null

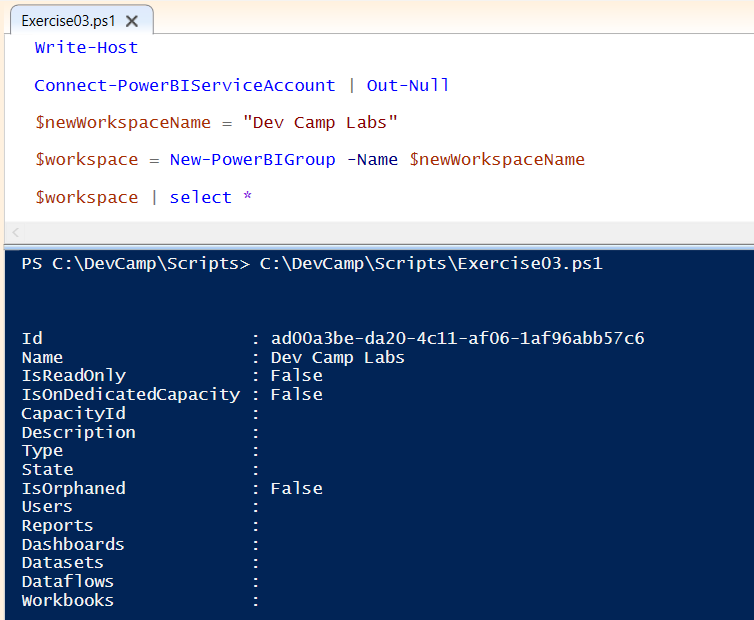
$newWorkspaceName = "Dev Camp Labs"

$workspace = New-PowerBIGroup -Name $newWorkspaceName

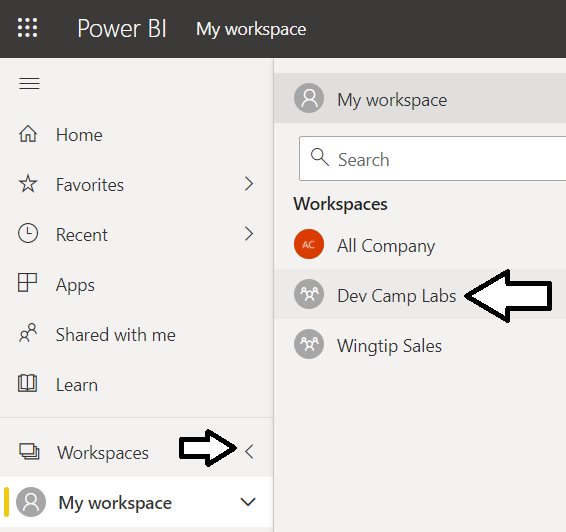
$workspace | select \*

From this point on, all the scripts you will write will connect to Power BI using a call to **Connect-PowerBIServiceAccount** which will require you to login interactively. If you want the convenience of being able to run and test your scripts without having to interactively supply a user name and password each time, you can copy and paste the code at the top of **Exercise02.ps1**.

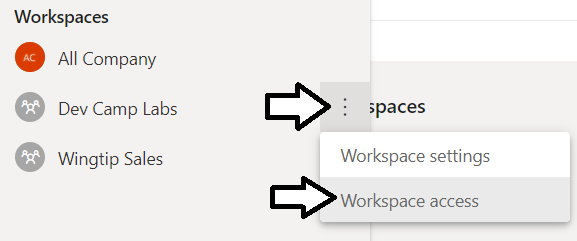
* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise03.ps1**.
  2. The script should create a new V2 app workspace display and display its properties in the PowerShell console window.



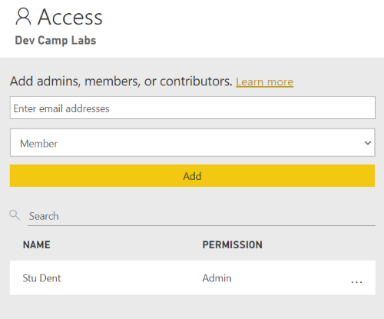
* 1. Navigate to the Power BI Service in the browser and verify that you can see the new workspace named **Dev Camp Labs**.



* 1. Expand the workspace context menu and select **Workspace access**.to display the **Access** pane for the workpace.



* 1. In the **Access** pane, you should be able to verify that your user account has **Admin** permissions.



The workspace creator is always given **Admin** permissions on a new workspace.

* 1. Return to the PowerShell script named Exercise03.ps1 in the Windows PowerShell ISE.
  2. Delete the code in Exercise03.ps1 and replace it with the following code.

Write-Host

Connect-PowerBIServiceAccount | Out-Null

$newWorkspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

if($workspace) {

Write-Host "The workspace named $newWorkspaceName already exists"

}

else {

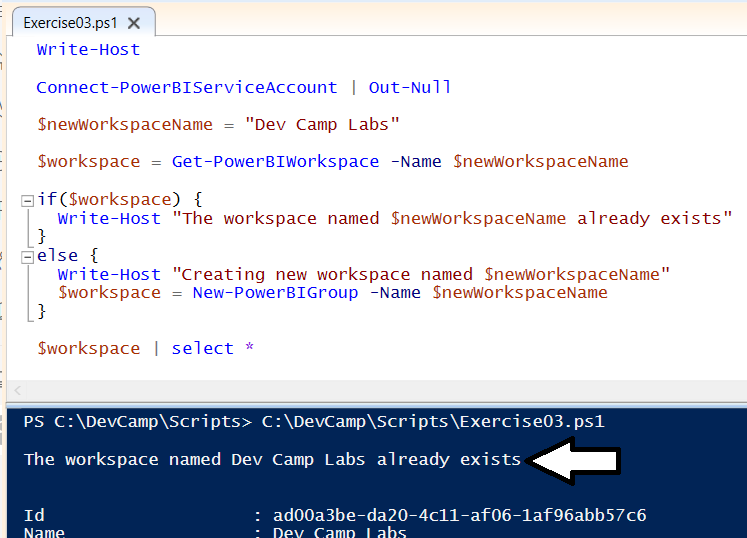
Write-Host "Creating new workspace named $newWorkspaceName"

$workspace = New-PowerBIGroup -Name $newWorkspaceName

}

$workspace | select \*

* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise03.ps1**.
  2. The code in the PowerShell script should be able to determine that the workspace named **Dev Camp Labs** already exists.



The next step require that your Power BI environment has another user account apart from the user account you are using to run your scripts. You will need the email address of any user that you want to add as a workspace user with a specific level of permissions.

1. Use the **Add-PowerBIWorkspaceUser** cmdlet to add a new workspace user.
   1. Remove the last line of code from **Exercise03.ps1** which contains the code **$workspace | select \***.
   2. Add the following code to **Exercise03.ps1** and replace the value for **$userEmail** with a valid email address for a user account in your Power BI test environment.

# add user as workspace member

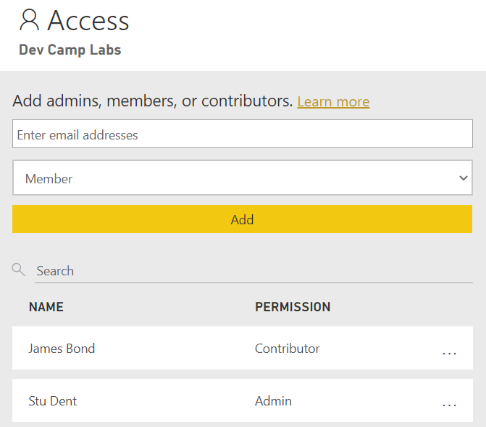
$userEmail = "JamesB@pbidev0924.onMicrosoft.com"

Add-PowerBIWorkspaceUser -Id $workspace.Id -UserEmailAddress $userEmail -AccessRight Contributor

* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise03.ps1**.

When it runs, the script should add a new user to the target workspace with permissions of a contributor.

* 1. Navigate to the Power BI Service in the browser and verify that you can see the new workspace named **Dev Camp Labs**.
  2. Expand the workspace context menu and select **Workspace access**.to display the **Access** pane for the workpace.
  3. In the **Access** pane, you should be able to verify that the new user you added has **Contributor** permissions.



### Exercise 4: Write a Script to Upload and Publish Content

In this exercise, you will write the PowerShell code required to upload PBIX files to publish and update datasets and reports.

1. Create a new PowerShell script named **Exercise04.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise04.ps1** using the following path.

C:\DevCamp\Scripts\Exercise04.ps1

* 1. Copy and paste the following code to provide a starting point for **Exercise04.ps1**.

Write-Host

Connect-PowerBIServiceAccount | Out-Null

$newWorkspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

if($workspace) {

Write-Host "The workspace named $newWorkspaceName already exists"

}

else {

Write-Host "Creating new workspace named $newWorkspaceName"

$workspace = New-PowerBIGroup -Name $newWorkspaceName

}

1. Add PowerShell code to publish a PBIX file.
   1. Add the following code to the bottom of **Exercise04.ps1**.

$pbixFilePath = "$PSScriptRoot\COVID-US.pbix"

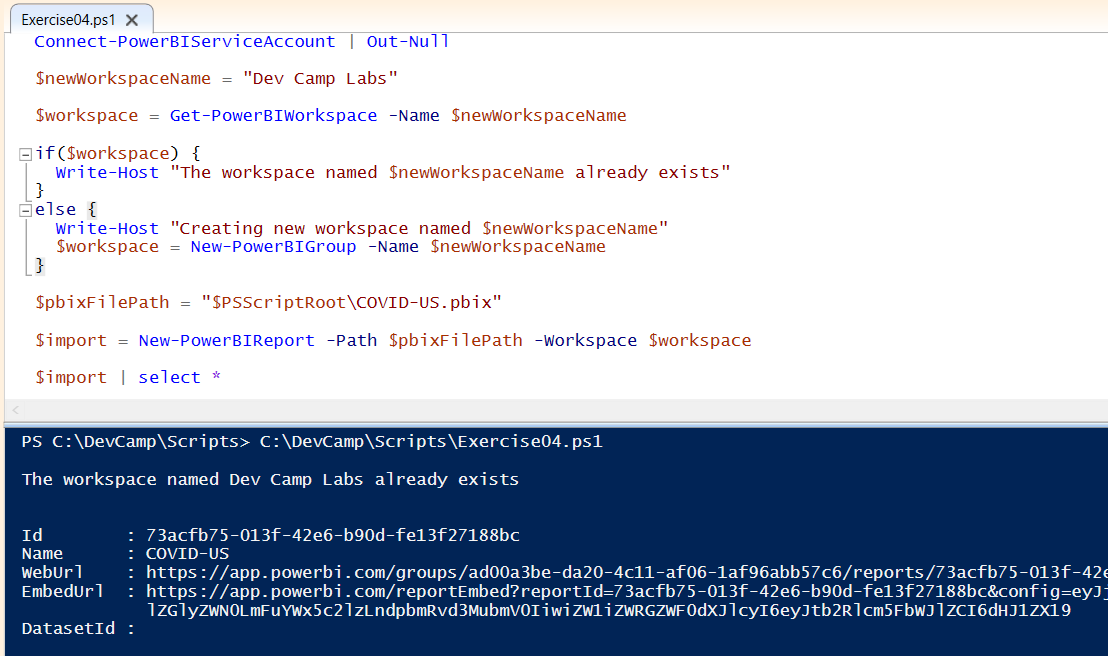
$import = New-PowerBIReport -Path $pbixFilePath -Workspace $workspace -ConflictAction CreateOrOverwrite

$import | select \*

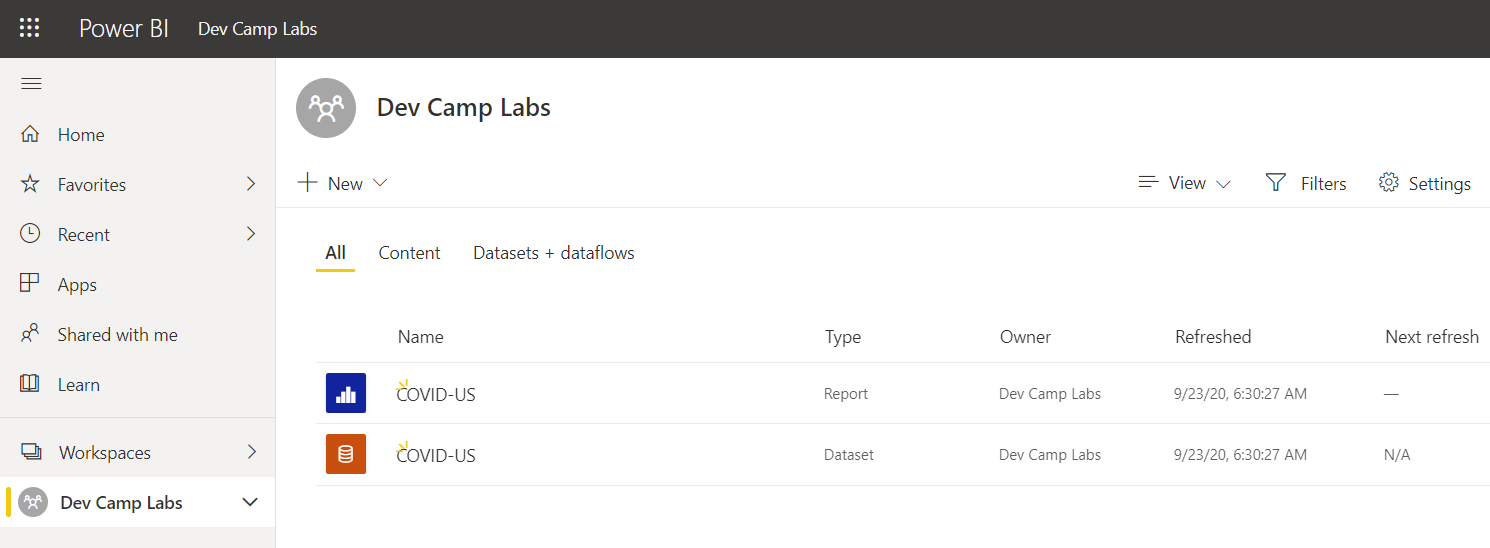
In the downloaded the student files, there should already be a PBIX file named **COVID-US.pbix** in the **Script** folder. The path created by the PowerShell expression **$PSScriptRoot\COVID-US.pbix** should reference this PBIX file. If the PBIX file named **COVID-US.pbix** is located in a different place on your PC, you should update the script accordingly.

Note the **-ConflictAction** parameter in the call to **New-PowerBIReport** has been given a value of **CreateOrOverwrite**. This parameter value is important. It causes the import behavior to overwrite any existing import with the same name. If you omit this parameter, you will find that it will create a report and dataset instead of overriding an existing report and dataset of the same name.

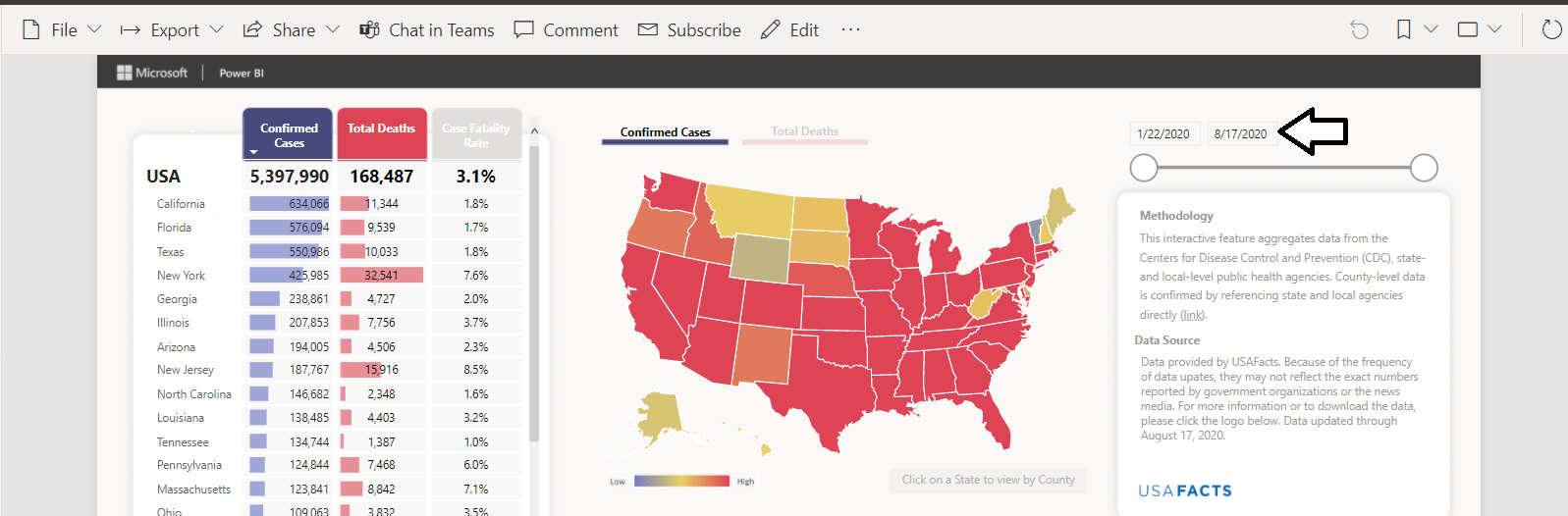
* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise04.ps1** and login when prompted.
  2. When the script runs it should import the PBIX file and display information about the imported item in the console window.



* 1. After the script runs, return to the **Dev Camp Labs** workspace in the Power BI Service
  2. Verify that PBIX file has been imported and that you can see a dataset and a report named **COVID-US**.



* 1. Open the report named COVID-US.
  2. Inspect the end date in the slicer visual in the top right and note that the last date is **8/17/2020**.



We wanted you to see the last date in the **COVID-US** report data. In the following exercise, you will write the PowerShell code required to patch the data source credentials and refresh this dataset.

### Exercise 5: Write a Script to Patch Datasource Credentials

In this exercise, you will write the PowerShell code to patch datasource credentials and to refresh a dataset.

1. Create a new PowerShell script named **Exercise05.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise05.ps1** using the following path.

C:\DevCamp\Scripts\Exercise05.ps1

* 1. Copy and paste the following code to provide a starting point for **Exercise05.ps1**.

Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$datasetName = "COVID-US"

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

$dataset = Get-PowerBIDataset -WorkspaceId $workspace.Id | Where-Object Name -eq $datasetName

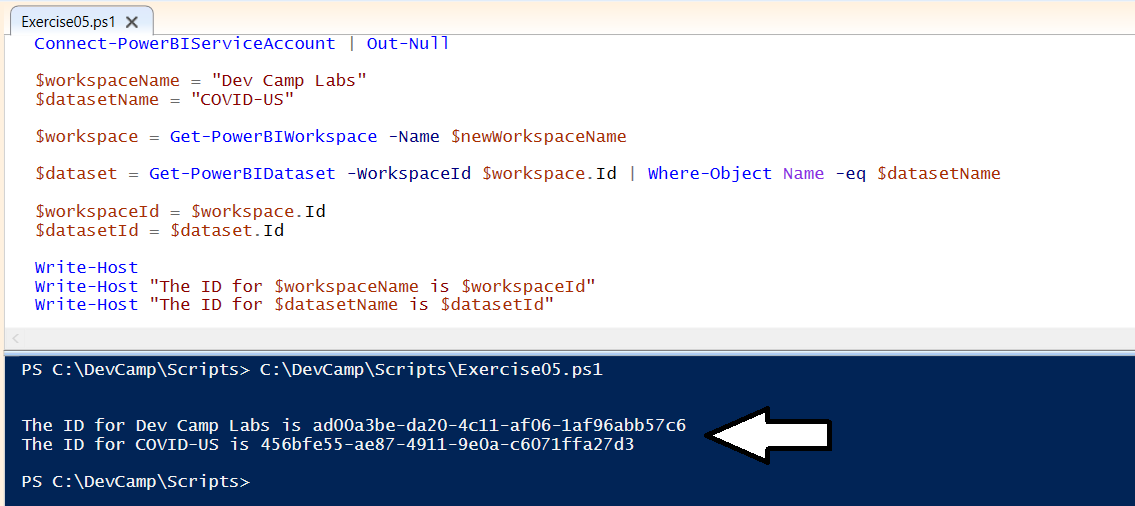
$workspaceId = $workspace.Id

$datasetId = $dataset.Id

Write-Host "The ID for $workspaceName is $workspaceId"

Write-Host "The ID for $datasetName is $datasetId"

1. Test the script.
   1. Press the **{F5}** key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.
   2. When the script runs it should display the GUIDs of the workspace and dataset in the console window.



1. Add the PowerShell code to enumerate through the datasource behind the **COVID-US** dataset.
   1. In **Exercise05.ps1**, delete the 3 lines of code that appear at the end an call **Write-Host**.
   2. Add the following code to the bottom of **Exercise05.ps1**.

$datasources = Get-PowerBIDatasource -WorkspaceId $workspaceId -DatasetId $datasetId

foreach($datasource in $datasources) {

$datasource | select \*

}

* 1. At this point, the contents of Exercise05.ps1 should match the following code listing.

Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$datasetName = "COVID-US"

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

$dataset = Get-PowerBIDataset -WorkspaceId $workspace.Id | Where-Object Name -eq $datasetName

$workspaceId = $workspace.Id

$datasetId = $dataset.Id

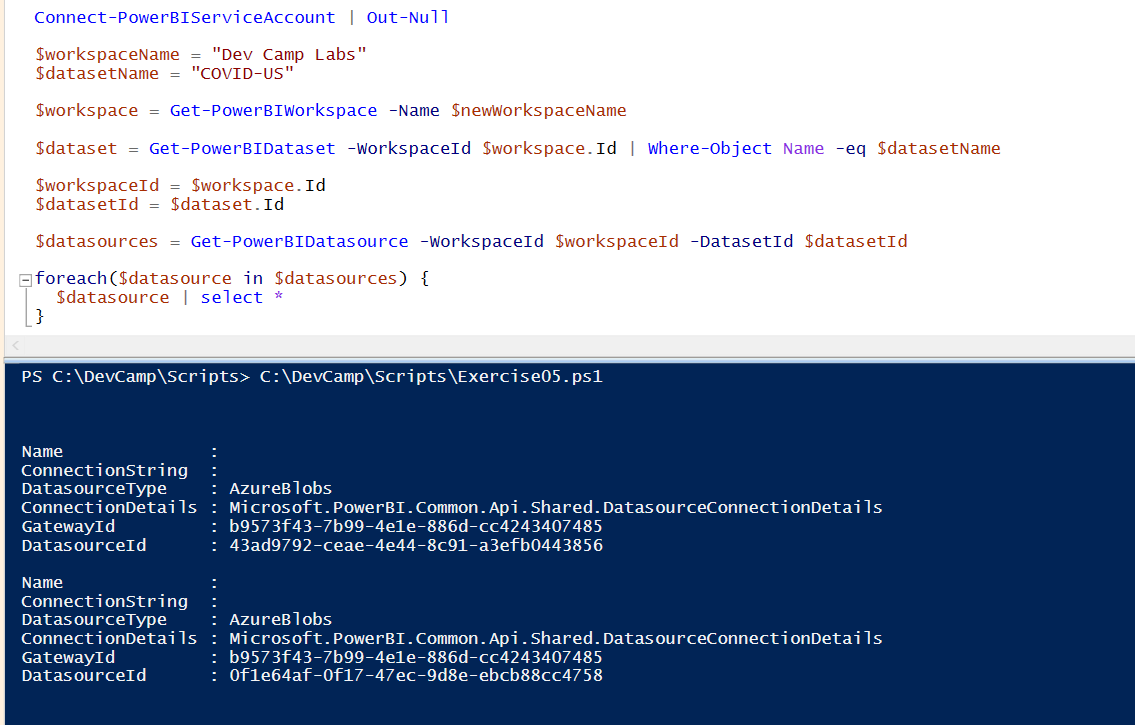
$datasources = Get-PowerBIDatasource -WorkspaceId $workspaceId -DatasetId $datasetId

foreach($datasource in $datasources) {

$datasource | select \*

}

1. Test the script.
   1. Press the **{F5}** key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.
   2. When the script runs it should display the properties of the two datasources associated with the **COVID-US** dataset.



Note that for each datasource, there is a **DatasourceId** and a **GatewayId**. It can be confusing at first when you learn that all datasources have a **GatewayId** even in cases when there is no Power BI Data Gateway involved. As you will see, the GatewayId is important because you must determine its value in order to parse together the REST URL used to patch the datasource credentials.

1. Add code to patch the datasource credentials using anonymous access.
   1. At this point, the **foreach** loop at the bottom of **Exercise05.ps1** looks like this.

foreach($datasource in $datasources) {

$datasource | select \*

}

* 1. Update the foreach loop with the following code.

foreach($datasource in $datasources) {

# parse together REST URL to reference datasource to be patched

$gatewayId = $datasource.gatewayId

$datasourceId = $datasource.datasourceId

$datasourePatchUrl = "gateways/$gatewayId/datasources/$datasourceId"

Write-Host "Patching credentials for $datasourceId"

# create HTTP request body to patch datasource credentials

$patchBody = @{

"credentialDetails" = @{

"credentials" = "{""credentialData"":""""}"

"credentialType" = "Anonymous"

"encryptedConnection" = "NotEncrypted"

"encryptionAlgorithm" = "None"

"privacyLevel" = "Public"

}

}

# convert body contents to JSON

$patchBodyJson = ConvertTo-Json -InputObject $patchBody -Depth 6 -Compress

# execute PATCH operation to set datasource credentials

Invoke-PowerBIRestMethod -Method Patch -Url $datasourePatchUrl -Body $patchBodyJson

}

Now that your code has patched the datasource credentials, you will be able to execute the code to start a refresh on the dataset.

1. Add code to refresh the dataset.
   1. Add the following code to the bottom of **Exercise05.ps1** after the end of the **foreach** loop.

# parse REST URL for dataset refresh

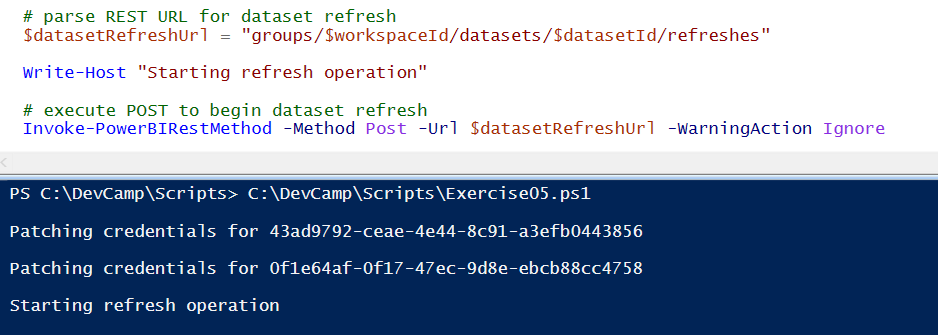
$datasetRefreshUrl = "groups/$workspaceId/datasets/$datasetId/refreshes"

Write-Host "Starting refresh operation"

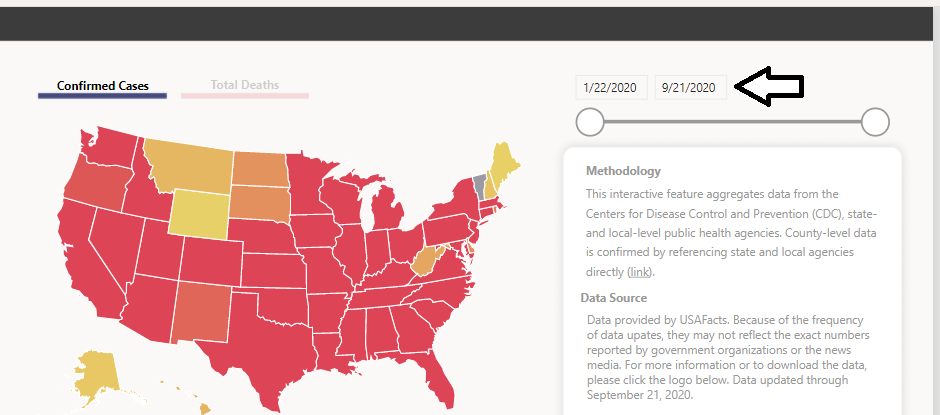
# execute POST to begin dataset refresh

Invoke-PowerBIRestMethod -Method Post -Url $datasetRefreshUrl -WarningAction Ignore

1. Test the script.
   1. Press the **{F5}** key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.
   2. When the script runs it should display the properties of the two datasources associated with the **COVID-US** dataset.



1. Inspect the **COVID-US** report to ensure the underlying dataset has been updated.
   1. In the browser, return to the Power BI Service and open the COVID-US report
   2. Verify that the latest date in the slicer now shows a more recent date than the original date of **8/17/2020**.



Microsoft updates the data behind the **COVID-US** report on a daily basis. After a refresh, the **COVID-US** report should have date up through the day just before the current day.

### Exercise 6: Write a Script to Update Dataset Parameters

In this exercise, you will begin by uploading a new PBIX file and patching datasource credentials for a SQL Server datasource. After that, you will write PowerShell code to update dataset parameters before triggering a dataset refresh.

1. Create a new PowerShell script named **Exercise06.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise06.ps1** using the following path.

C:\DevCamp\Scripts\Exercise06.ps1

* 1. Copy and paste the following code to provide a starting point for **Exercise06.ps1**.

Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

$pbixFilePath = "$PSScriptRoot\SalesByState.pbix"

$importName = " Sales Report for California"

$import = New-PowerBIReport -Path $pbixFilePath -WorkspaceId $workspace.Id `

-Name $importName -ConflictAction CreateOrOverwrite

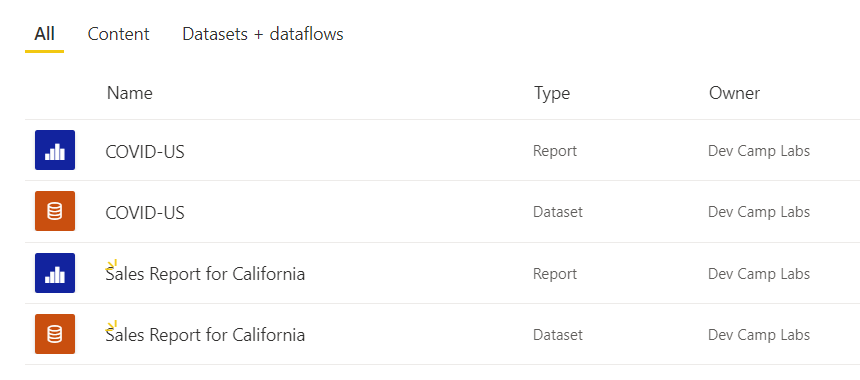
# get object for new dataset

$dataset = Get-PowerBIDataset -WorkspaceId $workspace.Id | Where-Object Name -eq $import.Name

$workspaceId = $workspace.Id

$datasetId = $dataset.Id

* 1. Press the **{F5}** key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.



1. X
   1. Xx

$datasources = Get-PowerBIDatasource -WorkspaceId $workspaceId -DatasetId $datasetId

Write-Host

foreach($datasource in $datasources) {

$gatewayId = $datasource.gatewayId

$datasourceId = $datasource.datasourceId

$datasourePatchUrl = "gateways/$gatewayId/datasources/$datasourceId"

Write-Host "Patching credentials for $datasourceId"

# add credentials for SQL datasource

$sqlUserName = "CptStudent"

$sqlUserPassword = "pass@word1"

# create HTTP request body to patch datasource credentials

$patchBody = @{

"credentialDetails" = @{

"credentials" = "{""credentialData"":[{""name"":""username"",""value"":""$sqlUserName""},{""name"":""password"",""value"":""$sqlUserPassword""}]}"

"credentialType" = "Basic"

"encryptedConnection" = "NotEncrypted"

"encryptionAlgorithm" = "None"

"privacyLevel" = "Organizational"

}

}

# convert body contents to JSON

$patchBodyJson = ConvertTo-Json -InputObject $patchBody -Depth 6 -Compress

# execute PATCH operation to set datasource credentials

Invoke-PowerBIRestMethod -Method Patch -Url $datasourePatchUrl -Body $patchBodyJson

}

$credentuals = "{""credentialData"":[{""name"":""username"",""value"":""$sqlUserName""},{""name"":""password"",""value"":""$sqlUserPassword""}]}"

1. x

### Exercise 7: Run Get-PowerBIWorkspace at Organization Scope

In this exercise, xxx.

1. Create a new PowerShell script named **Exercise05.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise05.ps1** using the following path.

C:\DevCamp\Scripts\Exercise05.ps1

1. Download the student lab files to a local folder on your developer workstation.
   1. Xx

Get-PowerBIWorkspace -Scope Organization

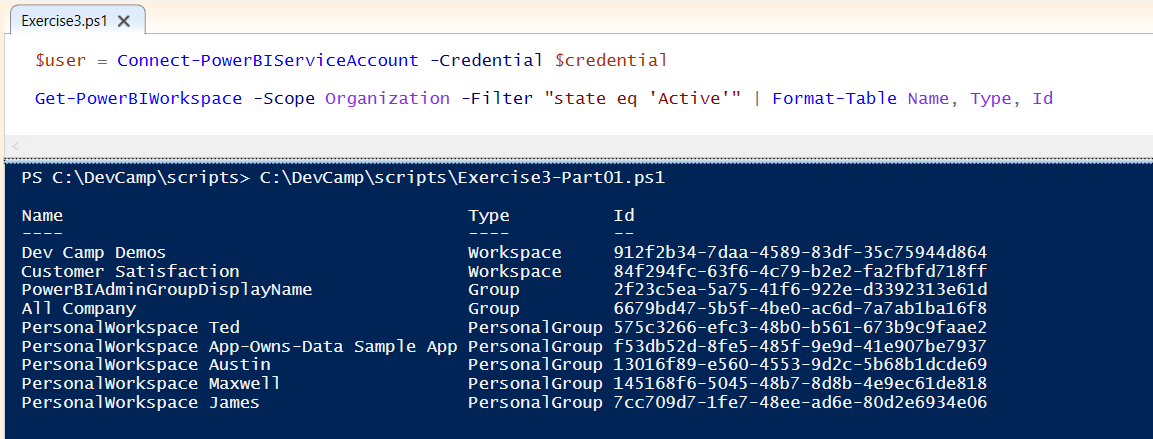
* 1. Sss

Get-PowerBIWorkspace -Scope Organization -Filter "state eq 'Active'"

* 1. Sss

Get-PowerBIWorkspace -Scope Organization -Filter "state eq 'Active'" | Format-Table Name, Type, Id

* 1. Sss



1. Zzz
   1. Zzzz

### Exercise 8: Write a Script that Exports Power BI Activity Events

In this exercise, xxx.

1. Create a new PowerShell script named **Exercise08.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise08.ps1** using the following path.

C:\DevCamp\Scripts\Exercise08.ps1

1. Cc

### Exercise 9: Write a Script that Runs as a Service Principal

In this exercise, xxx.

1. Create a new PowerShell script named **Exercise09.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise09.ps1** using the following path.

C:\DevCamp\Scripts\Exercise09.ps1

1. Takeover and Refresh a Dataset as Service Principal

### Exercise 10: Write a Script to Import a Dataflow

In this exercise, xxx.

1. Create a new PowerShell script named **Exercise10.ps1**.
   1. Return to the Windows PowerShell ISE and create a new PowerShell script,
   2. Save the new PowerShell script as **Exercise10.ps1** using the following path.

C:\DevCamp\Scripts\Exercise10.ps1

1. xx