**GitHub Copilot Chat to generate ARM and Terraform code with Copilot**

**About GitHub Copilot Chat and Visual Studio Code**

GitHub Copilot Chat allows you to ask coding questions and receive answers directly within the supported IDE. Copilot Chat can help you with a variety of coding-related tasks, like offering you code suggestions, providing natural language descriptions of a piece of code's functionality and purpose, generating unit tests for your code, and proposing fixes for bugs in your code. For more information, see "[About GitHub Copilot Chat](https://docs.github.com/en/copilot/github-copilot-chat/about-github-copilot-chat)."

**Use cases for GitHub Copilot Chat**

There are several situations in which GitHub Copilot Chat can help with coding.

* Generating unit test cases
* Explaining code
* Proposing code fixes
* Answering coding questions

In this exercise, you will utilize Copilot to generate code in ARM, Terraform, and PowerShell.

**Disclaimer**: GitHub Copilot will automatically suggest an entire function body or code in gray text. Examples of what you'll most likely see in this exercise, but the exact suggestion may vary.

**Task 1: Generate code by chat that uses ARM to deploy resources to Azure**

1. In the Visual Studio Code activity bar, click the GitHub Copilot Chat icon to open the GitHub Copilot Chat chat window.
2. At the bottom of the GitHub Copilot Chat window, in the **Ask Copilot a question or type / for topics** text box, type a coding-related question, then press Enter. For example, type "Write an ARM code for deploying a storage account to Azure with the code explanation."

A screenshot of a computer

Description automatically generated

1. GitHub Copilot Chat will process your question and provide an answer, with code suggestions when appropriate, in the chat window.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Note:** Here's an example of what you are likely to see; however, the precise recommendation could vary.

**Note**: Optionally, if GitHub Copilot Chat suggests a follow-up question above the **Ask Copilot a question or type / for topics** text box, click the follow-up question to ask it.

**Note**: If your question is outside the scope of GitHub Copilot Chat, it will tell you and may suggest an alternative question to ask.

1. You can view the response from GitHub Copilot in the chat. To insert code into a new file, click on **Ellipsis (...)** **(1)** and select **Insert Into New File** **(2)**.

A screenshot of a computer

Description automatically generated

1. Press CTRL + S to save the file. Name the file arm.json and click on **OK**

A screenshot of a computer program

Description automatically generated

**Task 2: Generate code by chat that uses Terraform to deploy resources to Azure**

1. In the Visual Studio Code activity bar, click the GitHub Copilot Chat icon to open the GitHub Copilot Chat chat window.
2. At the bottom of the GitHub Copilot Chat window, in the **Ask Copilot a question or type / for topics** text box, type a coding-related question, then press Enter. For instance, type "Write a Terraform code for deploying a storage account to Azure with the code explanation".

A screenshot of a computer

Description automatically generated

1. GitHub Copilot Chat will process your question and provide an answer, with code suggestions when appropriate, in the chat window.

A screenshot of a computer

Description automatically generated

**Note:** Here's an example of what you are likely to see; however, the precise recommendation could vary.

**Note**: Optionally, if GitHub Copilot Chat suggests a follow-up question above the **Ask Copilot a question or type / for topics** text box, click the follow-up question to ask it.

**Note**: If your question is outside the scope of GitHub Copilot Chat, it will tell you and may suggest an alternative question to ask.

1. You can view the response from GitHub Copilot in the chat. To insert code into a new file, click on **Ellipsis (...)** **(1)** and select **Insert Into New File** **(2)**.

A screenshot of a computer program

Description automatically generated

1. Press CTRL + S to save the file. Name the file terraform.tf and click on **OK**

A screenshot of a computer

Description automatically generated

**Task 3: Generate code by chat that uses PowerShell to deploy resources to Azure**

1. In the Visual Studio Code activity bar, click the GitHub Copilot Chat icon to open the GitHub Copilot Chat chat window.
2. At the bottom of the GitHub Copilot Chat window, in the **Ask Copilot a question or type / for topics** text box, type a coding-related question, then press Enter. For example, type "Write a PowerShell script for deploying a storage account to Azure."

A screenshot of a computer

Description automatically generated

1. GitHub Copilot Chat will process your question and provide an answer, with code suggestions when appropriate, in the chat window.

A screenshot of a computer

Description automatically generated

**Note:** Here's an example of what you are likely to see; however, the precise recommendation could vary.

**Note**: Optionally, if GitHub Copilot Chat suggests a follow-up question above the **Ask Copilot a question or type / for topics** text box, click the follow-up question to ask it.

**Note**: If your question is outside the scope of GitHub Copilot Chat, it will tell you and may suggest an alternative question to ask.

1. You can view the response from GitHub Copilot in the chat. To insert code into a new file, click on **Ellipsis (...)** **(1)** and select **Insert Into New File** **(2)**.

A screenshot of a computer program

Description automatically generated

1. Press CTRL + S to save the file, and you will see a recommendation to install the PowerShell extension. Click on Install. Name the file powershell.ps1 and click on **OK**.

A screenshot of a computer

Description automatically generated

**Task 4: Push code to your repository from the codespace**

1. Use the VS Code terminal to add files to the repository. Open VS Code Terminal if it's not opened yet.
2. Run the below command to add the arm.json , terraform.tf and powershell.ps1 files to the repository:
3. git add arm.json terraform.tf powershell.ps1

git add arm.json terraform.tf powershell.ps1

1. Next, from the VS Code terminal stage, commit the changes to the repository:
2. git commit -m "Copilot third commit"

git commit -m "Copilot third commit"

1. Finally, from the VS Code terminal, push to code to the repository:
2. git push

git push

A screenshot of a computer

Description automatically generated

**Note**: Wait about 60 seconds, then refresh your repository landing page for the next step.

1. You can verify the arm.json, powershell.ps1, and terraform.tf files available in your GitHub repository.

A screenshot of a computer

Description automatically generated

1. Click on Next from the bottom right to continue with the next exercise.

**Summary**

In this exercise, you have employed Copilot to automatically generate code in both ARM, Terraform, and PowerShell programming languages.

**xercise 5: Using GitHub Copilot for Code Refactoring**

Duration: 40 minutes

GitHub code refactoring refers to the process of restructuring and improving the quality of code in a GitHub repository without changing its external behavior. Code refactoring aims to enhance readability, maintainability, and performance while reducing technical debt and potential bugs.

In this exercise, you will participate in a learning or practice activity where your main goal will be to use GitHub Copilot for code refactoring using the C# programming language. You will also be generating unit test functions using GitHub Copilot Chat.

**Disclaimer**: GitHub Copilot will automatically suggest an entire function body or code in gray text. Examples of what you'll most likely see in this lab, but the exact suggestion may vary.

**Task 1: Understand the code already available for you.**

**Installing C Sharp (C#) extension in VS Code**

1. To install the C Sharp (C#) extension, the following steps are to be performed within Visual Studio Code:
   * Click on the **Extensions** **(1)** icon in the activity bar present on the left side of the Visual Studio Code Window.
   * In the "Search Extensions in Marketplace" search box, type and search for the **C#** **(2)** extension.
   * Select **C#** **(3)** from the list of results that show up.
   * Click on the **Install** **(4)** button.

A screenshot of a computer

Description automatically generated

1. To demonstrate the GitHub code refactor, let us take an example of a poorly written code and analyze it. The below code is written using the C# programming language.
2. using System;
3. class Program
4. {
5. const int MAX = 100;
6. static int Sum(int[] arr, int n)
7. {
8. int result = 0;
9. for (int i = 0; i < n; i++)
10. {
11. result += arr[i];
12. }
13. return result;
14. }
15. static void Main()
16. {
17. int n;
18. Console.Write("Enter the number of elements (1-100): ");
19. if (!int.TryParse(Console.ReadLine(), out n) || n < 1 || n > MAX)
20. {
21. Console.WriteLine("Invalid input. Please provide a digit ranging from 1 to 100.");
22. Environment.Exit(1);
23. }
24. int[] arr = new int[n];
25. Console.WriteLine("Enter " + n + " integers:");
26. for (int i = 0; i < n; i++)
27. {
28. if (!int.TryParse(Console.ReadLine(), out arr[i]))
29. {
30. Console.WriteLine("Invalid input. Please enter valid integers.");
31. Environment.Exit(1);
32. }
33. }
34. int total = Sum(arr, n);
35. Console.WriteLine("Sum of the numbers: " + total);
36. // No need to free memory in C#, as it's managed by the runtime.
37. }
38. }
39. using System;
40. class Program
41. {
42. const int MAX = 100;
43. static int Sum(int[] arr, int n)
44. {
45. int result = 0;
46. for (int i = 0; i < n; i++)
47. {
48. result += arr[i];
49. }
50. return result;
51. }
52. static void Main()
53. {
54. int n;
55. Console.Write("Enter the number of elements (1-100): ");
56. if (!int.TryParse(Console.ReadLine(), out n) || n < 1 || n > MAX)
57. {
58. Console.WriteLine("Invalid input. Please provide a digit ranging from 1 to 100.");
59. Environment.Exit(1);
60. }
61. int[] arr = new int[n];
62. Console.WriteLine("Enter " + n + " integers:");
63. for (int i = 0; i < n; i++)
64. {
65. if (!int.TryParse(Console.ReadLine(), out arr[i]))
66. {
67. Console.WriteLine("Invalid input. Please enter valid integers.");
68. Environment.Exit(1);
69. }
70. }
71. int total = Sum(arr, n);
72. Console.WriteLine("Sum of the numbers: " + total);
73. // No need to free memory in C#, as it's managed by the runtime.
74. }

}

1. This code is a poorly written example of a program that prompts the user for the number of elements to sum and takes those integers as input. It employs dynamic memory allocation for the integer array and handles allocation failures with an error message. Here are a few code-related issues:
   * The code begins with the inclusion of the System namespace for essential input and output operations.
   * It defines a constant MAX with a value of 100 to represent the maximum number of elements allowed.
   * The Sum method calculates the sum of elements in an array.
   * In the Main method:
     + It prompts the user to enter the number of elements (between 1 and 100) and validates the input.
     + Allocates an array 'arr' to store the entered integers.
     + Prompts the user to enter the specified number of integers and validates each input.
     + Calculates the sum of the entered integers using the Sum method.
     + Outputs the sum to the console.
     + There's no need to manually free memory in C# because it manages memory automatically.

**Task 2: Use GitHub Copilot to refactor the code**

1. Now, let's use the **GitHub Copilot Chat** feature to refactor the above code.
2. Click on the **Chat** extension icon from the activity bar in the left navigation pane. This opens the GitHub Copilot Chat window into which the above code is to be **pasted**.
3. Type the prompt Refactor the code at the end of the code and then press **enter**.
4. The GitHub Copilot Chat will give the response as shown below.

A screen shot of a computer program

Description automatically generated

**Task 3: Review the refactored code and understand how it works.**

1. The newly generated code will look as shown below:
2. using System;
3. class Program
4. {
5. const int MAX = 100;
6. static int Sum(int[] arr)
7. {
8. int result = 0;
9. foreach (int num in arr)
10. {
11. result += num;
12. }
13. return result;
14. }
15. static void Main()
16. {
17. int n;
18. Console.Write("Enter the number of elements (1-100): ");
19. if (!int.TryParse(Console.ReadLine(), out n) || n < 1 || n > MAX)
20. {
21. Console.WriteLine("Invalid input. Please enter a number between 1 and 100.");
22. Environment.Exit(1);
23. }
24. int[] arr = new int[n];
25. Console.WriteLine("Enter " + n + " integers:");
26. for (int i = 0; i < n; i++)
27. {
28. if (!int.TryParse(Console.ReadLine(), out arr[i]))
29. {
30. Console.WriteLine("Invalid input. Please enter valid integers.");
31. Environment.Exit(1);
32. }
33. }
34. int total = Sum(arr);
35. Console.WriteLine("Sum of the numbers: " + total);
36. }
37. }
38. using System;
39. class Program
40. {
41. const int MAX = 100;
42. static int Sum(int[] arr)
43. {
44. int result = 0;
45. foreach (int num in arr)
46. {
47. result += num;
48. }
49. return result;
50. }
51. static void Main()
52. {
53. int n;
54. Console.Write("Enter the number of elements (1-100): ");
55. if (!int.TryParse(Console.ReadLine(), out n) || n < 1 || n > MAX)
56. {
57. Console.WriteLine("Invalid input. Please enter a number between 1 and 100.");
58. Environment.Exit(1);
59. }
60. int[] arr = new int[n];
61. Console.WriteLine("Enter " + n + " integers:");
62. for (int i = 0; i < n; i++)
63. {
64. if (!int.TryParse(Console.ReadLine(), out arr[i]))
65. {
66. Console.WriteLine("Invalid input. Please enter valid integers.");
67. Environment.Exit(1);
68. }
69. }
70. int total = Sum(arr);
71. Console.WriteLine("Sum of the numbers: " + total);
72. }

}

1. Now, let's analyze the changes made to the code by GitHub Copilot Chat
   * Here, GitHub Copilot removed the n parameter from the Sum method since it's not needed. Instead, the Copilot used a for-each loop to iterate over the array. As a result, the code is easier to comprehend and more concise.
   * This includes the system namespace for input and output operations.
   * Defines a constant MAX with a value of 100 for the maximum number of elements allowed in an array.
   * The Sum method calculates the sum of elements in an integer array using a for-each loop.
   * In the Main method:
     + Asks the user to input the number of elements and validate it.
     + Creates an integer array to store user-entered values.
     + Prompts the user to enter integers, validates the input, and stores them in the array.
     + Calculates the sum of the integers using the Sum method.
     + Shows the sum on the console.
     + The code includes input validation and provides the sum of user-entered integers.
2. Notice that now the code is more modular, readable, and easier to understand.

**Note**: It's essential to carefully review copilot suggestions before applying them.

**Task 4: Use GitHub Copilot Chat with code to refactor the code**

In this task, you will demonstrate the chat with code feature of GitHub Copilot. With this feature, developers can engage in real-time conversations with Copilot directly through code comments, making it feel like they are collaborating with a coding partner.

1. From the codespace in the VS Code Explorer window, create a new file.

A screenshot of a computer

Description automatically generated

1. Name the file codechat.cs and you will see a recommendation to install the C# extension. Click on Install.

A screenshot of a computer

Description automatically generated

1. Copy and paste the code given below in the newly created file, i.e., codechat.cs.
2. using System;
3. class Program
4. {
5. static void Main()
6. {
7. int health = 100;
8. int score = 0;
9. Console.WriteLine("Welcome to the Adventure Game!");
10. Console.WriteLine("You are in a dark forest.");
11. while (health > 0)
12. {
13. Console.WriteLine("\nOptions:");
14. Console.WriteLine("1. Go deeper into the forest.");
15. Console.WriteLine("2. Rest by the campfire.");
16. Console.WriteLine("3. Quit the game.");
17. int choice;
18. Console.Write("Enter your choice: ");
19. if (int.TryParse(Console.ReadLine(), out choice))
20. {
21. switch (choice)
22. {
23. case 1:
24. Console.WriteLine("You go farther into the forest and discover a treasure chest!");
25. score += 10;
26. break;
27. case 2:
28. Console.WriteLine("You rest by the campfire and regain 20 health.");
29. health += 20;
30. break;
31. case 3:
32. Console.WriteLine($"Thanks for playing! Your score: {score}");
33. return;
34. default:
35. Console.WriteLine("Invalid choice. Try again.");
36. break;
37. }
38. health -= 10;
39. if (health <= 0)
40. {
41. Console.WriteLine($"Game over. Your score: {score}");
42. }
43. }
44. else
45. {
46. Console.WriteLine("Invalid input. Please enter a valid number.");
47. }
48. }
49. }
50. }
51. using System;
52. class Program
53. {
54. static void Main()
55. {
56. int health = 100;
57. int score = 0;
58. Console.WriteLine("Welcome to the Adventure Game!");
59. Console.WriteLine("You are in a dark forest.");
60. while (health > 0)
61. {
62. Console.WriteLine("\nOptions:");
63. Console.WriteLine("1. Go deeper into the forest.");
64. Console.WriteLine("2. Rest by the campfire.");
65. Console.WriteLine("3. Quit the game.");
66. int choice;
67. Console.Write("Enter your choice: ");
68. if (int.TryParse(Console.ReadLine(), out choice))
69. {
70. switch (choice)
71. {
72. case 1:
73. Console.WriteLine("You go farther into the forest and discover a treasure chest!");
74. score += 10;
75. break;
76. case 2:
77. Console.WriteLine("You rest by the campfire and regain 20 health.");
78. health += 20;
79. break;
80. case 3:
81. Console.WriteLine($"Thanks for playing! Your score: {score}");
82. return;
83. default:
84. Console.WriteLine("Invalid choice. Try again.");
85. break;
86. }
87. health -= 10;
88. if (health <= 0)
89. {
90. Console.WriteLine($"Game over. Your score: {score}");
91. }
92. }
93. else
94. {
95. Console.WriteLine("Invalid input. Please enter a valid number.");
96. }
97. }
98. }

}

1. Let's ask Copilot to use if else statements instead of the switch statement.
2. Identify the section of code where the switch statement is present and select it.
3. Right-click on the code window and click on the **Copilot** **(1)** option. From the following list of options in Copilot, select **Start Code Chat** **(2)**.

A screenshot of a computer program

Description automatically generated

1. Now type the prompt "Use if-else statements instead of the switch statement" **(1)** to make the code more understandable and click > or press Enter **(2)**. Copilot will give a response, and you can review it and click **Accept** **(3)**. Also, you can **Discard** the suggestion as depicted in the image below.

A screenshot of a computer program

Description automatically generated

**Note:** It's essential to carefully review copilot suggestions before applying them.

**Task 5: Create unit test functions**

1. From the codespace in the VS Code Explorer window, create a new file.

A screenshot of a computer

Description automatically generated

1. Name the file test.js as shown below:

A screenshot of a computer

Description automatically generated

1. Now, navigate to the GitHub Copilot Chat and give the prompt **Compose a full JavaScript program for constructing a calculator**.
2. The GitHub Copilot Chat will generate the code as shown below.

A screenshot of a computer program

Description automatically generated

1. Copy and paste the code in the file named **test.js** that you just created.
2. Now, to generate test cases for each function, select the function for which the test case has to be generated, and back in the GitHub Copilot Chat window, type **/tests** and hit enter.

A screenshot of a computer program

Description automatically generated

1. It will generate a test case for the add function.

A screenshot of a computer program

Description automatically generated

**Summary**

In this exercise, you have successfully employed GitHub Copilot to carry out code refactoring tasks and created unit test functions. Code refactoring with Copilot's assistance will make your coding projects more efficient and collaborative.

**Exercise 6: Using IDEs such as JetBrains IntelliJ for Java**

Duration: 40 minutes

GitHub Copilot, an innovative code generation tool, can be integrated with JetBrains IntelliJ, a widely used IDE for Java development. This integration enhances the Java coding experience by providing intelligent code suggestions and auto-completion within IntelliJ, boosting productivity and code quality.

In this exercise, you will use the JetBrains IntelliJ Idea IDE for Java. You will install the GitHub Copilot plugin and create a new project.

**Disclaimer**: GitHub Copilot will automatically suggest an entire function body or code in gray text. Examples of what you'll most likely see in this exercise, but the exact suggestion may vary.

**Task 1: Install the GitHub Copilot plugin and create a new project in the IntelliJ Idea IDE**

1. Navigate to the desktop and open the **IntelliJ IDEA Community** IDE.

A blue light on a black surface

Description automatically generated

1. In the IntelliJ IDEA Agreement, check **I confirm that I have read and accept the terms of this User Agreement (1)** and click on **Continue (2)**.

A screenshot of a computer screen

Description automatically generated

**Note:** When the data sharing box appears, click on **Don't Send**.

A screenshot of a computer

Description automatically generated

1. From the left menu, click on **Plugins** **(1)** and select **Marketplace** **(2)**. Search **GitHub Copilot(3)**, and once you find **GitHub Copilot** plugin, click on **Install** **(4)**.

A screenshot of a computer

Description automatically generated

**Note** : In the Third-Party Plugins notice, click **Accept**.

1. Verify the GitHub Copilot plugin is installed. Next, you need to restart the IDE by clicking on **Restart IDE**. Click on **Restart** in the pop-up once again to restart the IDE.

A screenshot of a computer program

Description automatically generated

1. Once the **IntelliJ Idea Community** IDE is restarted, select **Projects** **(1)** from the left menu and click on **New Project** **(2)**.

A screenshot of a computer

Description automatically generated

1. In the new project pane, enter the project name as **demo-copilot** **(1)**. Select language as **Java(2)** and build system as **IntelliJ** **(3)**. For JDK, click on the drop-down **(4)** and click on **+ Add SDK(5)** then click on **Download JDK** **(6)**.

A screenshot of a computer

Description automatically generated

1. You will see a pop-up to download JDK, verify the location, and click on **Download**.

A screenshot of a computer

Description automatically generated

1. Once JDK is downloaded, verify the JDK version **(1)** and click on **Create** **(2)**.

A screenshot of a computer

Description automatically generated

1. Review the created **demo-copilot** project as shown below:

A screenshot of a computer program

Description automatically generated

**Task 2: Create a basic Java project**

1. Click the **Tools (1)** menu. Click GitHub Copilot, then click **Login to GitHub (2)**.

A screenshot of a computer

Description automatically generated

1. In the "Sign in to GitHub" dialog box, to copy the device code and open the device activation window, click **Copy and Open**.

A screenshot of a computer

Description automatically generated

1. A device activation window will open in your browser. Paste the device **code (1)**, then click **Continue (2)**.

A screenshot of a device activation

Description automatically generated

1. GitHub will request the necessary permissions for GitHub Copilot. To approve these permissions, click **Authorize GitHub Copilot Plugin**.
2. After the permissions have been approved, your JetBrains IDE will show a confirmation. To begin using GitHub Copilot, click **OK**.
3. In your JetBrains IDE, right-click on the **src (1)**, select **New (2)** and click on **File (3)** to create a new Java (\*.java) file named **Test.java**.

A screenshot of a computer

Description automatically generated

1. Create a class by typing **class Test**. Copilot will suggest a class body, and you can press **tab** to accept the suggestion.

A screenshot of a computer program

Description automatically generated

1. Below the bracket of the main function, type the following function header:
2. int calculateDaysBetweenDates(

int calculateDaysBetweenDates(

1. GitHub Copilot will automatically suggest an entire function body in gray text, as shown below. The exact suggestion may vary.

A screenshot of a computer program

Description automatically generated

1. Press **Tab** to accept the suggestion.

**Task 3: Getting more suggestions**

Sometimes, you may not want to use any of the initial suggestions. You can ask GitHub Copilot to return more.

1. Remove the function you entered and type the following again:
2. int calculateDaysBetweenDates(

int calculateDaysBetweenDates(

1. Open GitHub Copilot.

-On macOS, press Option + Enter.

-On Windows or Linux, press Alt + Enter.

1. Select "Open Copilot". GitHub Copilot will open a new tab and suggest multiple options, as shown below.

A screenshot of a computer program

Description automatically generated

1. Pick a suggestion that you want to use, then click "Accept solution".
2. If you don't like any of the returned suggestions, just close the suggestions tab.

**Task 4: Getting code from a comment**

GitHub Copilot can understand significantly more context than most code assistants and can generate entire functions from something as simple as a comment.

1. Remove the function you entered and type the following comment:
2. // Identify all the images without alternate text
3. // and add a red border to them
4. void process() {
5. // Identify all the images without alternate text
6. // and add a red border to them

void process() {

1. GitHub Copilot will automatically suggest an implementation.

A screen shot of a computer

Description automatically generated

1. Click on **Next** from the bottom right to continue with the next exercise.

**Summary**

In this exercise, you have successfully configured the JetBrains IntelliJ Idea IDE for Java, installed the GitHub Copilot plugin, and explored the basic Java code that produced the suggestions using Copilot.

**Exercise 7: Using GitHub Copilot for T-SQL and YAML Code [Optional]**

Duration: 20 minutes

In this exercise, the primary objective is to leverage the capabilities of GitHub Copilot, a powerful AI-assisted coding tool, to assist you in generating code for two distinct programming languages: T-SQL and YAML. You can leverage comments to generate Copilot suggestions!

**Disclaimer**: GitHub Copilot will automatically suggest an entire function body or code in grayed text. Examples of what you'll most likely see in this exercise, but the exact suggestion may vary.

**Task 1: Generate a SQL query with GitHub Copilot using comments**

1. Navigate back to Visual Studio Code, and from inside the codespace in the VS Code explorer window, create a new file.

A screenshot of a computer

Description automatically generated

1. Name the file demo.sql and type the below comment:
2. -- Create a table for 5 products with product names and prices

-- Create a table for 5 products with product names and prices

1. To open a new tab with multiple synthesized solutions, press Ctrl+Enter. GitHub Copilot will synthesize around 10 different code suggestions in a new tab. You can view the solutions, and to accept a suggestion, you need to click on Accept Solution above the suggestion and then save the file.

A screenshot of a computer

Description automatically generated

1. After accepting the suggestion, review it carefully before applying it.

A screen shot of a computer program

Description automatically generated

**Note**: You may not see the same suggestions as shown in the screenshot; exact suggestions may vary.

**Task 2: Generate YAML with GitHub Copilot using comments**

1. From inside the codespace in the VS Code Explorer window, create a new file.

A screenshot of a computer

Description automatically generated

1. Name the file deploy-app.yml and type the below comment:
2. # Create a GitHub action to email a report from a file at 6 a.m. daily

# Create a GitHub action to email a report from a file at 6 a.m. daily

1. To open a new tab with multiple synthesized solutions, press Ctrl+Enter. GitHub Copilot will synthesize around 10 different code suggestions in a new tab. You can view the solutions, and to accept a suggestion, you need to click on Accept Solution above the suggestion and then save the file.

A screenshot of a computer

Description automatically generated

1. After accepting the suggestion, review it carefully before applying it.

A computer screen shot of a program

Description automatically generated

**Note**: You may not see the same suggestions as shown in the screenshot; exact suggestions may vary.

**Task 3: Push code to your repository from the codespace**

1. Use the VS Code terminal to add files to the repository. Open VS Code Terminal if it's not opened yet.
2. Run the below command to add all the files to the repository:
3. git add --all

git add --all

1. Next, from the VS Code terminal stage, commit the changes to the repository:
2. git commit -m "Copilot fourth commit"

git commit -m "Copilot fourth commit"

1. Finally, from the VS Code terminal, push code to the repository:
2. git push

git push

A screenshot of a computer program

Description automatically generated

**Note**: Wait about 60 seconds, then refresh your repository landing page for the next step.

1. You can verify the newly added files available in your GitHub repository.

A screenshot of a computer

Description automatically generated

**Summary**

In this exercise, you have successfully generated code for SQL and YAML using comments with the help of GitHub Copilot.

**Exercise 9: Working with the Copilot for Machine Learning [Optional]**

Duration: 90 minutes

Working with Copilot for machine learning involves leveraging GitHub Copilot, an AI-powered code completion tool developed by GitHub in collaboration with OpenAI. Here's a quick summary of the key steps and considerations:

Installation: Ensure you have GitHub Copilot installed as an extension in your integrated development environment (IDE), such as Visual Studio Code.

GitHub Integration: Link your IDE to your GitHub account to enable seamless integration. This allows Copilot to access your code repositories and provide context-aware suggestions.

Machine Learning Frameworks: Copilot supports various Machine Learning libraries and frameworks like TensorFlow, PyTorch, scikit-learn, and more. It can assist with code generation for tasks like data preprocessing, model building, and evaluation.

In this exercise, you will be cloning the Git repository with the required dataset into your environment. Here, you will be working with Copilot for Machine Learning, which involves leveraging GitHub Copilot.

**Disclaimer**: GitHub Copilot will automatically suggest an entire function body or code in grayed text. Examples of what you'll most likely see in this exercise, but the exact suggestion may vary.

**Note**: Before proceeding with the exercise, make sure you have installed Python and pip packages.

**Task 1: Prerequisites and Injecting the Required Dataset into Your Environment**

1. Open VS Code Terminal by clicking on **Ellipsis (...)** **(1)**, selecting **Terminal** **(2)**, and clicking on **New Terminal** **(3)**.

A screenshot of a computer

Description automatically generated

1. Clone the below git repository in your environment.
2. git clone https://github.com/CloudLabsAI-Azure/ml-copilot-workshop.git

git clone https://github.com/CloudLabsAI-Azure/ml-copilot-workshop.git

1. Change the directory in the terminal by running the below command:
2. cd ml-copilot-workshop

cd ml-copilot-workshop

1. To install all the required Python dependencies in your environment before working with Copilot run the below command in your terminal:
2. pip install -r requirements.txt

pip install -r requirements.txt

**Task 2: Auto\_Completion of Code with Different Experiments**

1. From the VS Code explorer window, right-click on the folder named **ml-copilot-workshop** **(1)**and click on **New File**. Name the file Experiments.ipynb **(2)** and verify that your new file looks as shown below:

A screenshot of a computer

Description automatically generated

1. Type the below comments to import all the libraries where the Copilot automatically prompts all the libraries; press "tab" and then press enter to get output.
2. # Import libraries with respect to loading data and creating a random forest model

# Import libraries with respect to loading data and creating a random forest model

A screenshot of a computer

Description automatically generated

1. Accept all the suggestions for importing libraries as shown in the below screenshot **(1)** and click the **Run** **(2)** button to execute the cell. Click on +Code **(3)** to add the new cell.

A screenshot of a computer program

Description automatically generated

**Note**: While running the cell, you may need to install the required packages and select the kernel.

1. Type the below comments to load the data using the Copilot prompt. Press "enter" to get into the next line and review the suggestion, and press "tab" to accept the suggestion.
2. # Load the data from a csv file, and the name of the file is diabetes.csv

# Load the data from a csv file, and the name of the file is diabetes.csv

A screenshot of a computer

Description automatically generated

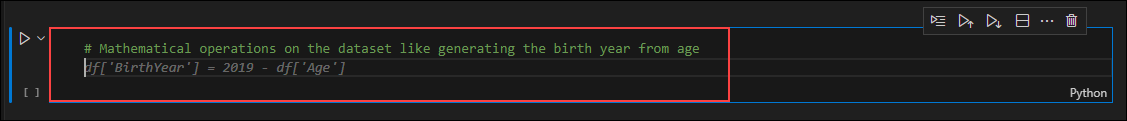
1. Now click on **+Code** and move on to the next task.

**Task 3: Mathematical and Machine Learning with Different Examples**

**Task 3.1: Mathematical Operations**

1. Type the below comments to performe the first mathematical experiment would be to generate the birth year from the age column present in the dataset, press "tab", and then press "enter".
2. # Mathematical operations on the dataset, like generating the birth year from age

# Mathematical operations on the dataset, like generating the birth year from age



1. Type the below comment, press "tab", and then press "enter" to get the output similar to the below image.
2. # Show the new column

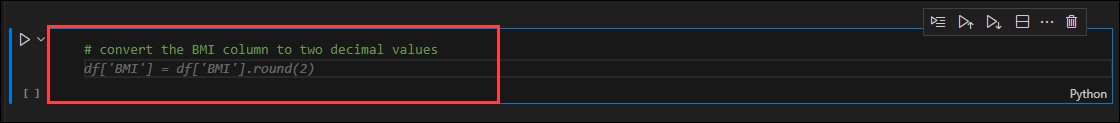
# Show the new column

A screenshot of a computer

Description automatically generated

1. Type the below comments to convert the BMI column up to two decimal values, press "tab", and then press "enter".
2. # convert the BMI column to two decimal values

# convert the BMI column to two decimal values



1. In the same code cell, type the below comment, press "tab", and then press "enter" to get the output similar to the below image.
2. # Show the new column only

# Show the new column only

A screenshot of a computer

Description automatically generated

**Task 3.2: Machine Learning**

1. Type the below comments to performe the data analysis and summary statistics on dataset, press "tab", and then press "enter" to get the output.
2. # Perform count, min, max, std, mean, 25%, 50%, and 75% on the dataset

# Perform count, min, max, std, mean, 25%, 50%, and 75% on the dataset

A screen shot of a computer

Description automatically generated

1. Once the cell run is completed, you will get an output similar to the below image.

A screenshot of a computer

Description automatically generated

1. Before building the model, the main frame is to split the data into train tests and splits and this would be done by Copilot itself. Type the below comments, press "tab" and run the cell.
2. # Split the data into training and testing data and the column name Diabetic is the target column

# Split the data into training and testing data and the column name Diabetic is the target column

A screenshot of a video game

Description automatically generated

A screenshot of a computer

Description automatically generated

**Task 4: Data Visualization and Data Transformation**

**Task 4.1: Data Visualization**

1. Click on **+Code** to open the new cell, and type the below comments to performe the basic operations on dataset, press "tab", and then press "enter" to get the output.
2. # Perform univariate analysis on the dataset and plot the graphs

# Perform univariate analysis on the dataset and plot the graphs

A screen shot of a computer

Description automatically generated

1. Once the cell run is completed, you will be getting a graphical representation output similar to the below image.

A screenshot of a computer

Description automatically generated

1. Now, type the below comments for specifying certain commonly used plots for visualization, press the "tab", and then press "enter" to get the output as shown in the below-given image.
2. # Perform scatter plot on the dataset and plot the graphs

# Perform scatter plot on the dataset and plot the graphs

A screenshot of a computer

Description automatically generated

1. Type the below comments for performing Joint Plot or Grid (rarely used plots) for visualization, press "tab", and then press "enter" to get the output as shown in the below-given image.
2. # Perform Joint Grid plot on the dataset and plot the graphs

# Perform Joint Grid plot on the dataset and plot the graphs

A screenshot of a computer

Description automatically generated

A graph with blue dots

Description automatically generated

1. Type the below comments for performing visualization on all features of the dataset, press "tab", and then press "enter" to get the output as shown in the below-given image.
2. # Perform comparison on all features of the dataset and plot the graphs in a single plot using heatmap

# Perform comparison on all features of the dataset and plot the graphs in a single plot using heatmap

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Task 4.2: Data Transformations**

1. Click **+Code** to add a new cell, and type the below comments for Standardization which is the process of scaling and centering numeric features to have a mean of 0 and a standard deviation of 1, making them comparable and suitable for certain algorithms, press "tab", and then press "enter".
2. # Perform standardization on the data

# Perform standardization on the data



A screen shot of a computer

Description automatically generated

1. You will get the output as shown in the below-given image once the code cell finishes running.

A screen shot of a computer

Description automatically generated

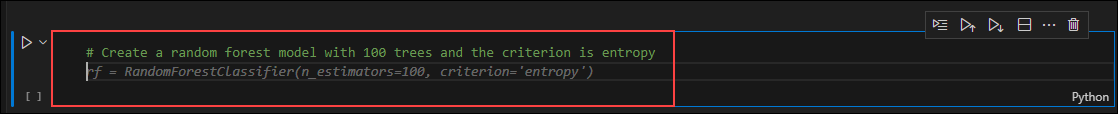
**Task 5: Training the sample model**

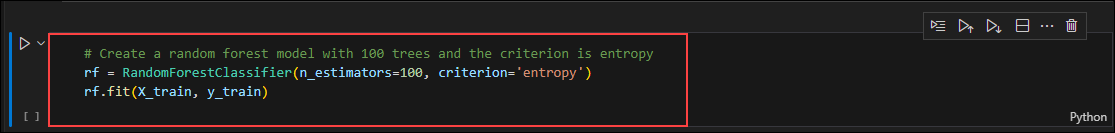
Training a sample model using Random Forest.

**Note**: The model training is a continuation of the train test split step to train the model; run the train test split step first and then continue with the model building.

1. Click **+ Code** to add a new cell, and type the below comments for creating the random forest model, press "tab", and then press "enter".
2. # Create a random forest model with 100 trees, and the criterion is entropy

# Create a random forest model with 100 trees, and the criterion is entropy





A screen shot of a computer screen

Description automatically generated

1. In a new cell, type the below comments for calculating the accuracy of the model, press "tab", and then press "enter" to see the output.
2. # Calculate the accuracy of the model

# Calculate the accuracy of the model

A black screen with a red border

Description automatically generated

**Summary**

In this exercise, you have successfully leveraged the GitHub Copilot for Machine Learning.