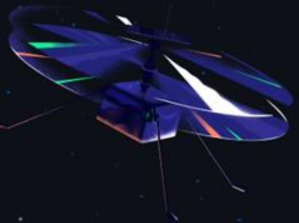
GitHub Copilot for ML/Data Ops



Beyond code completion

It's no longer just a code completion tool in your editor—it now includes a chat interface that you can use in your IDE, a command line tool via a GitHub CLI extension, a summary tool in your pull requests, a helper tool in your terminals, and much, much more.

- Copilot

Inline Chat with GitHub Copilot

- CTRL + I on Windows
- Ask specific questions
- Inline code diffs
- Slash command support
- Let’s give it a go

Copilot Chat

- Remove irrelevant requests
- Navigate through your conversation
- Use the @workspace agent
- Highlight relevant code
- Organize your conversations with threads

Copilot Edits

- Select which files to edit
- Provide the relevant context and prompt
- Review the suggested edits
- Accept or discard the suggested edits
- Iterate on the code changes

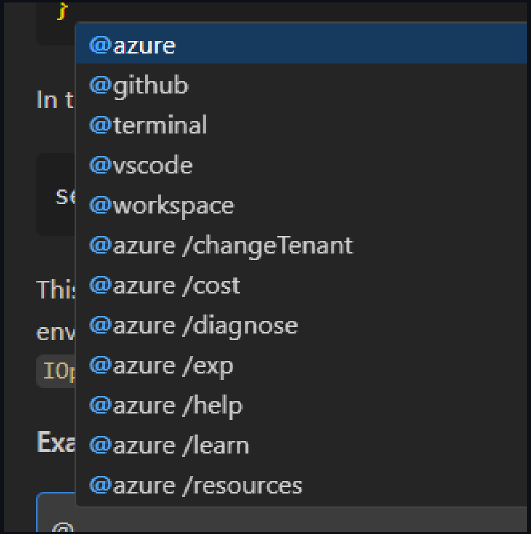
Slash Commands

Command	Description	Usage
/explain	Get code explanations	Open file with code or highlight code you want explained and type: /explain what is the fetchPrediction method?
/fix	Receive a proposed fix for the problems in the selected code	Highlight problematic code and type: /fix propose a fix for the problems in fetchAirports route
/tests	Generate unit tests for selected code	Open file with code or highlight code you want tests for and type: /tests
/help	Get help on using Copilot Chat	Type: /help what can you do?
/clear	Clear current conversation	Type: /clear
/doc	Add a documentation comment	Highlight code and type: /doc You can also press CMD+I in your editor and type /doc/ inline

Slash Commands

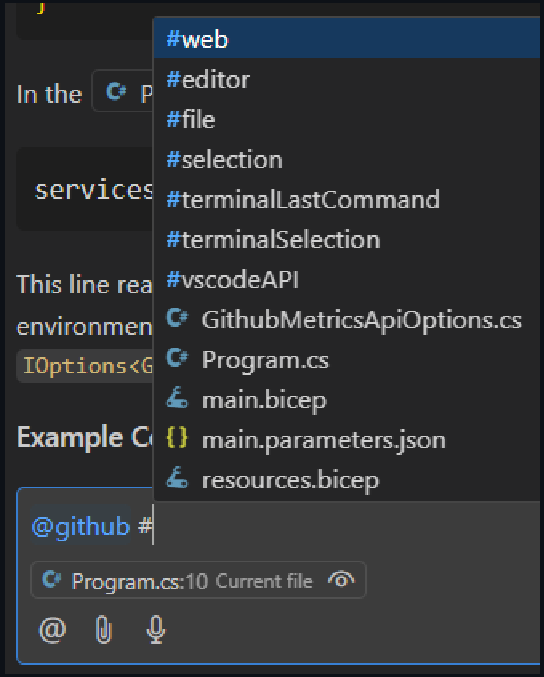
/generate	Generate code to answer your question	Type: /generate code that validates a phone number
/optimize	Analyze and improve running time of the selected code	Highlight code and type: /optimize fetchPrediction method
/clear	Clear current chat	Type: /clear
/new	Scaffold code for a new workspace	Type: /new create a new django app
/simplify	Simplify the selected code	Highlight code and type: /simplify
/feedback	Provide feedback to the team	Type: /feedback

Chat Participants



- **Doesn’t respond straight away**
- **Analyzes the current workspace files and directories**
- **Creates a context from everything it was able to collect.**

Chat Variables



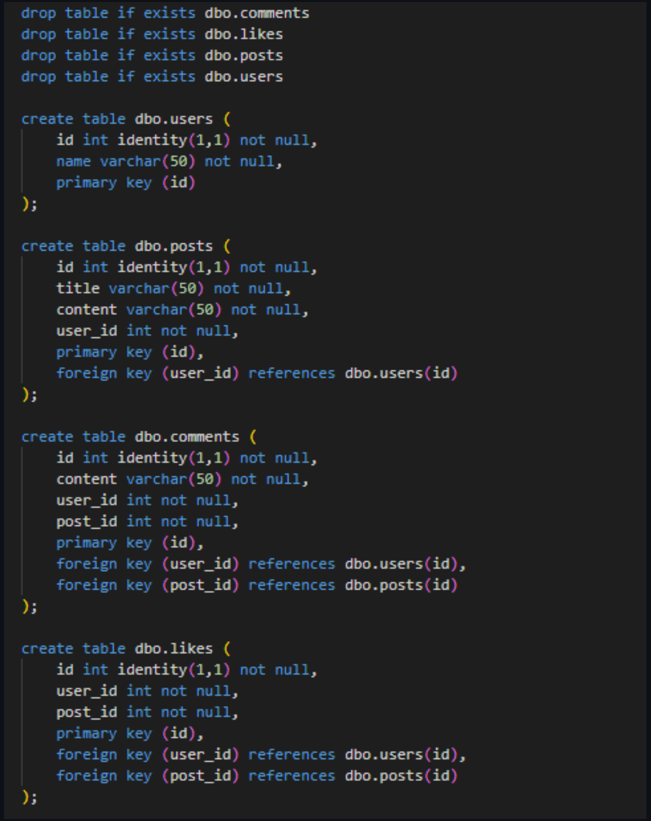
- **Use chat variables to include specific context in your prompt.**
- **Use # in the chat prompt box, followed by a chat variable.**
- **#webSearch is in GA that uses Bing web search engine**

ML/Data Ops Use Cases

Schema Generation

Generate table **schemas** with relevant foreign key constraints

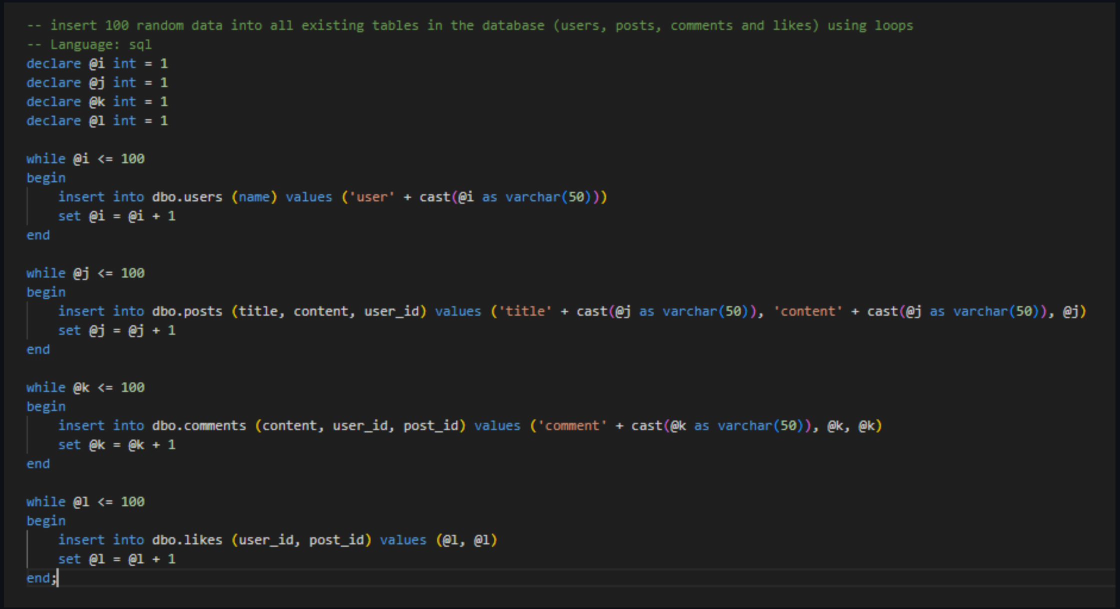
Azure Data Studio



```
drop table if exists dbo.comments
drop table if exists dbo.likes
drop table if exists dbo.posts
drop table if exists dbo.users

create table dbo.users (
    id int identity(1,1) not null,
    name varchar(50) not null,
    primary key (id)
);

create table dbo.posts (
    id int identity(1,1) not null,
    title varchar(50) not null,
    content varchar(50) not null,
    user_id int not null,
    primary key (id),
    foreign key (user_id) references dbo.users(id)
);

create table dbo.comments (
    id int identity(1,1) not null,
    content varchar(50) not null,
    user_id int not null,
    post_id int not null,
    primary key (id),
    foreign key (user_id) references dbo.users(id),
    foreign key (post_id) references dbo.posts(id)
);

create table dbo.likes (
    id int identity(1,1) not null,
    user_id int not null,
    post_id int not null,
    primary key (id),
    foreign key (user_id) references dbo.users(id),
    foreign key (post_id) references dbo.posts(id)
);
```

Data Generation

Generate dummy data for development and testing

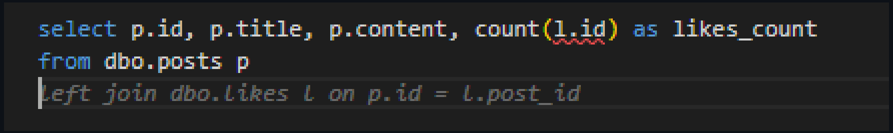


```sql
-- insert 100 random data into all existing tables in the database (users, posts, comments and likes) using loops
-- Language: sql
declare @i int = 1
declare @j int = 1
declare @k int = 1
declare @l int = 1

while @i <= 100
begin
    insert into dbo.users (name) values ('user' + cast(@i as varchar(50)))
    set @i = @i + 1
end

while @j <= 100
begin
    insert into dbo.posts (title, content, user_id) values ('title' + cast(@j as varchar(50)), 'content' + cast(@j as varchar(50)), @j)
    set @j = @j + 1
end

while @k <= 100
begin
    insert into dbo.comments (content, user_id, post_id) values ('comment' + cast(@k as varchar(50)), @k, @k)
    set @k = @k + 1
end

while @l <= 100
begin
    insert into dbo.likes (user_id, post_id) values (@l, @l)
    set @l = @l + 1
end;
```

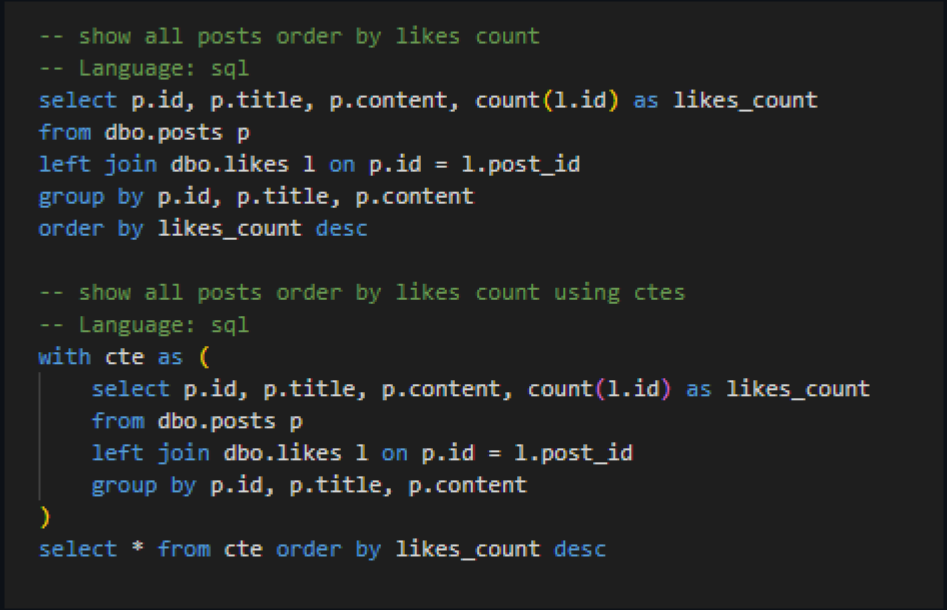
Azure Data Studio

SQL Query Generation

Generate **SQL queries** based on context and/or comments



```sql
select p.id, p.title, p.content, count(l.id) as likes_count
from dbo.posts p
left join dbo.likes l on p.id = l.post_id
```

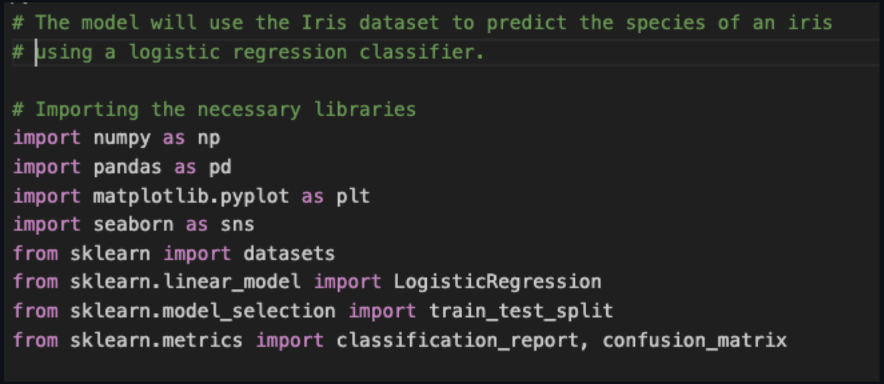


```sql
-- show all posts order by likes count
-- Language: sql
select p.id, p.title, p.content, count(l.id) as likes_count
from dbo.posts p
left join dbo.likes l on p.id = l.post_id
group by p.id, p.title, p.content
order by likes_count desc

-- show all posts order by likes count using ctes
-- Language: sql
with cte as (
    select p.id, p.title, p.content, count(l.id) as likes_count
    from dbo.posts p
    left join dbo.likes l on p.id = l.post_id
    group by p.id, p.title, p.content
)
select * from cte order by likes_count desc
```

Azure Data Studio

Library Imports

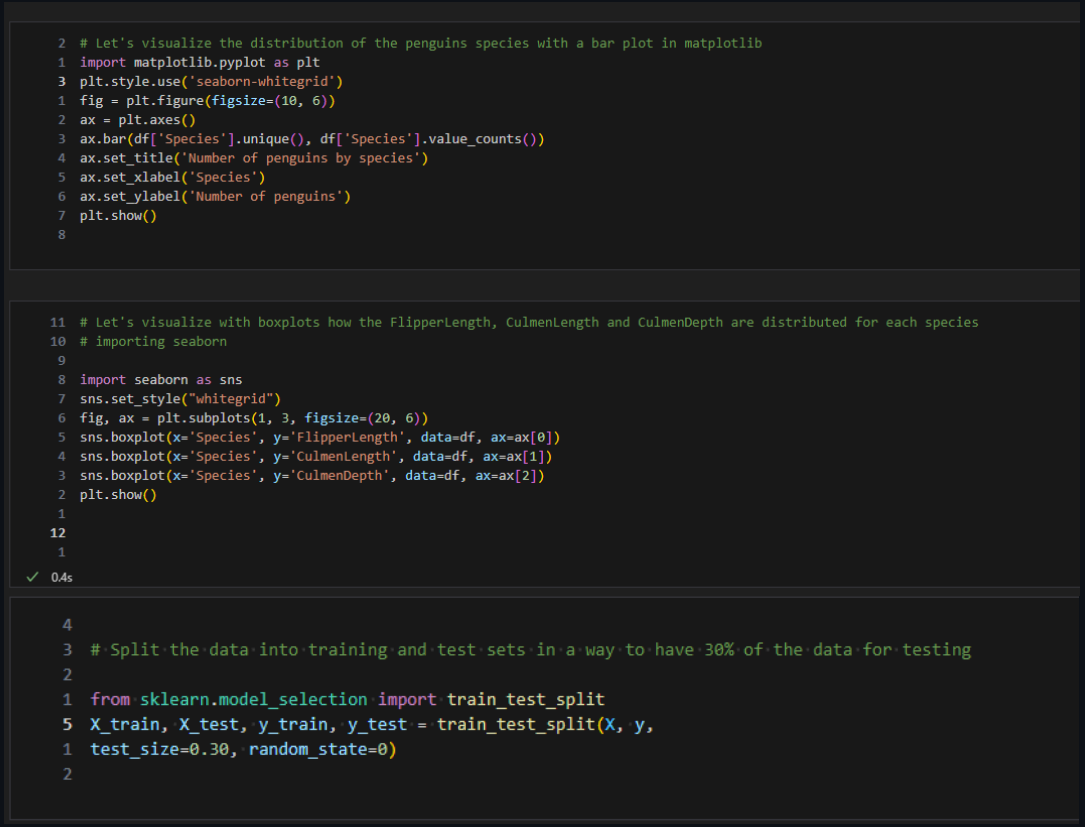
Auto import required libraries from initial input



```
# The model will use the Iris dataset to predict the species of an iris
# using a logistic regression classifier.

# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

Comment Driven Development

Generate detailed **code** based on comments



```
# Let's visualize the distribution of the penguins species with a bar plot in matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(10, 6))
ax = plt.axes()
ax.bar(df['Species'].unique(), df['Species'].value_counts())
ax.set_title('Number of penguins by species')
ax.set_xlabel('Species')
ax.set_ylabel('Number of penguins')
plt.show()

```

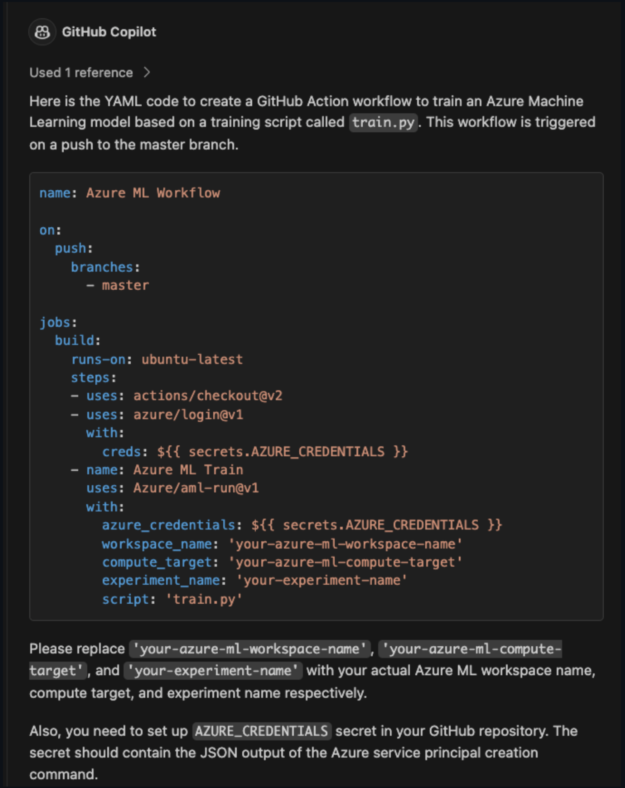
```
# Let's visualize with boxplots how the FlipperLength, CulmenLength and CulmenDepth are distributed for each species
# importing seaborn

import seaborn as sns
sns.set_style("whitegrid")
fig, ax = plt.subplots(1, 3, figsize=(20, 6))
sns.boxplot(x='Species', y='FlipperLength', data=df, ax=ax[0])
sns.boxplot(x='Species', y='CulmenLength', data=df, ax=ax[1])
sns.boxplot(x='Species', y='CulmenDepth', data=df, ax=ax[2])
plt.show()



```

✓ 0.4s

```

# Split the data into training and test sets in a way to have 30% of the data for testing

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=0)

```

DevOps

Use Copilot to generate your CI/CD deployment pipelines



GitHub Copilot

Used 1 reference

Here is the YAML code to create a GitHub Action workflow to train an Azure Machine Learning model based on a training script called `train.py`. This workflow is triggered on a push to the master branch.

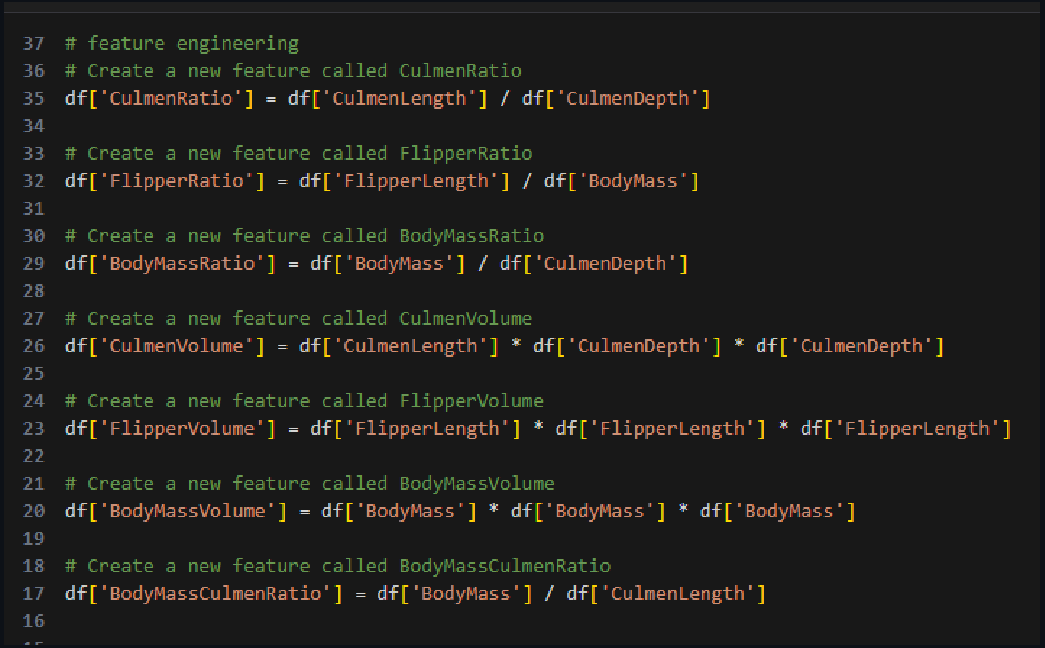
```
name: Azure ML Workflow

on:
  push:
    branches:
      - master

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}
    - name: Azure ML Train
      uses: Azure/aml-run@v1
      with:
        azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
        workspace_name: 'your-azure-ml-workspace-name'
        compute_target: 'your-azure-ml-compute-target'
        experiment_name: 'your-experiment-name'
        script: 'train.py'
```

Please replace `'your-azure-ml-workspace-name'`, `'your-azure-ml-compute-target'`, and `'your-experiment-name'` with your actual Azure ML workspace name, compute target, and experiment name respectively.

Also, you need to set up `AZURE_CREDENTIALS` secret in your GitHub repository. The secret should contain the JSON output of the Azure service principal creation command.

Feature Engineering

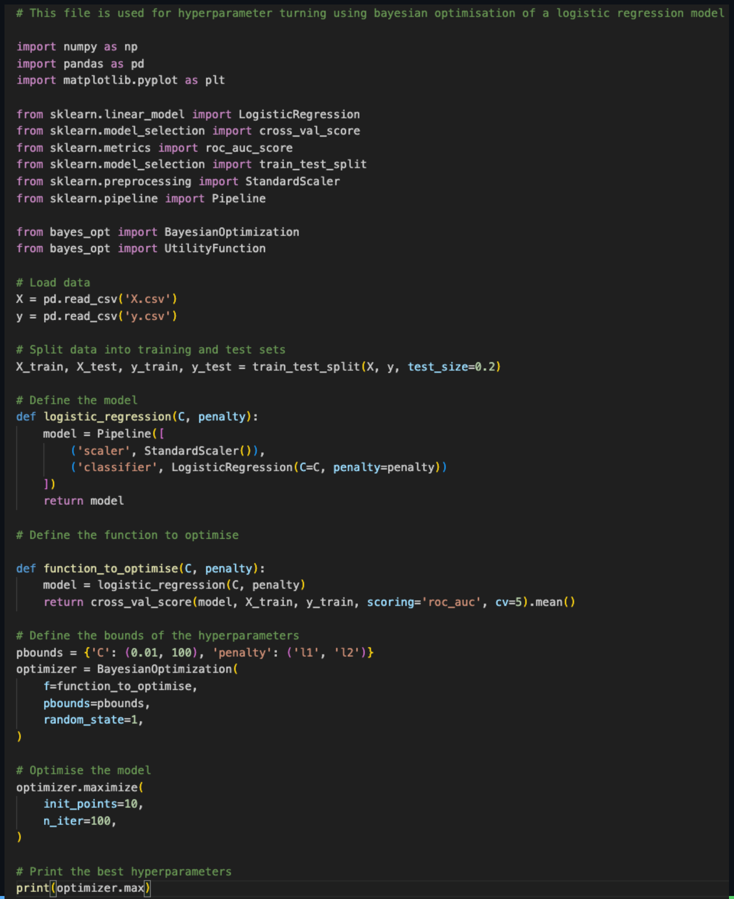
Can generate features based on the columns and their relationships with each other



```
# feature engineering
# Create a new feature called CulmenRatio
df['CulmenRatio'] = df['CulmenLength'] / df['CulmenDepth']

# Create a new feature called FlipperRatio
df['FlipperRatio'] = df['FlipperLength'] / df['BodyMass']

# Create a new feature called BodyMassRatio
df['BodyMassRatio'] = df['BodyMass'] / df['CulmenDepth']

# Create a new feature called CulmenVolume
df['CulmenVolume'] = df['CulmenLength'] * df['CulmenDepth'] * df['CulmenDepth']

# Create a new feature called FlipperVolume
df['FlipperVolume'] = df['FlipperLength'] * df['FlipperLength'] * df['FlipperLength']

# Create a new feature called BodyMassVolume
df['BodyMassVolume'] = df['BodyMass'] * df['BodyMass'] * df['BodyMass']

# Create a new feature called BodyMassCulmenRatio
df['BodyMassCulmenRatio'] = df['BodyMass'] / df['CulmenLength']

```

Hyperparameter tuning

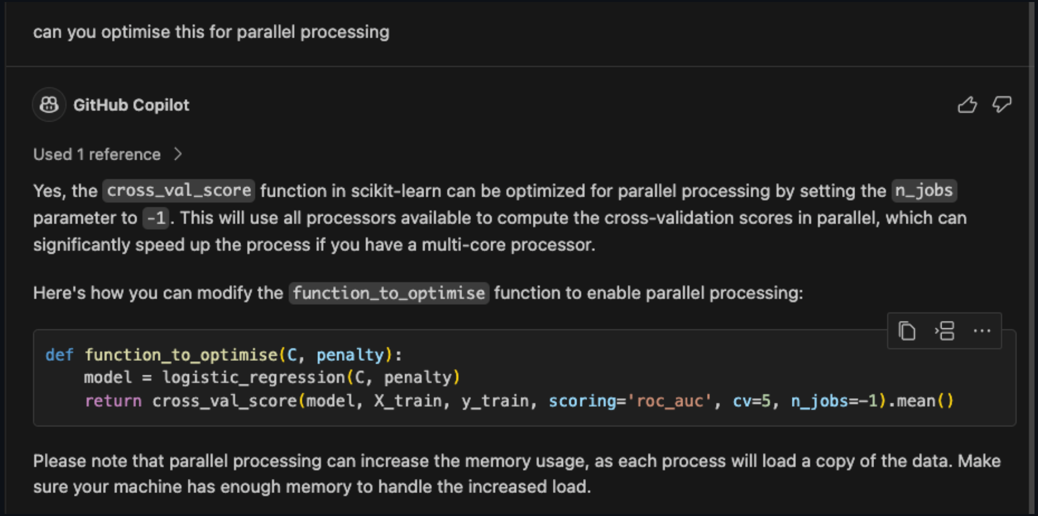
Copilot can be used to help generate code for hyperparameter tuning



```python
# This file is used for hyperparameter turning using bayesian optimisation of a logistic regression model

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from bayes_opt import BayesianOptimization
from bayes_opt import UtilityFunction

# Load data
X = pd.read_csv('X.csv')
y = pd.read_csv('y.csv')

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Define the model
def logistic_regression(C, penalty):
    model = Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', LogisticRegression(C=C, penalty=penalty))
    ])
    return model

# Define the function to optimise

def function_to_optimise(C, penalty):
    model = logistic_regression(C, penalty)
    return cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=5).mean()

# Define the bounds of the hyperparameters
pbounds = {'C': (0.01, 100), 'penalty': ('l1', 'l2')}
optimizer = BayesianOptimization(
    f=function_to_optimise,
    pbounds=pbounds,
    random_state=1,
)

# Optimise the model
optimizer.maximize(
    init_points=10,
    n_iter=100,
)

# Print the best hyperparameters
print(optimizer.max)
```

Parallel Processing Optimisation

Optimisation of code using **parallel processing** can be achieved



can you optimise this for parallel processing

GitHub Copilot

Used 1 reference

Yes, the `cross_val_score` function in scikit-learn can be optimized for parallel processing by setting the `n_jobs` parameter to `-1`. This will use all processors available to compute the cross-validation scores in parallel, which can significantly speed up the process if you have a multi-core processor.

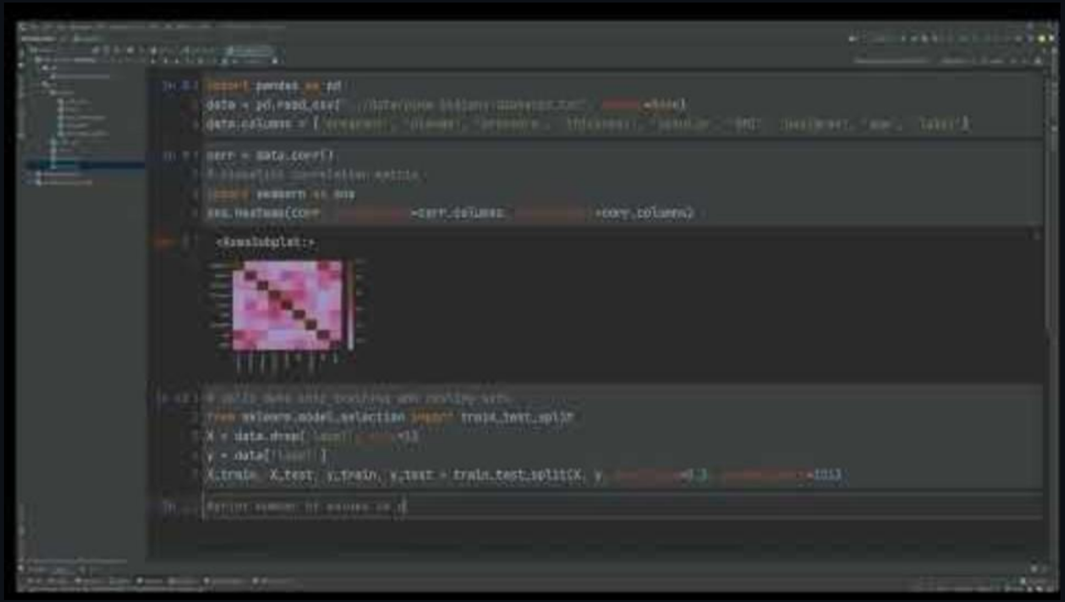
Here's how you can modify the `function_to_optimise` function to enable parallel processing:

```
def function_to_optimise(C, penalty):
    model = logistic_regression(C, penalty)
    return cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=5, n_jobs=-1).mean()
```

Please note that parallel processing can increase the memory usage, as each process will load a copy of the data. Make sure your machine has enough memory to handle the increased load.

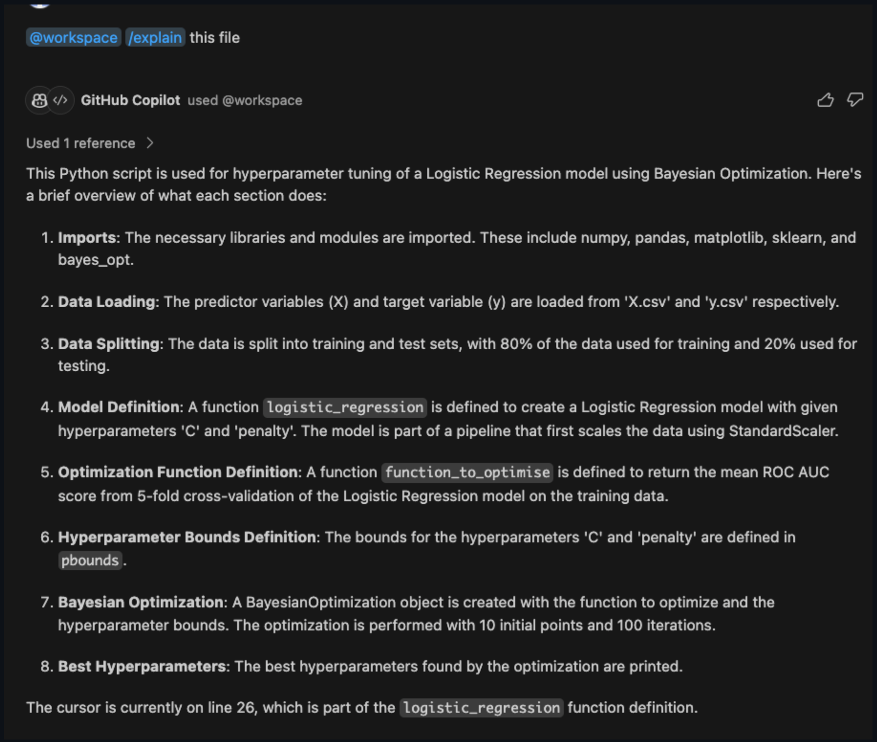
EDA (Exploratory Data Analysis)

Analyse a dataset and rapidly develop a basic model



Understand complex code

Use of /explain can assist in making code easier to understand



@workspace /explain this file

GitHub Copilot used @workspace

Used 1 reference

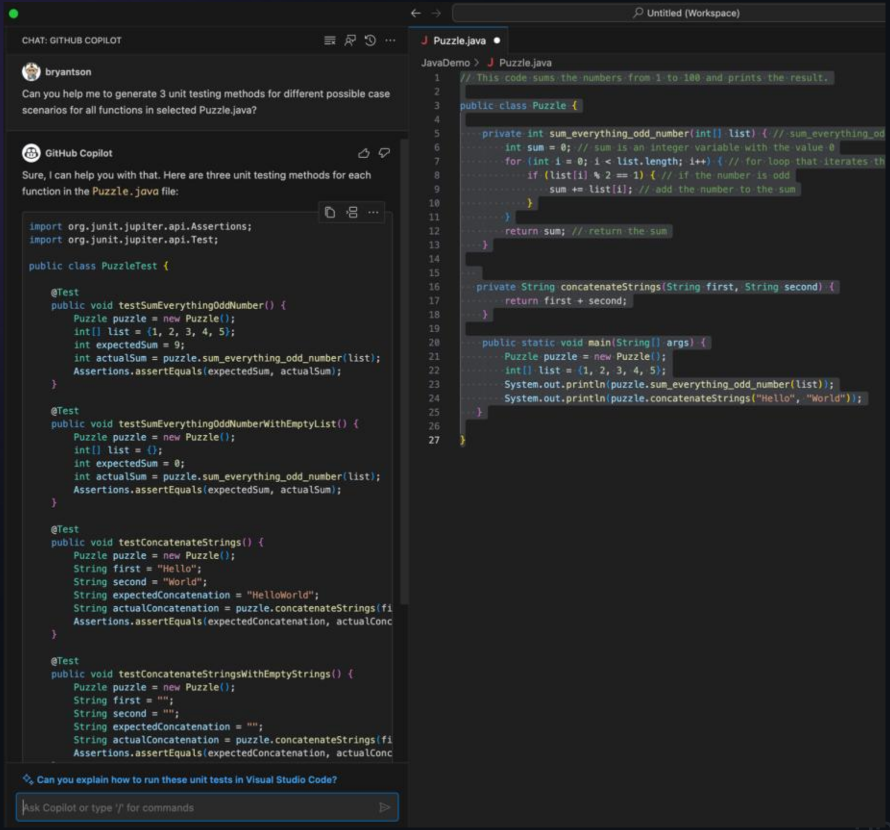
This Python script is used for hyperparameter tuning of a Logistic Regression model using Bayesian Optimization. Here's a brief overview of what each section does:

1. **Imports**: The necessary libraries and modules are imported. These include numpy, pandas, matplotlib, sklearn, and bayes_opt.
2. **Data Loading**: The predictor variables (X) and target variable (y) are loaded from 'X.csv' and 'y.csv' respectively.
3. **Data Splitting**: The data is split into training and test sets, with 80% of the data used for training and 20% used for testing.
4. **Model Definition**: A function `logistic_regression` is defined to create a Logistic Regression model with given hyperparameters 'C' and 'penalty'. The model is part of a pipeline that first scales the data using StandardScaler.
5. **Optimization Function Definition**: A function `function_to_optimise` is defined to return the mean ROC AUC score from 5-fold cross-validation of the Logistic Regression model on the training data.
6. **Hyperparameter Bounds Definition**: The bounds for the hyperparameters 'C' and 'penalty' are defined in `pbounds`.
7. **Bayesian Optimization**: A BayesianOptimization object is created with the function to optimize and the hyperparameter bounds. The optimization is performed with 10 initial points and 100 iterations.
8. **Best Hyperparameters**: The best hyperparameters found by the optimization are printed.

The cursor is currently on line 26, which is part of the `logistic_regression` function definition.

Unit Testing Generation - Copilot Chat

Works best for general helper functions with well defined input and output



bryantson

Can you help me to generate 3 unit testing methods for different possible case scenarios for all functions in selected Puzzle.java?

GitHub Copilot

Sure, I can help you with that. Here are three unit testing methods for each function in the `Puzzle.java` file:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class PuzzleTest {

    @Test
    public void testSumEverythingOddNumber() {
        Puzzle puzzle = new Puzzle();
        int[] list = {1, 2, 3, 4, 5};
        int expectedSum = 9;
        int actualSum = puzzle.sum_everything_odd_number(list);
        Assertions.assertEquals(expectedSum, actualSum);
    }

    @Test
    public void testSumEverythingOddNumberWithEmptyList() {
        Puzzle puzzle = new Puzzle();
        int[] list = {};
        int expectedSum = 0;
        int actualSum = puzzle.sum_everything_odd_number(list);
        Assertions.assertEquals(expectedSum, actualSum);
    }

    @Test
    public void testConcatenateStrings() {
        Puzzle puzzle = new Puzzle();
        String first = "Hello";
        String second = "World";
        String expectedConcatenation = "HelloWorld";
        String actualConcatenation = puzzle.concatenateStrings(fi
        Assertions.assertEquals(expectedConcatenation, actualConc
    }

    @Test
    public void testConcatenateStringsWithEmptyStrings() {
        Puzzle puzzle = new Puzzle();
        String first = "";
        String second = "";
        String expectedConcatenation = "";
        String actualConcatenation = puzzle.concatenateStrings(fi
        Assertions.assertEquals(expectedConcatenation, actualConc
```

Can you explain how to run these unit tests in Visual Studio Code?

```
// This code sums the numbers from 1 to 100 and prints the result.

public class Puzzle {

    private int sum_everything_odd_number(int[] list) { // sum_everything_od
        int sum = 0; // sum is an integer variable with the value 0
        for (int i = 0; i < list.length; i++) { // for loop that iterates th
            if (list[i] % 2 == 1) { // if the number is odd
                sum += list[i]; // add the number to the sum
            }
        }
        return sum; // return the sum
    }


   private String concatenateStrings(String first, String second) {
        return first + second;
    }

    public static void main(String[] args) {
        Puzzle puzzle = new Puzzle();
        int[] list = {1, 2, 3, 4, 5};
        System.out.println(puzzle.sum_everything_odd_number(list));
        System.out.println(puzzle.concatenateStrings("Hello", "World"));
   }

}
```

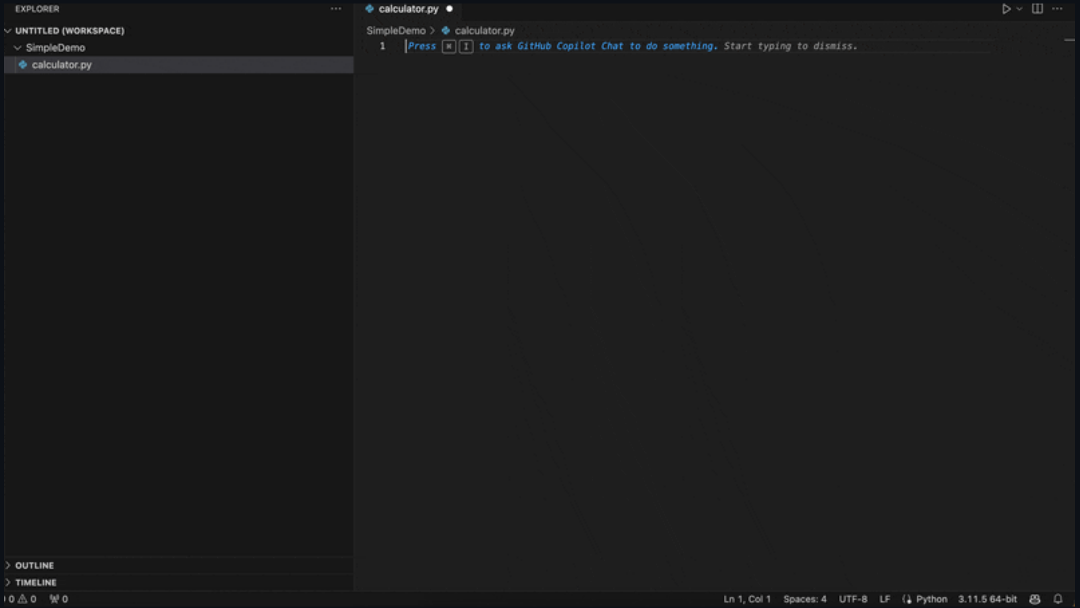
Common Problems

Problems #1:

Copilot fails to produce answer or keep repeating

Some problems

- Fails to produce answer
- Hallucination - Keeps repeating



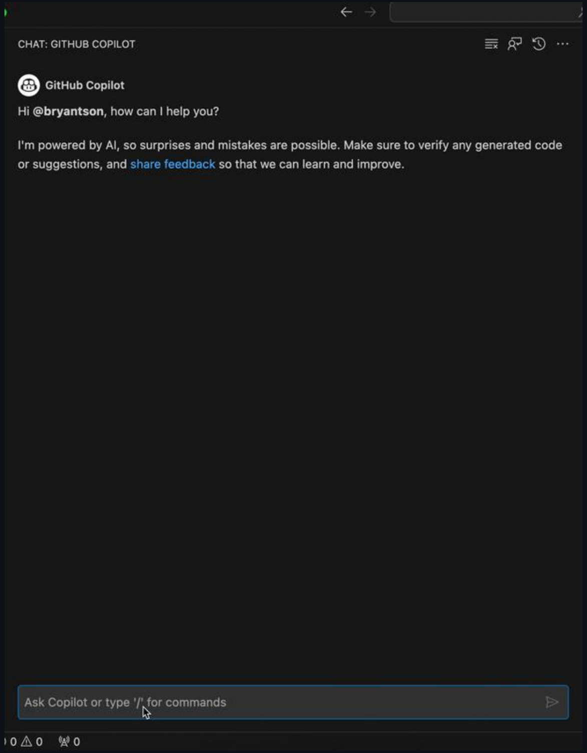
```
Press ⌘ I to ask GitHub Copilot Chat to do something. Start typing to dismiss.
```

Problems #2:

Copilot generates incorrect result

Reasons why...

- Not enough context
- Old trained data
- Generative in nature



CHAT: GITHUB COPILOT

GitHub Copilot

Hi **@bryantson**, how can I help you?

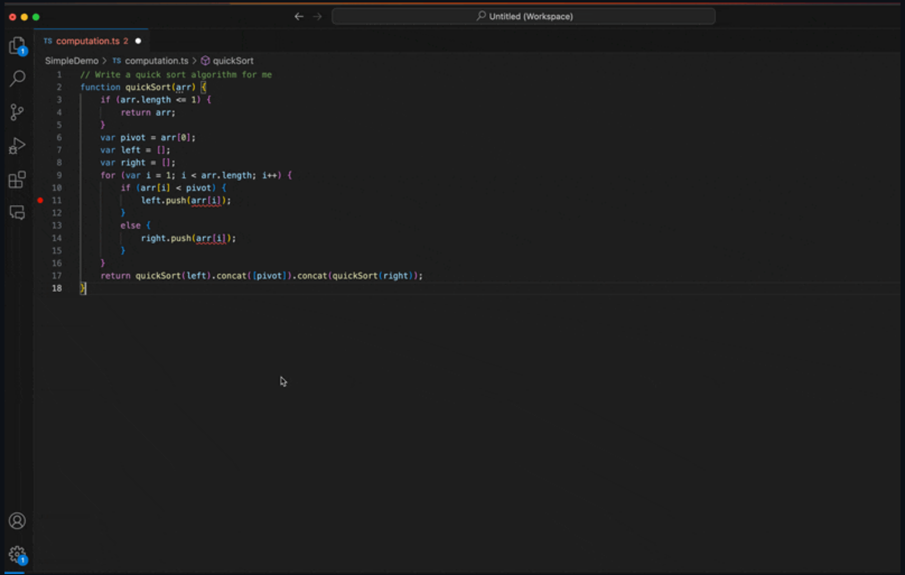
I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn and improve.

Problem #3:

Copilot suggests non-optimal solution

How to improve

- Ask in Copilot Chat
- Use in-line suggestion from Copilot



```typescript
// Write a quick sort algorithm for me
function quickSort(arr) {
    if (arr.length <= 1) {
        return arr;
    }
    var pivot = arr[0];
    var left = [];
    var right = [];
    for (var i = 1; i < arr.length; i++) {
        if (arr[i] < pivot) {
            left.push(arr[i]);
        }
        else {
            right.push(arr[i]);
        }
    }
    return quickSort(left).concat([pivot]).concat(quickSort(right));
}
```

Lab

Frequently Asked Questions

“Is the information shared with GitHub Copilot secure?

“For example, let’s say I enter some sensitive information through GitHub Copilot, is it going to store the data like my user information?”

“Can I train GitHub Copilot on my private codebase?”

“Is the Copilot suggestion from an IDE based on my computer, a project opened in my IDE, files opened through tabs or just a single file?”

“What telemetry data can I get from GitHub Copilot ?”

"How can GitHub Copilot integrate with existing pipelines/automations/scripts that we have?

"For example, I have this GitHub Actions/Azure DevOps pipeline, can GitHub Copilot integrate with those products so our code will always be secure?"

“Will Copilot replace my job”

“How much of my work can it take over?”

Q & A

Leveraging GitHub Copilot for Data Engineering

- Boosting data engineering workflows
- https://github.com/CleveritDemo/copilot-data-engineering/tree/main

Leveraging GitHub Copilot for DataOps and Machine Learning

- Use GitHub Copilot to assist your DataOps and Machine Learning tasks.
- https://github.com/arinco-crew-community/copilot-ml-data-ops/tree/main

Thank you