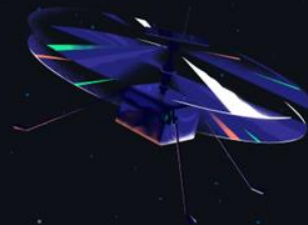




# GitHub Copilot for ML/Data Ops





# ML/Data Ops

## Use Cases

# Schema Generation

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



```
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



```
-- 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;
```

# SQL Query Generation

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



```
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
```

```
-- 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
```

# 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

```
2 # Let's visualize the distribution of the penguins species with a bar plot in matplotlib
1 import matplotlib.pyplot as plt
3 plt.style.use('seaborn-whitegrid')
1 fig = plt.figure(figsize=(10, 6))
2 ax = plt.axes()
3 ax.bar(df['Species'].unique(), df['Species'].value_counts())
4 ax.set_title('Number of penguins by species')
5 ax.set_xlabel('Species')
6 ax.set_ylabel('Number of penguins')
7 plt.show()
8
```


```
11 # Let's visualize with boxplots how the FlipperLength, CulmenLength and CulmenDepth are distributed for each species
10 # importing seaborn
9
8 import seaborn as sns
7 sns.set_style("whitegrid")
6 fig, ax = plt.subplots(1, 3, figsize=(20, 6))
5 sns.boxplot(x='Species', y='FlipperLength', data=df, ax=ax[0])
4 sns.boxplot(x='Species', y='CulmenLength', data=df, ax=ax[1])
3 sns.boxplot(x='Species', y='CulmenDepth', data=df, ax=ax[2])
2 plt.show()
1
12
1
```

✓ 0.4s

```
4
3 # Split the data into training and test sets in a way to have 30% of the data for testing
2
1 from sklearn.model_selection import train_test_split
5 X_train, X_test, y_train, y_test = train_test_split(X, y,
1 test_size=0.30, random_state=0)
2
```

# 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

```
37 # feature engineering
36 # Create a new feature called CulmenRatio
35 df['CulmenRatio'] = df['CulmenLength'] / df['CulmenDepth']
34
33 # Create a new feature called FlipperRatio
32 df['FlipperRatio'] = df['FlipperLength'] / df['BodyMass']
31
30 # Create a new feature called BodyMassRatio
29 df['BodyMassRatio'] = df['BodyMass'] / df['CulmenDepth']
28
27 # Create a new feature called CulmenVolume
26 df['CulmenVolume'] = df['CulmenLength'] * df['CulmenDepth'] * df['CulmenDepth']
25
24 # Create a new feature called FlipperVolume
23 df['FlipperVolume'] = df['FlipperLength'] * df['FlipperLength'] * df['FlipperLength']
22
21 # Create a new feature called BodyMassVolume
20 df['BodyMassVolume'] = df['BodyMass'] * df['BodyMass'] * df['BodyMass']
19
18 # Create a new feature called BodyMassCulmenRatio
17 df['BodyMassCulmenRatio'] = df['BodyMass'] / df['CulmenLength']
16
15
```

# Hyperparameter tuning

Copilot can be  
used to help  
generate code for  
hyperparameter  
tuning

```
# This file is used for hyperparameter tuning 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:

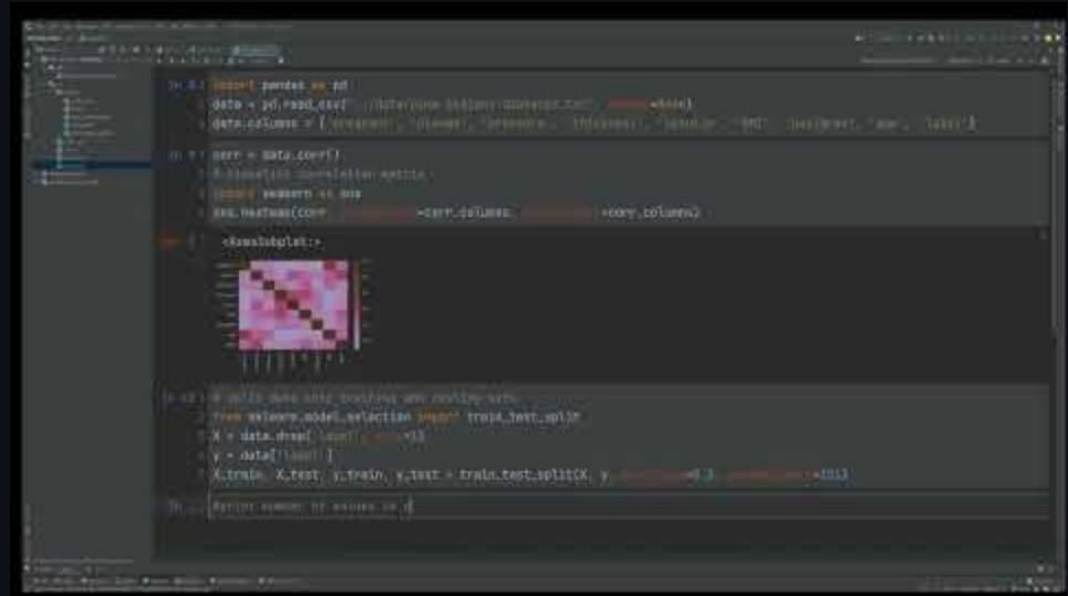


```
def function_to_optimise(C, penalty):  
    model = logistic_regression(C, penalty)  
    return cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=5, n_jobs=-1).mean()
```

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 >

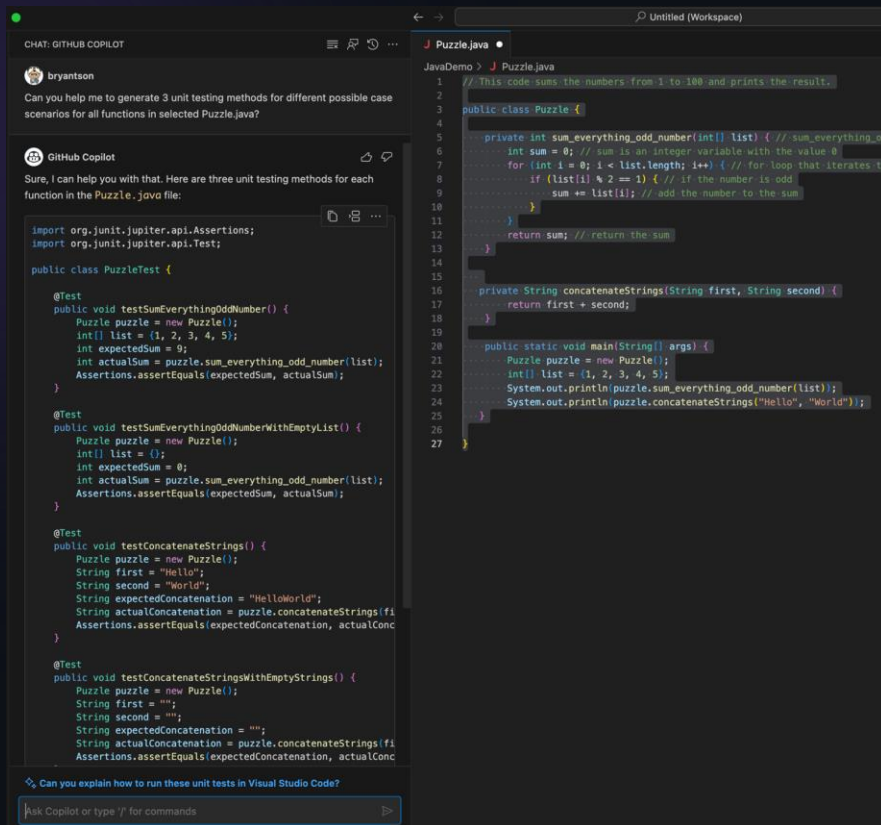
This Python script is used for hyperparameter tuning of a Logistic Regression model using Bayesian Optimization. Here's a brief overview of what each section does:

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



The screenshot shows a VS Code interface with a chat window on the left and a code editor on the right. The chat window is titled "CHAT: GITHUB COPILOT" and shows a conversation with a user named "bryantson". The user asks for help generating 3 unit testing methods for different case scenarios for all functions in a selected file named "Puzzle.java". The Copilot response provides three unit testing methods for the "Puzzle.java" file, using JUnit 5 and AssertJ. The code editor shows the "Puzzle.java" file with the following code:

```
1 // This code sums the numbers from 1 to 100 and prints the result.
2
3 public class Puzzle {
4
5     private int sum_everything_odd_number(int[] list) { // sum_everything_odd_number
6         int sum = 0; // sum is an integer variable with the value 0
7         for (int i = 0; i < list.length; i++) { // for loop that iterates to the end of the list
8             if (list[i] % 2 == 1) { // if the number is odd
9                 sum += list[i]; // add the number to the sum
10            }
11        }
12        return sum; // return the sum
13    }
14
15     private String concatenateStrings(String first, String second) {
16         return first + second;
17     }
18
19     public static void main(String[] args) {
20         Puzzle puzzle = new Puzzle();
21         int[] list = {1, 2, 3, 4, 5};
22         System.out.println(puzzle.sum_everything_odd_number(list));
23         System.out.println(puzzle.concatenateStrings("Hello", "World"));
24     }
25 }
26
27 }
```

The chat window also shows a prompt at the bottom: "Can you explain how to run these unit tests in Visual Studio Code?"

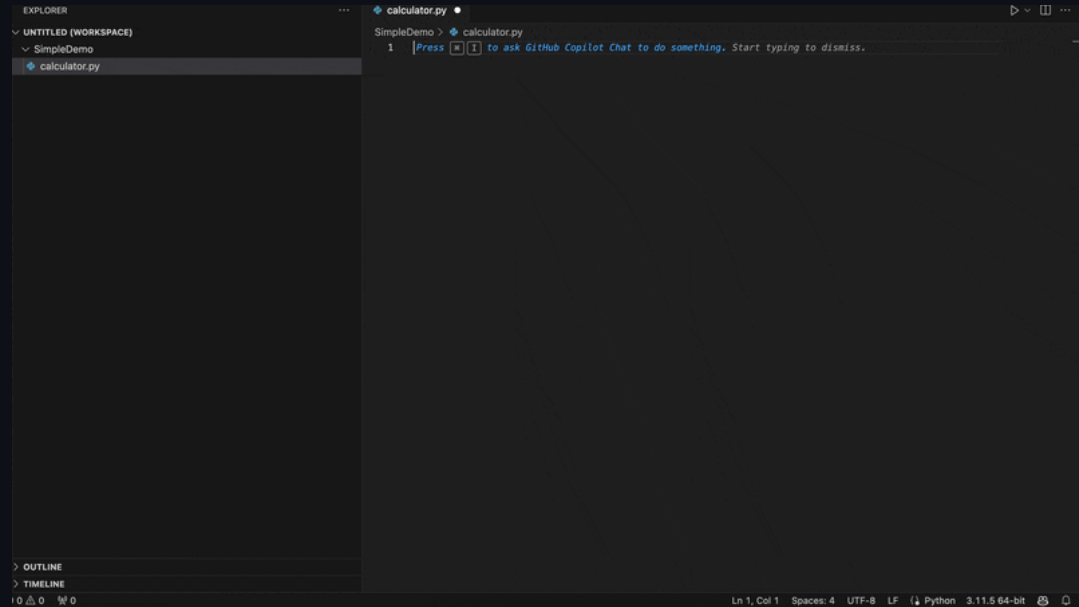
# Common Problems

# Problems #1:

Copilot fails to produce answer or keep repeating

## Some problems

- Fails to produce answer
- Hallucination - Keeps repeating



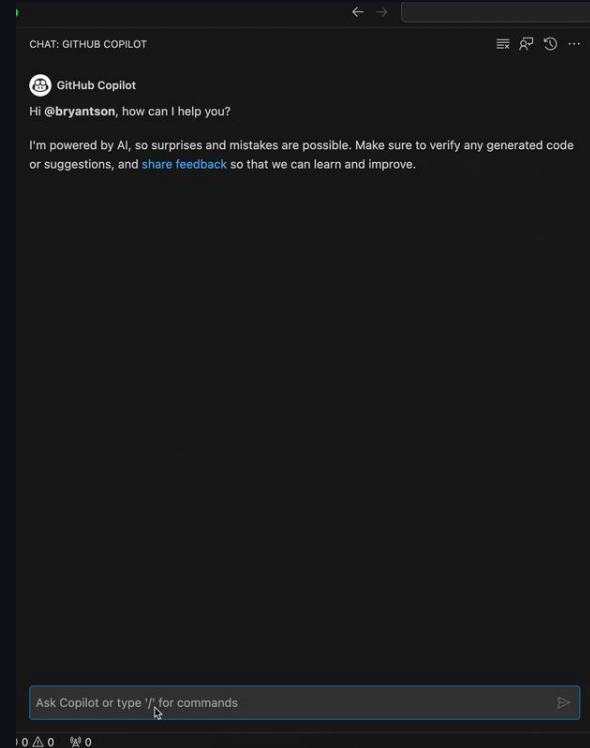


# Problems #2:

Copilot generates incorrect result

Reasons why...

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



# Problem #3:

Copilot suggests non-optimal solution

How to improve

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

A screenshot of a Visual Studio Code editor window. The title bar at the top says "SimpleDemo > TS computation.ts 2" and "quickSort". The editor is in dark mode. On the left sidebar, there are icons for Explorer, Search, Source Control, and Run and Debug. The main editor area shows a TypeScript file named "TS computation.ts 2" with the following code:

```
1 // Write a quick sort algorithm for me
2 function quickSort(arr) {
3   if (arr.length <= 1) {
4     return arr;
5   }
6   var pivot = arr[0];
7   var left = [];
8   var right = [];
9   for (var i = 1; i < arr.length; i++) {
10    if (arr[i] < pivot) {
11      left.push(arr[i]);
12    }
13    else {
14      right.push(arr[i]);
15    }
16  }
17  return quickSort(left).concat([pivot]).concat(quickSort(right));
18 }
```



# 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?”***



The background of the slide is a deep space image. It features a dark blue to black gradient, densely populated with stars of varying sizes and colors, including white, yellow, and blue. There are also nebulae and interstellar dust clouds visible, particularly on the right side where a bright, glowing blue and white nebula is prominent. The overall effect is a sense of vastness and cosmic wonder.

**Q & A**



Thank you