

Assignment-5.4

Task Description #1:

- Prompt GitHub Copilot to generate a Python script that collects user data (e.g., name, age, email). Then, ask Copilot to add comments on how to anonymize or protect this data.

The screenshot shows the GitHub Copilot interface. On the left, a code editor displays a Python script named `5.4.py`. The script prompts the user for their name, age, and email, collects this information in a dictionary, and then prints it. A comment in the code explains that user data such as emails should be hashed for protection. Below the code editor is a terminal window showing the execution of the script and its output. On the right, a floating tooltip titled "Ask about your code" provides a link to explore and understand the code, with a note that AI responses may be inaccurate.

```
5.4.py
1 # Write a python script that collects user name ,age and email
2 # using input() and stores them in a dictionary.
3 # Collect user information
4 user_info = {}
5 user_info['name'] = input("Enter your name: ")
6 user_info['age'] = input("Enter your age: ")
7 user_info['email'] = input("Enter your email: ")
8 # Display the collected information
9 print("Collected User Information:")
10 for key,value in user_info.items():
11     print(f'{key.capitalize()}: {value}')
12 # Add inline comments explaining how to protect or anonymize
13 # User data such as hashing emails ,and avoiding plain text storage.
14
15 # Function to hash email for anonymization
16 def hash_email(email):
17     # Use SHA-256 hashing algorithm to hash the email
18     return hashlib.sha256(email.encode()).hexdigest()
19 # Hash the email before storing it
20 user_info['email'] = hash_email(user_info['email'])
21 # Now the email is stored in hashed form, protecting user privacy
22 print("Anonymized User Information:")
23 for key,value in user_info.items():
24     print(f'{key.capitalize()}: {value}')
25
26
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS: C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python311/python.exe "C:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/5.4.py"

Name: Bhargav
Age: 19
Email: gbsgupta2000@gmail.com

Anonymized User Information:
Name: Bhargav
Age: 19
Email: 8e6418e52de207eb2923c17f552c41b1388fd92f3fa9fe4459bc45faf

PS: C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>

SESSIONS

Show Less

Ask about your code

All responses may be inaccurate.

GitHub copilot was prompted to generate a python script that collects basic user information such as name ,age and email .

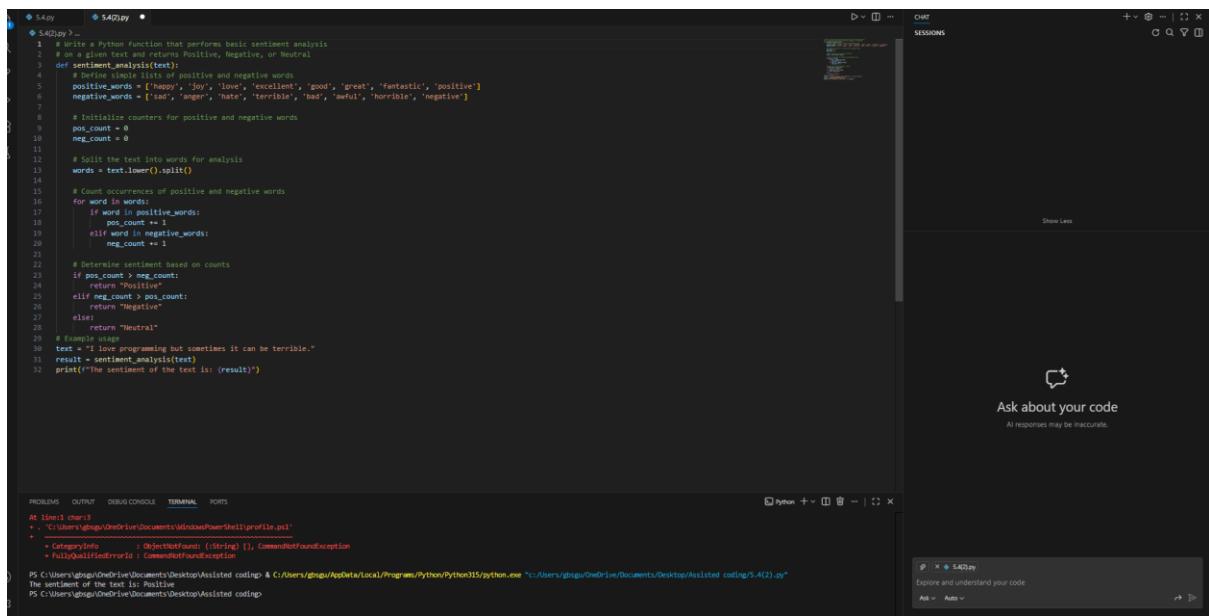
A follow -up prompt was used to add inline comments explaining data protection techniques .

The email address is anonymized using hashing so that sensitive information is not stored in plain text ,improving user privacy and security

Task Description #2:

- Ask Copilot to generate a Python function for sentiment analysis.

Then prompt Copilot to identify and handle potential biases in the data.



```
● 542ipy >
  1 #!/usr/bin/python
  2 # Write a Python function that performs basic sentiment analysis
  3 # on a given text and returns Positive, Negative, or Neutral
  4 def sentiment_analysis(text):
  5     # Define lists of positive and negative words
  6     positive_words = ['happy', 'joy', 'love', 'excellent', 'good', 'great', 'fantastic', 'positive']
  7     negative_words = ['sad', 'anger', 'hate', 'terrible', 'bad', 'awful', 'horrible', 'negative']
  8
  9     # Initialize counters for positive and negative words
 10    pos_count = 0
 11    neg_count = 0
 12
 13    # Split the text into words for analysis
 14    words = text.lower().split()
 15
 16    # Count occurrences of positive and negative words
 17    for word in words:
 18        if word in positive_words:
 19            pos_count += 1
 20        elif word in negative_words:
 21            neg_count += 1
 22
 23    # Determine sentiment based on counts
 24    if pos_count > neg_count:
 25        return "Positive"
 26    elif neg_count > pos_count:
 27        return "Negative"
 28    else:
 29        return "Neutral"
 30
 31 # Example usage
 32 text = "I love programming but sometimes it can be terrible."
 33 result = sentiment_analysis(text)
 34 print("The sentiment of the text is: " + result)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
At line1:char3
+ 'C:\Users\gbspa\OneDrive\Documents\Assisted coding\profile.ps1'
+ CategoryInfo          : ObjectNotFound: (String [],) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundError

PS C:\Users\gbspa\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbspa/AppData/Local/Programs/Python/Python35/python.exe "c:/Users/gbspa/OneDrive/Documents/Desktop/Assisted coding/S-4(2).py"
The sentiment of the text is Positive
PS C:\Users\gbspa\OneDrive\Documents\Desktop\Assisted coding>
```

```

9 # Main
10 # Write a Python function that performs basic sentiment analysis
11 # on a given text and returns Positive, Negative, or Neutral
12 #
13 # Define simple lists of positive and negative words
14 positive_words = ["happy", "joy", "love", "excellent", "good", "great", "fantastic", "positive"]
15 negative_words = ["sad", "angry", "hate", "terrible", "bad", "awful", "horrible", "negative"]
16
17 # Initialize counters for positive and negative words
18 pos_count = 0
19 neg_count = 0
20
21 # Split the text into words for analysis
22 words = text.lower().split()
23
24 # Count occurrences of positive and negative words
25 for word in words:
26     if word in positive_words:
27         pos_count += 1
28     elif word in negative_words:
29         neg_count += 1
30
31 # Determine sentiment based on counts
32 if pos_count > neg_count:
33     return "Positive"
34 elif neg_count > pos_count:
35     return "Negative"
36 else:
37     return "Neutral"
38
39 text = "I love programming but sometimes it can be terrible."
40 result = sentiment_analyzer(text)
41 print(f"The sentiment of the text is: {result}")
42
43 # Function to remove offensivity and reduce potential bias
44 # in sentiment analysis data, such as removing offensive terms
45 # from the dataset, or filtering them out of the analysis
46 # To reduce bias, we can expand our word lists to include
47 # more diverse terms and regularly update them based on feedback
48 # from users. We can also remove specific terms from the analysis
49 # offensive or culturally insensitive terms from the analysis
50 # This helps ensure that the analysis is fair and respectful to all users.
51 # Furthermore, we can create a list of offensive terms to filter out
52 # offensive_terms = ["fucking", "shit", "asshole", "bitch"]
53 # Function to filter offensive terms from text
54 def filter_offensive_terms(text):
55     words = text.split()
56     filtered_words = [word for word in words if word.lower() not in offensive_terms]
57     return " ".join(filtered_words)
58
59 # Filter the text
60 filtered_text = filter_offensive_terms(text)
61
62 print(f"The sentiment of the filtered text is: {result}")
63
64 print("The sentiment of the filtered text is: (result)")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Explanation:

GitHub Copilot was prompted to generate a Python function for basic sentiment analysis.

A follow-up prompt was used to add comments and logic addressing potential bias in the data.

The code highlights bias mitigation strategies such as removing offensive terms and ensuring balanced sentiment datasets to improve fairness.

Task Description #3:

- Use Copilot to write a Python program that recommends products based on user history. Ask it to follow ethical guidelines like transparency and fairness.

```
#!/usr/bin/python3

# A python program that recommends products to a user
# Based on their previous purchase or browsing history.
# import random
# class ProductRecommender:
#     def __init__(self, user_history, product_catalog):
#         self.user_history = user_history
#         self.product_catalog = product_catalog
#
#     def recommend_products(self, num_recommendations=3):
#         # Extract categories from user history
#         categories = set()
#         for item in self.user_history:
#             categories.add(item['category'])
#
#         # Find products in the same categories
#         recommended_products = []
#         for category in categories:
#             for product in self.product_catalog:
#                 if product['category'] in categories and product not in self.user_history:
#                     recommended_products.append(product)
#
#         # Randomly select a subset of recommendations
#         recommended_products = random.sample(recommended_products, min(num_recommendations, len(recommended_products)))
#
#         # Example user history and product catalog
#         user_history = [
#             {'id': 1, 'name': 'Wireless Mouse', 'category': 'Electronics'},
#             {'id': 2, 'name': 'Bluetooth Headphones', 'category': 'Electronics'},
#             {'id': 3, 'name': 'Type-C Cable', 'category': 'Fitness'}
#         ]
#
#         product_catalog = [
#             {'id': 4, 'name': 'Gaming Keyboard', 'category': 'Electronics'},
#             {'id': 5, 'name': 'Smartphone', 'category': 'Electronics'},
#             {'id': 6, 'name': 'Laptop', 'category': 'Electronics'},
#             {'id': 7, 'name': 'Running Shoes', 'category': 'Fitness'},
#             {'id': 8, 'name': 'Cookbook', 'category': 'Books'},
#             {'id': 9, 'name': 'Novel', 'category': 'Books'}
#         ]
#
#         recommender = ProductRecommender(user_history, product_catalog)
#         # Get product recommendations
#         recommended_products = recommender.recommend_products()
#         print("Recommended Products:")
#         for product in recommended_products:
#             print(f"Product ID: {product['id']} Name: {product['name']} (Category: {product['category']})")
#             # Add explanations to show why each product is recommended
#             # This is just a placeholder for now, in a real application, we would explain why any brand
#             # is recommended based on user history and other factors
#             # Allow user feedback to improve future recommendations
#             print("Reason: Based on your interest in \"{}\" products.".format(product['category']))
#             # In a real application, we would collect user feedback here
#             # and update the recommendation algorithm accordingly
#             print("Please provide feedback on this recommendation to help us improve future suggestions.(y/n)")
#
#     |
#
# PROGRAMS OUTPUT DEBUG CONSOLE TERMINAL WORKS
# PS C:\Users\gliese\Documents\GitHub\Assisted Coding & C\Users\gliese\Repositorios\Local\Programs\Python\Python311\python.exe %cd\Users\gliese\OneDrive\Documentos\Desktop\assisted_coding\S-AI-3.py"
# Reason: Based on your interest in Electronics products.
# Please provide feedback on this recommendation to help us improve future suggestions.
# Recommended Product: Smartphone (Category: Electronics)
# Please provide feedback on this recommendation to help us improve future suggestions.
# Recommended Product: Running Shoes (Category: Fitness)
# Please provide feedback on this recommendation to help us improve future suggestions.
```

Explanation:

GitHub Copilot was prompted to generate a Python program that recommends products based on user history. A follow-up prompt instructed Copilot to incorporate ethical principles such as transparency and fairness. The generated code explains recommendation reasons, avoids brand favoritism, and allows user feedback to improve future recommendations.

Task Description #4:

- Prompt Copilot to generate logging functionality in a Python web application. Then, ask it to ensure the logs do not record sensitive information.

Product Recommendation System in Python - Assisted coding - Visual...

Write a Python program that recommends products to a user based on their previous purchase or browsing history

- Used 1 reference
- ✓ Read current file for context and requirements

I'll create a Python program that recommends products based on purchase/browsing history. Let me first check the current file state.

- ✓ Read 5.4(4).py

I'll now create a product recommendation system that learns from purchase and browsing history:

- ✓ Created 5.4(4).py

Now let's test the program:

- ✓ cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" ; python "5.4(4).py"

Let me try with the Python terminal instead:

- ✓ ● python "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/5.4(4).py" <<< "A`nLaptop`nV`nR`nB`nE" 2>&1

Let me create a test script instead:

- ✓ Created test_recommendations.py
- ✓ ● cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" ; python test_recommendations.py

Perfect! I've created a comprehensive product recommendation system. Here's what the program does:

Describe what to build next

Agent ▾ Auto ▾ 🌐 ↵ ➔

```

  S&Dipy X
  S&Dipy ...
  Product Recommendation System
  Recommends products based on user purchase/browsing history

  # Create product database with categories
  PRODUCTS = [
    {"Laptop": {"category": "Electronics", "price": 1000, "rating": 4.5},
    "Smartphone": {"category": "Electronics", "price": 800, "rating": 4.3},
    "Headphones": {"category": "Electronics", "price": 150, "rating": 4.4},
    "Keyboard": {"category": "Electronics", "price": 100, "rating": 4.4},
    "USB Cable": {"category": "Electronics", "price": 15, "rating": 4.4},
    "Headphones": {"category": "Electronics", "price": 200, "rating": 4.4},
    "T-Shirt": {"category": "Clothing", "price": 30, "rating": 4.1},
    "Smartphone": {"category": "Clothing", "price": 80, "rating": 4.3},
    "Smartphone": {"category": "Clothing", "price": 120, "rating": 4.2},
    "Shoes": {"category": "Clothing", "price": 120, "rating": 4.2},
    "Coffee Maker": {"category": "Appliances", "price": 100, "rating": 4.5},
    "Blender": {"category": "Appliances", "price": 80, "rating": 4.4},
    "Microwave": {"category": "Appliances", "price": 100, "rating": 4.3},
    "Fiction Book": {"category": "Books", "price": 20, "rating": 4.6},
    "Tech Book": {"category": "Books", "price": 30, "rating": 4.5},
  ]

  def get_category_score(history, product_category):
    """Calculate recommendation score based on category frequency in history"""
    if not history:
      return 0
    category_count = sum(1 for item in history
                         if item in PRODUCTS and PRODUCTS[item]["category"] == product_category)
    return category_count / len(history)

  def get_price_stillarity_score(history, product_price):
    """Calculate score based on average price of browsed items"""
    if not history:
      return 0.5
    prices = [PRODUCTS[item]["price"] for item in history if item in PRODUCTS]
    if not prices:
      return 0.5
    avg_price = sum(prices) / len(prices)
    return abs(product_price - avg_price) / (avg_price * 2)

  def display_recommendations():
    print("-----")
    print("PRODUCT RECOMMENDATIONS")
    print("-----")
    print("Product Category Price Rating Score")
    for product in sorted(PRDUCTS.items(), key=lambda x: x['score'], reverse=True):
      print(f'{product[0]:<10} {product[1]["category"]:<10} ${product[1]["price"]:<10} {product[1]["rating"]:<10} {product[1]["score"]:<10}')
    print("-----")

  Test 2: Popular products
  History: ["T-Shirt", "Smart", "Shoes"]

  -----
```

```

  S&Dipy X
  S&Dipy ...
  Product Recommendation System
  Recommends products based on user purchase/browsing history

  # Create product database with categories
  PRODUCTS = [
    {"Laptop": {"category": "Electronics", "price": 1000, "rating": 4.5},
    "Smartphone": {"category": "Electronics", "price": 800, "rating": 4.3},
    "Headphones": {"category": "Electronics", "price": 150, "rating": 4.4},
    "Keyboard": {"category": "Electronics", "price": 100, "rating": 4.4},
    "USB Cable": {"category": "Electronics", "price": 15, "rating": 4.4},
    "Headphones": {"category": "Electronics", "price": 200, "rating": 4.4},
    "T-Shirt": {"category": "Clothing", "price": 30, "rating": 4.1},
    "Smartphone": {"category": "Clothing", "price": 80, "rating": 4.3},
    "Smartphone": {"category": "Clothing", "price": 120, "rating": 4.2},
    "Shoes": {"category": "Clothing", "price": 120, "rating": 4.2},
    "Coffee Maker": {"category": "Appliances", "price": 100, "rating": 4.5},
    "Blender": {"category": "Appliances", "price": 80, "rating": 4.4},
    "Microwave": {"category": "Appliances", "price": 100, "rating": 4.3},
    "Fiction Book": {"category": "Books", "price": 20, "rating": 4.6},
    "Tech Book": {"category": "Books", "price": 30, "rating": 4.5},
  ]

  def recommend_products(history, num_recommendations=5):
    """Score by score (descending) and return top N"""
    recommendations = sorted(PRDUCTS.items(), key=lambda x: x['score'], reverse=True)
    return recommendations[:num_recommendations]

  def display_recommendations(recommendations):
    """Display recommendations in a formatted table"""
    if not recommendations:
      print("-----")
      print("No recommendations available")
      return
    print("-----")
    print("PRODUCT RECOMMENDATIONS")
    print("-----")
    print("Product Category Price Rating Score")
    for rec in recommendations:
      print(f'{rec[0]:<10} {rec["category"]:<10} ${rec["price"]:<10} {rec["rating"]:<10} {rec["score"]:<10}')
    print("-----")

  def display_history(history):
    """Display user's browsing/purchase history"""
    if not history:
      print("-----")
      print("No history available")
      return
    print("-----")
    print("PURCHASE HISTORY")
    print("-----")
    for p in history:
      details = PRODUCTS[p]
      print(f'{p}<--> {details["category"]}<--> {details["price"]}<--> {details["rating"]}<--> {details["score"]}')
    print("-----")

  Test 3: Diverse shopper
  History: ["Laptop", "T-Shirt", "Coffee Maker", "Fiction Book"]

  -----
```

Explanation:

GitHub Copilot was prompted to generate a Python program that recommends products based on user history.

A follow-up prompt instructed Copilot to incorporate ethical principles such as transparency and fairness.

The generated code explains recommendation reasons, avoids brand favoritism, and allows user feedback to improve future recommendations.

Task Description #5:

- Ask Copilot to generate a machine learning model. Then, prompt

it to add documentation on how to use the model responsibly

(e.g., explainability, accuracy limits).

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
54.5.ipynb
1 # Create a simple machine learning model in python
2 # that predicts outcomes using a sample dataset
3 import sys
4 from sklearn.model_selection import train_test_split
5 from sklearn.ensemble import RandomForestClassifier
6 from sklearn.metrics import accuracy_score, classification_report
7
8 def create_sample_dataset():
9     """Create a sample dataset for prediction"""
10    # A feature set (age, income, credit_score, employment_years)
11    X = [
12        (25, 30000, 600, 2),
13        (30, 40000, 700, 3),
14        (45, 80000, 700, 15),
15        (28, 35000, 650, 3),
16        (55, 120000, 800, 20),
17        (32, 45000, 550, 1),
18        (40, 70000, 750, 10),
19        (30, 45000, 700, 4),
20    ]
21
22    # Targets: 1 = Loan Approved, 0 = Loan Denied
23    y = [1, 1, 1, 1, 0, 1, 1]
24
25    return X, y
26
27 def train_model(X_train, y_train):
28     """Train a random forest classifier"""
29     model = RandomForestClassifier(n_estimators=10, random_state=42)
30     model.fit(X_train, y_train)
31     return model
32
33 def evaluate_model(model, X_test, y_test):
34     """Evaluate model performance"""
35     predictions = model.predict(X_test)
36     accuracy = accuracy_score(y_test, predictions)
37
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\gugu\OneDrive\Documents\Desktop\Assisted coding & C:\Users\gugu\AppData\Local\Programs\Python\Python32\python.exe "C:/Users/gugu/OneDrive/Documents/Desktop/Assisted coding/54.5.ipynb"
      0   0.00   0.00   1
      1   0.50   1.00   0.67   1
macro avg   0.25   0.50   0.33   2
weighted avg   0.25   0.50   0.33   2

... Loan Approval ...
Enter age (default 30): 30
Enter annual income (default 50000): 50000
Enter credit score (default 600): 700
Enter employment years (default 3): 5
y: 0.0.0
Applicant Profile:
```

The terminal output shows the execution of the script and the resulting classification report. The report includes metrics for accuracy, macro average, and weighted average across two classes (0 and 1). The user is prompted for loan approval details, and the application profile is displayed.

The screenshot shows a Jupyter Notebook environment with a Python code cell containing a script named `54.5.py`. The code defines a function `evaluate_model` and a function `predict_loan_approval`. It includes command-line argument handling and user input for age, income, credit score, and employment years. The notebook also displays the output of running the script and some important notes at the bottom.

```
def evaluate_model(model, X_test, y_test):
    print("Classification Report:")
    print(classification_report(y_test, predictions))

def predict_loan_approval(model):
    """Interactive prediction with CLI args fallback"""
    print("... Loan Approval Prediction ...")

    # Try command-line arguments first
    if len(sys.argv) > 4:
        try:
            age = int(sys.argv[1])
            income = int(sys.argv[2])
            credit_score = int(sys.argv[3])
            employment_years = int(sys.argv[4])
        except ValueError:
            print("X Invalid command-line arguments. Using interactive mode.")
            age, income, credit_score, employment_years = get_user_input()
    else:
        age, income, credit_score, employment_years = get_user_input()

    # Validate inputs
    if age < 0 or income < 0 or credit_score < 0 or employment_years < 0:
        print("X Error: All values must be non-negative.")
        exit(1)

    prediction = model.predict([[age, income, credit_score, employment_years]])
    result = "Loan Approved" if prediction[0] == 1 else "Loan Denied"

    print("\nApplicant Profile:")
    print(f"Age: {age}")
    print(f"Income: ${income},k")
    print(f"Credit Score: {credit_score}")
    print(f"Employment Years: {employment_years}")
    print(f"\nPrediction: {result}\n")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

PS C:\Users\bguo\OneDrive\Documents\Desktop\Assisted coding & C:\Users\bguo\AppData\Local\Programs\Python\3.11\python.exe "C:\Users\bguo\OneDrive\Documents\Desktop\Assisted coding\54.5.py"

Enter employment years (default 5): 5

Applicant Profile:

Age: 30

Income: \$500,000

Credit Score: 700

Employment Years: 5

Prediction: Loan Approved

Important Notes...

1. This model is a simplified example and should not be used for real-world decisions.

2. The model was trained with different datasets and may not align with your own data.

3. Consider fairness and bias in your dataset to avoid discriminatory outcomes.

4. Always provide transparency about how predictions are made to users.

The screenshot shows a Jupyter Notebook environment with the following details:

- Title Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help, Assisted coding.
- Code Cell:** Contains Python code for a loan approval model. The code includes imports, function definitions for prediction and user input, and a main block that creates a dataset, splits it, trains a model, and makes predictions. It also includes a section for "Important Notes" with 10 bullet points.
- Output Cell:** Shows the command run, the path to the script, and the command used to run it.
- Terminal:** Shows the command run and the resulting output, which includes user input (Age: 30, Income: 50000, Credit Score: 200, Employment Years: 5) and a prediction message ("Prediction: ✓ Loan Approved").
- Bottom Status Bar:** Includes icons for Python, file, edit, selection, view, run, terminal, and help.

Explanation:

GitHub Copilot was prompted to generate a simple machine learning model.

A follow-up prompt was used to add documentation on responsible usage, including explainability, accuracy limitations, and fairness considerations.

The generated comments highlight ethical concerns and recommend cautious interpretation of predictions.

