

Section 1: Setup & Imports

Install and import libraries for semantic analysis, visualization, parsing, and code generation.

```
# Install dependencies
!pip install sentence-transformers nltk textstat pandas matplotlib wordcloud --quiet

# Imports
import nltk
import re
from sentence_transformers import SentenceTransformer
import pandas as pd
import ast
import tokenize
import io
from collections import Counter
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

# Download NLTK data, including punkt_tab
try:
    nltk.download('punkt_tab', quiet=True)
    nltk.download('stopwords', quiet=True)
except Exception as e:
    print(f"Error downloading NLTK data: {e}")
print("[✓] Setup complete")
```

239.2/239.2 kB 7.1 MB/s eta 0:00:00

2.1/2.1 MB 26.7 MB/s eta 0:00:00

```
[✓] Setup complete
```

Section 2: Example Code Snippets

```
code_snippets = [
    """# 1. Add two numbers
number1 = input("First number: ")
number2 = input("Second number: ")
sum = float(number1) + float(number2)
print("The sum of {0} and {1} is {2}".format(number1, number2, sum))
""",
    """# 2. Factorial of a number
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)

num = 5
print("Factorial of", num, "is", factorial(num))
""",
    """# 3. Check if number is prime
num = 11
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            print(num, "is not a prime number")
            break
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")
""",
    """# 4. Find area of a circle
radius = 6
pi = 3.14159
area = pi * radius * radius
print("Area of circle is:", area)
""",
    """# 5. Fibonacci sequence
def fibonacci(n):
    if n <= 0:
        print("Incorrect input")
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(9))
""",
    """# 6. Reverse a string
```



```
text = input("Enter a string: ")
reversed_text = text[::-1]
print("Reversed string:", reversed_text)
"""
"""# 7. Count vowels in a string
def count_vowels(s):
    vowels = 'aeiouAEIOU'
    count = sum(1 for char in s if char in vowels)
    return count
text = input("Enter a string: ")
print("Number of vowels:", count_vowels(text))
"""
"""# 8. Sum of list elements
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)
print("Sum of list:", total)
"""

]

snippet_names = [f"Program_{i+1}" for i in range(len(code_snippets))]

print("Loaded 8 example code snippets:")
for name, text in zip(snippet_names, code_snippets):
    print(f"{name}: {len(text)} characters")
```

Loaded 8 example code snippets:
Program_1: 197 characters
Program_2: 188 characters
Program_3: 268 characters
Program_4: 113 characters
Program_5: 235 characters
Program_6: 123 characters
Program_7: 225 characters
Program_8: 102 characters

Section 3: Multi-Model Code Explainer Module

Use three pretrained SentenceTransformer models to generate context-aware explanations. Compare explanations across models.

Task

Implement a Python interface `codeexplainerinterface` with methods `explain_code` and `compare_explanations`, and modify the existing `codeexplainer` class to implement this interface.

Define the `codeexplainerinterface`

Subtask:

Create an abstract base class (ABC) in Python to define the interface with the required methods (`explain_code` and `compare_explanations`).

Reasoning: Define the abstract base class `CodeExplainerInterface` with the specified abstract methods as requested by the subtask.

```
from abc import ABC, abstractmethod

class CodeExplainerInterface(ABC):
    @abstractmethod
    def explain_code(self, code_text, model_name):
        """
        Abstract method to explain code using a specific model.

        Args:
            code_text: The code snippet as a string.
            model_name: The name of the model to use for explanation.

        Returns:
            A dictionary of explanations, where keys are line numbers and values are explanations.
        """
        pass

    @abstractmethod
    def compare_explanations(self, code_text, snippet_name):
        """
        Abstract method to compare explanations from multiple models for a code snippet.

        Args:
```

```
code_text: The code snippet as a string.
snippet_name: The name of the code snippet.

Returns:
    A pandas DataFrame comparing explanations from different models.
"""
pass
```

▼ Modify `codeexplainer` to implement the interface

Subtask:

Update the existing `CodeExplainer` class to inherit from the `CodeExplainerInterface` and ensure it implements all the required methods.

Reasoning: Modify the existing CodeExplainer class to inherit from CodeExplainerInterface and keep the existing methods.

Reasoning: The previous code block failed because SentenceTransformer was not defined. Need to include the import statement for SentenceTransformer.

```
from sentence_transformers import SentenceTransformer

class CodeExplainer(CodeExplainerInterface):
    def __init__(self):
        self.models = {
            'MiniLM': SentenceTransformer('all-MiniLM-L12-v2'),
            'DistilRoBERTa': SentenceTransformer('all-distilroberta-v1'),
            'MPNet': SentenceTransformer('all-mpnet-base-v2')
        }
        self.explanation_templates = {
            'def': [
                "Defines function '{}'' with parameters", # MiniLM
                "Creates a function '{}'' for computation", # DistilRoBERTa
                "Sets up function '{}'' to perform a task" # MPNet
            ],
            'if': [
                "Evaluates condition: '{}'",
                "Checks if: '{}'",
                "Tests condition: '{}'"
            ],
            'elif': [
                "Evaluates additional condition: '{}'",
                "Checks another condition: '{}'",
                "Tests further condition: '{}'"
            ],
            'else': [
                "Handles alternative case when conditions fail",
                "Executes if prior conditions are false",
                "Catches remaining cases"
            ],
            'for': [
                "Loops over sequence: '{}'",
                "Iterates through: '{}'",
                "Performs iteration: '{}'"
            ],
            'while': [
                "Loops while: '{}'",
                "Continues looping if: '{}'",
                "Repeats until condition changes: '{}'"
            ],
            'assign': [
                "Assigns value to '{}'",
                "Sets variable '{}' to a value",
                "Stores result in '{}'"
            ],
            'print': [
                "Outputs: '{}'",
                "Displays to console: '{}'",
                "Prints result: '{}'"
            ],
            'return': [
                "Returns: '{}'",
                "Yields value: '{}'",
                "Provides output: '{}'"
            ],
            'default': [
                "Executes: '{}'",
                "Performs operation: '{}'",
                "Runs code: '{}'"
            ]
        }
```

```

    }

def explain_line(self, line, model_name):
    line = line.strip()
    if not line or line.startswith('#'):
        return None
    # Mock semantic analysis: Use model index to select explanation variant
    model_index = {'MiniLM': 0, 'DistilRoBERTa': 1, 'MPNet': 2}
    idx = model_index[model_name]
    # Compute embedding for context (mock usage)
    self.models[model_name].encode([line])[0]
    if line.startswith('def '):
        return self.explanation_templates['def'][idx].format(line.split('(')[0].replace('def ', ''))
    elif line.startswith('if '):
        return self.explanation_templates['if'][idx].format(line)
    elif line.startswith('elif '):
        return self.explanation_templates['elif'][idx].format(line)
    elif line.startswith('else:'):
        return self.explanation_templates['else'][idx]
    elif line.startswith('for '):
        return self.explanation_templates['for'][idx].format(line)
    elif line.startswith('while '):
        return self.explanation_templates['while'][idx].format(line)
    elif '=' in line and not line.startswith('print'):
        return self.explanation_templates['assign'][idx].format(line.split('=')[0].strip())
    elif line.startswith('print'):
        return self.explanation_templates['print'][idx].format(line)
    elif line.startswith('return'):
        return self.explanation_templates['return'][idx].format(line)
    return self.explanation_templates['default'][idx].format(line)

def explain_code(self, code_text, model_name):
    explanations = {}
    for i, line in enumerate(code_text.splitlines(), 1):
        explanation = self.explain_line(line, model_name)
        if explanation:
            explanations[i] = explanation
    return explanations

def compare_explanations(self, code_text, snippet_name):
    # Generate explanations for the code snippet using all three models
    explanations = {
        model_name: self.explain_code(code_text, model_name)
        for model_name in self.models.keys()
    }
    # Create a DataFrame to compare explanations line by line
    comparison_data = []
    for line_num in sorted(set(sum([list(exp.keys()) for exp in explanations.values()], []))):
        row = {'Line Number': line_num, 'Code': code_text.splitlines()[line_num-1].strip()}
        for model_name in self.models.keys():
            row[model_name] = explanations[model_name].get(line_num, "No explanation")
        comparison_data.append(row)
    return pd.DataFrame(comparison_data)

explainer = CodeExplainer()

# Generate and compare explanations for all snippets
comparisons = []
for name, text in zip(snippet_names, code_snippets):
    comparison_df = explainer.compare_explanations(text, name)
    comparisons.append((name, comparison_df))

# Print comparison for Program 1
print("\nExplanation Comparison for Program_1:")
display(comparisons[0][1])

# Print summary of differences for all snippets
print("\nSummary of Explanation Differences:")
for name, df in comparisons:
    print(f"\n{name}:")
    differing_lines = df[df['MiniLM'] != df['DistilRoBERTa']][['Line Number', 'Code', 'MiniLM', 'DistilRoBERTa', 'MPNet']]
    if not differing_lines.empty:
        print("Lines with differing explanations:")
        display(differing_lines)
    else:
        print("All models provided identical explanations.")

```


/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.

```
warnings.warn(  
modules.json: 100%                               349/349 [00:00<00:00, 27.0kB/s]  
  
config_sentence_transformers.json: 100%          116/116 [00:00<00:00, 9.54kB/s]  
  
README.md:      10.5k/? [00:00<00:00, 489kB/s]  
  
sentence_bert_config.json: 100%                  53.0/53.0 [00:00<00:00, 5.97kB/s]  
  
config.json: 100%                               615/615 [00:00<00:00, 57.4kB/s]  
  
model.safetensors: 100%                         133M/133M [00:03<00:00, 73.9MB/s]  
  
tokenizer_config.json: 100%                     352/352 [00:00<00:00, 6.78kB/s]  
  
vocab.txt:      232k/? [00:00<00:00, 2.41MB/s]  
  
tokenizer.json:   466k/? [00:00<00:00, 6.36MB/s]  
  
special_tokens_map.json: 100%                   112/112 [00:00<00:00, 1.79kB/s]  
  
config.json: 100%                              190/190 [00:00<00:00, 5.05kB/s]  
  
modules.json: 100%                             349/349 [00:00<00:00, 7.39kB/s]  
  
config_sentence_transformers.json: 100%          116/116 [00:00<00:00, 2.71kB/s]  
  
README.md:      10.1k/? [00:00<00:00, 211kB/s]  
  
sentence_bert_config.json: 100%                  53.0/53.0 [00:00<00:00, 1.42kB/s]  
  
config.json: 100%                              653/653 [00:00<00:00, 13.2kB/s]  
  
model.safetensors: 100%                         328M/328M [00:04<00:00, 92.0MB/s]  
  
tokenizer_config.json: 100%                     333/333 [00:00<00:00, 4.35kB/s]  
  
vocab.json:      798k/? [00:00<00:00, 12.1MB/s]  
  
merges.txt:      456k/? [00:00<00:00, 4.43MB/s]  
  
tokenizer.json:   1.36M/? [00:00<00:00, 27.5MB/s]  
  
special_tokens_map.json: 100%                   239/239 [00:00<00:00, 6.02kB/s]  
  
config.json: 100%                              190/190 [00:00<00:00, 3.90kB/s]  
  
modules.json: 100%                             349/349 [00:00<00:00, 8.66kB/s]  
  
config_sentence_transformers.json: 100%          116/116 [00:00<00:00, 2.81kB/s]  
  
README.md:      11.6k/? [00:00<00:00, 239kB/s]  
  
sentence_bert_config.json: 100%                  53.0/53.0 [00:00<00:00, 2.27kB/s]  
  
config.json: 100%                              571/571 [00:00<00:00, 12.3kB/s]  
  
model.safetensors: 100%                         438M/438M [00:06<00:00, 108MB/s]  
  
tokenizer_config.json: 100%                     363/363 [00:00<00:00, 27.6kB/s]  
  
vocab.txt:      232k/? [00:00<00:00, 2.34MB/s]  
  
tokenizer.json:   466k/? [00:00<00:00, 4.16MB/s]  
  
special_tokens_map.json: 100%                   239/239 [00:00<00:00, 26.6kB/s]  
  
config.json: 100%                              190/190 [00:00<00:00, 15.8kB/s]
```

Explanation Comparison for Program_1:

Line Number		Code	MiniLM	DistilRoBERTa	MPNet
0	2	number1 = input("First number: ")	Assigns value to 'number1'	Sets variable 'number1' to a value	Stores result in 'number1'
1	3	number2 = input("Second number: ")	Assigns value to 'number2'	Sets variable 'number2' to a value	Stores result in 'number2'
2	4	sum = float(number1) + float(number2)	Assigns value to 'sum'	Sets variable 'sum' to a value	Stores result in 'sum'
3	5	print("The sum of {0} and {1} is {2}".format(n...	Outputs: 'print("The sum of {0} and {1} is {2}...	Displays to console: 'print("The sum of {0} an...	Prints result: 'print("The sum of {0} and {1} ...



Summary of Explanation Differences:

Program_1:



Lines with differing explanations:

Line Number		Code	MiniLM	DistilRoBERTa	MPNet
0	2	number1 = input("First number: ")	Assigns value to 'number1'	Sets variable 'number1' to a value	Stores result in 'number1'
1	3	number2 = input("Second number: ")	Assigns value to 'number2'	Sets variable 'number2' to a value	Stores result in 'number2'
2	4	sum = float(number1) + float(number2)	Assigns value to 'sum'	Sets variable 'sum' to a value	Stores result in 'sum'
3	5	print("The sum of {0} and {1} is {2}".format(n...	Outputs: 'print("The sum of {0} and {1} is {2}...	Displays to console: 'print("The sum of {0} an...	Prints result: 'print("The sum of {0} and {1} ...





Program_2:
Lines with differing explanations:

Line Number		Code	MiniLM	DistilRoBERTa	MPNet	
0	2	def factorial(n):	Defines function 'factorial' with parameters	Creates a function 'factorial' for computation	Sets up function 'factorial' to perform a task	
1	3	if n == 0 or n == 1:	Evaluates condition: 'if n == 0 or n == 1:'	Checks if: 'if n == 0 or n == 1:'	Tests condition: 'if n == 0 or n == 1:'	
2	4	return 1	Returns: 'return 1'	Yields value: 'return 1'	Provides output: 'return 1'	
3	5	else:	Handles alternative case when conditions fail	Executes if prior conditions are false	Catches remaining cases	
4	6	return n * factorial(n-1)	Returns: 'return n * factorial(n-1)'	Yields value: 'return n * factorial(n-1)'	Provides output: 'return n * factorial(n-1)'	
5	7	num = 5	Assigns value to 'num'	Sets variable 'num' to a value	Stores result in 'num'	
6	8	print("Factorial of", num, "is", factorial(num))	Outputs: 'print("Factorial of", num, "is", fac...	Displays to console: 'print("Factorial of", nu...	Prints result: 'print("Factorial of", num, "is...	



Program_3:
Lines with differing explanations:

Line	Number	Code	MiniLM	DistilRoBERTa	MPNet	
0	2	num = 11	Assigns value to 'num'	Sets variable 'num' to a value	Stores result in 'num'	
1	3	if num > 1:	Evaluates condition: 'if num > 1:'	Checks if: 'if num > 1:'	Tests condition: 'if num > 1:'	
2	4	for i in range(2, num):	Loops over sequence: 'for i in range(2, num):'	Iterates through: 'for i in range(2, num):'	Performs iteration: 'for i in range(2, num):'	
3	5	if (num % i) == 0:	Evaluates condition: 'if (num % i) == 0:'	Checks if: 'if (num % i) == 0:'	Tests condition: 'if (num % i) == 0:'	
4	6	print(num, "is not a prime number")	Outputs: 'print(num, "is not a prime number")'	Displays to console: 'print(num, "is not a pri...	Prints result: 'print(num, "is not a prime num...	
5	7	break	Executes: 'break'	Performs operation: 'break'	Runs code: 'break'	
6	8	else:	Handles alternative case when conditions fail	Executes if prior conditions are false	Catches remaining cases	
7	9	print(num, "is a prime number")	Outputs: 'print(num, "is a prime number")'	Displays to console: 'print(num, "is a prime n...	Prints result: 'print(num, "is a prime number")'	
8	10	else:	Handles alternative case when conditions fail	Executes if prior conditions are false	Catches remaining cases	
9	11	print(num, "is not a prime number")	Outputs: 'print(num, "is not a prime number")'	Displays to console: 'print(num, "is not a pri...	Prints result: 'print(num, "is not a prime num...	

```
Program_4:
Lines with differing explanations:
```

Line	Number	Code	MiniLM	DistilRoBERTa	MPNet	
0	2	radius = 6	Assigns value to 'radius'	Sets variable 'radius' to a value	Stores result in 'radius'	
1	3	pi = 3.14159	Assigns value to 'pi'	Sets variable 'pi' to a value	Stores result in 'pi'	
2	4	area = pi * radius * radius	Assigns value to 'area'	Sets variable 'area' to a value	Stores result in 'area'	
3	5	print("Area of circle is:", area)	Outputs: 'print("Area of circle is:", area)'	Displays to console: 'print("Area of circle is:...	Prints result: 'print("Area of circle is:", ar...	

Program_5:
Lines with differing explanations:

Line	Number	Code	MiniLM	DistilRoBERTa	MPNet	
0	2	def fibonacci(n):	Defines function 'fibonacci' with parameters	Creates a function 'fibonacci' for computation	Sets up function 'fibonacci' to perform a task	
1	3	if n <= 0:	Evaluates condition: 'if n <= 0:'	Checks if: 'if n <= 0:'	Tests condition: 'if n <= 0:'	
2	4	print("Incorrect input")	Outputs: 'print("Incorrect input")'	Displays to console: 'print("Incorrect input")'	Prints result: 'print("Incorrect input")'	
3	5	elif n == 1:	Evaluates additional condition: 'elif n == 1:'	Checks another condition: 'elif n == 1:'	Tests further condition: 'elif n == 1:'	
4	6	return 0	Returns: 'return 0'	Yields value: 'return 0'	Provides output: 'return 0'	
5	7	elif n == 2:	Evaluates additional condition: 'elif n == 2:'	Checks another condition: 'elif n == 2:'	Tests further condition: 'elif n == 2:'	
6	8	return 1	Returns: 'return 1'	Yields value: 'return 1'	Provides output: 'return 1'	
7	9	else:	Handles alternative case when conditions fail	Executes if prior conditions are false	Catches remaining cases	
8	10	return fibonacci(n-1) + fibonacci(n-2)	Returns: 'return fibonacci(n-1) + fibonacci(n-2)'	Yields value: 'return fibonacci(n-1) + fibonac...	Provides output: 'return fibonacci(n-1) + fibo...	
9	11	print(fibonacci(9))	Outputs: 'print(fibonacci(9))'	Displays to console: 'print(fibonacci(9))'	Prints result: 'print(fibonacci(9))'	

```
Program_6:
Lines with differing explanations:
```

Line	Number	Code	MiniLM	DistilRoBERTa	MPNet
0	2	text = input("Enter a string: ")	Assigns value to 'text'	Sets variable 'text' to a value	Stores result in 'text'
1	3	reversed_text = text[::-1]	Assigns value to 'reversed_text'	Sets variable 'reversed_text' to a value	Stores result in 'reversed_text'

Next steps: 2. `Generate code` `print("Reversed string")` reversed `display_text` `Output display` `print("Reversed string")` `with text` `Display` to console `print("Reversed string")` `Generate` `Prints result` `print("Reversed string")`

Program_7:
Lines with differing explanations:

Line Number		Code	MiniLM	DistilRoBERTa	MPNet
0	2	def count_vowels(s):	Defines function 'count_vowels' with parameters	Creates a function 'count_vowels' for computation	Sets up function 'count_vowels' to perform a task
1	3	vowels = 'aeiouAEIOU'	Assigns value to 'vowels'	Sets variable 'vowels' to a value	Stores result in 'vowels'

Reasoning: The previous code block failed because `snippet_names` and `code_snippets` were not defined in the current cell. I need to include the definition of these variables from the previous relevant cell.

Reasoning: The previous code block failed because `pd` was not defined. I need to include the import statement for pandas.

5	7 print("Number of vowels:", count_vowels(text))	Outputs: 'print("Number of vowels:", count_vow...	Displays to console: 'print("Number of vowels:...	Prints result: 'print("Number of vowels:", cou....
---	--	---	---	--

```
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=differing_lines)
```

[illegible]

Demonstrate the interface usage

Subtask:

Write a code snippet that shows how to use the `CodeExplainerInterface` to interact with the `CodeExplainer` class.

Reasoning: Instantiate a CodeExplainer object, define a sample code snippet, and use the explain_code and compare_explanations methods as specified in the instructions.

```
# 1. Instantiate an object of the CodeExplainer class.
explainer = CodeExplainer()

# 2. Define a sample code snippet string.
sample_code = """
def greet(name):
    print(f"Hello, {name}!")

greet("World")
"""

# 3. Use the explain_code method and print the explanations.
print("Explanation using MiniLM:")
explanations_minilm = explainer.explain_code(sample_code, 'MiniLM')
for line_num, explanation in explanations_minilm.items():
    print(f"Line {line_num}: {explanation}")

# 4. Use the compare_explanations method and display the DataFrame.
print("\nComparison of Explanations for Sample Program:")
comparison_df = explainer.compare_explanations(sample_code, 'Sample Program')
display(comparison_df)
```


Comparison of Explanations for Sample Program:

```
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=comparison_df)
```

File Edit View Insert Format Data Tools Extensions Help

^

[illegible]