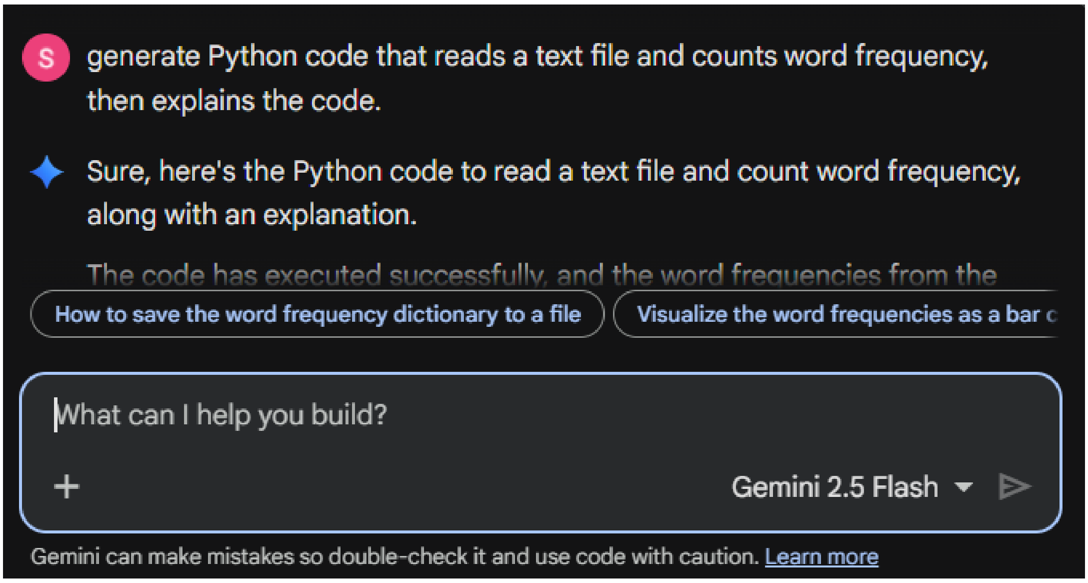
ASSIGNMENT 2.3

HT NO:2303A51556

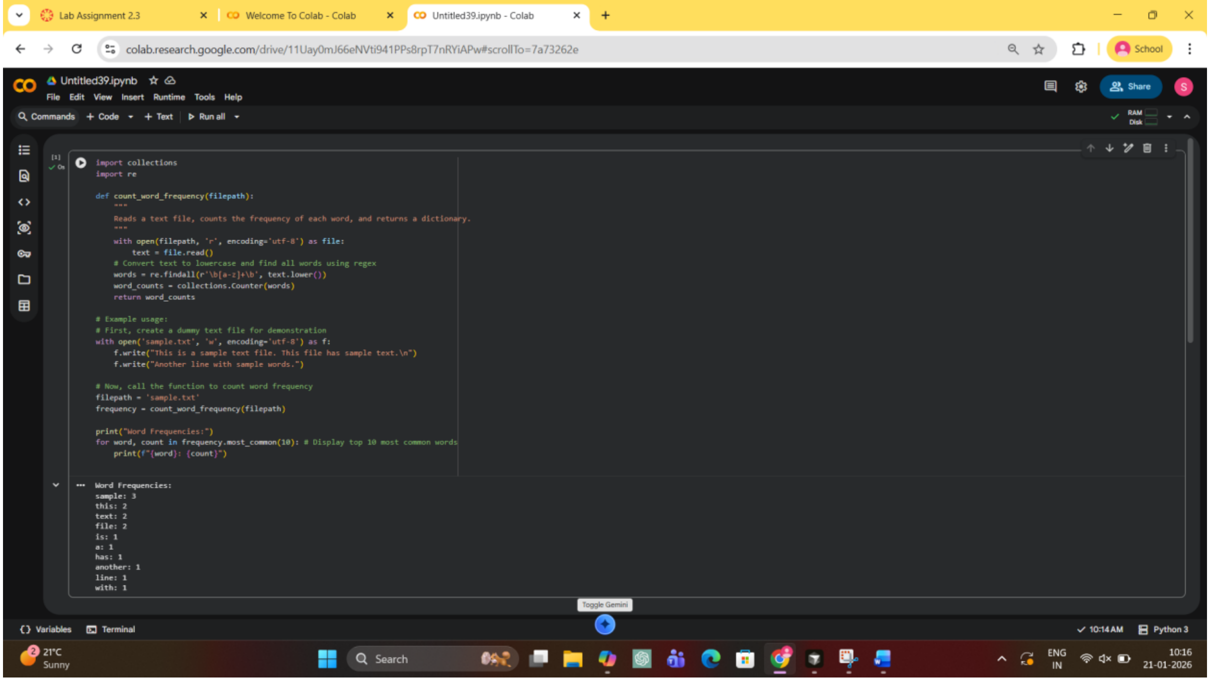


generate Python code that reads a text file and counts word frequency, then explains the code.

Sure, here's the Python code to read a text file and count word frequency, along with an explanation.

The code has executed successfully, and the word frequencies from the

How to save the word frequency dictionary to a file
Visualize the word frequencies as a bar c
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more



Lab Assignment 2.3
Welcome To Colab - Colab
Untitled39.ipynb - Colab
colab.research.google.com/drive/11Uay0mJ66eNVti941PPs8rpT7nRYiAPw#scrollTo=7a73262e
Untitled39.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share

```python
import collections
import re

def count_word_frequency(filepath):
    """
    Reads a text file, counts the frequency of each word, and returns a dictionary.
    """
    with open(filepath, 'r', encoding='utf-8') as file:
        text = file.read()
    # Convert text to lowercase and find all words using regex
    words = re.findall(r'\b[a-z]+\b', text.lower())
    word_counts = collections.Counter(words)
    return word_counts

# Example usage:
# First, create a dummy text file for demonstration
with open('sample.txt', 'w', encoding='utf-8') as f:
    f.write("This is a sample text file. This file has sample text.\n")
    f.write("Another line with sample words.")

# Now, call the function to count word frequency
filepath = 'sample.txt'
frequency = count_word_frequency(filepath)

print("Word Frequencies:")
for word, count in frequency.most_common(10): # Display top 10 most common words
    print(f"{word}: {count}")
```

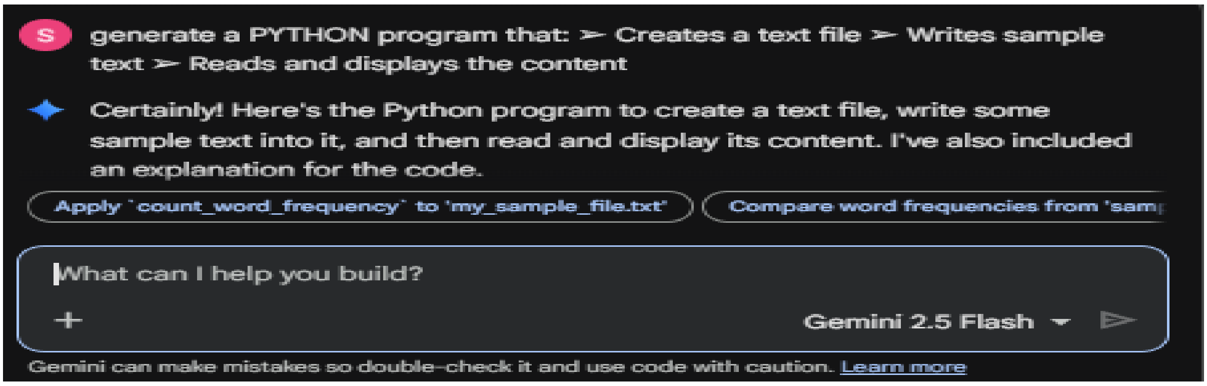
```
Word Frequencies:
sample: 3
this: 2
text: 2
file: 2
is: 1
a: 1
has: 1
another: 1
line: 1
with: 1
```

Code Explanation:

This code defines a function count_word_frequency that takes a file path as input and returns a dictionary containing the frequency of each word in the file.

1. **Import collections and re**:
    - collections.Counter is used to easily count hashable objects.
    - re (regular expressions) is used to find words in the text.
2. **count_word_frequency(filepath) function**:
    - It opens the specified filepath in read mode ('r') with UTF-8 encoding.

- Reads the entire content of the file into the text variable.
- text.lower(): Converts all characters in the text to lowercase to ensure that words like "The" and "the" are counted as the same word.
- re.findall(r'\b[a-z]+\b', ...): Uses a regular expression to find all sequences of one or more lowercase English letters ([a-z]+) that are surrounded by word boundaries (\b). This extracts individual words and ignores punctuation and numbers.
- collections.Counter(words): Creates a Counter object from the list of extracted words. The Counter automatically tallies the occurrences of each unique word.
- Returns the word_counts dictionary.

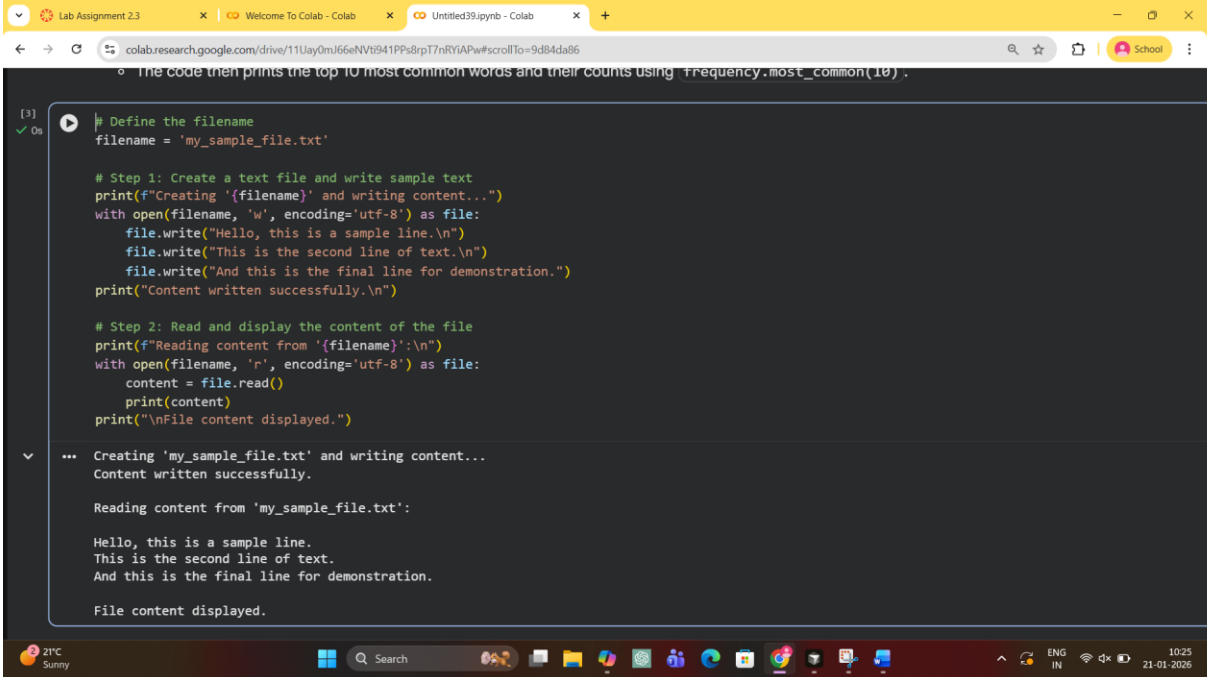


generate a PYTHON program that: ➢ Creates a text file ➢ Writes sample text ➢ Reads and displays the content

Certainly! Here's the Python program to create a text file, write some sample text into it, and then read and display its content. I've also included an explanation for the code.

Apply `count_word_frequency` to 'my_sample_file.txt'
Compare word frequencies from 'sam
What can I help you build?
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more



The code then prints the top 10 most common words and their counts using `frequency.most_common(10)`.

```
# Define the filename
filename = 'my_sample_file.txt'

# Step 1: Create a text file and write sample text
print(f"Creating '{filename}' and writing content...")
with open(filename, 'w', encoding='utf-8') as file:
    file.write("Hello, this is a sample line.\n")
    file.write("This is the second line of text.\n")
    file.write("And this is the final line for demonstration.")
print("Content written successfully.\n")

# Step 2: Read and display the content of the file
print(f"Reading content from '{filename}':\n")
with open(filename, 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
print("\nFile content displayed.")
```

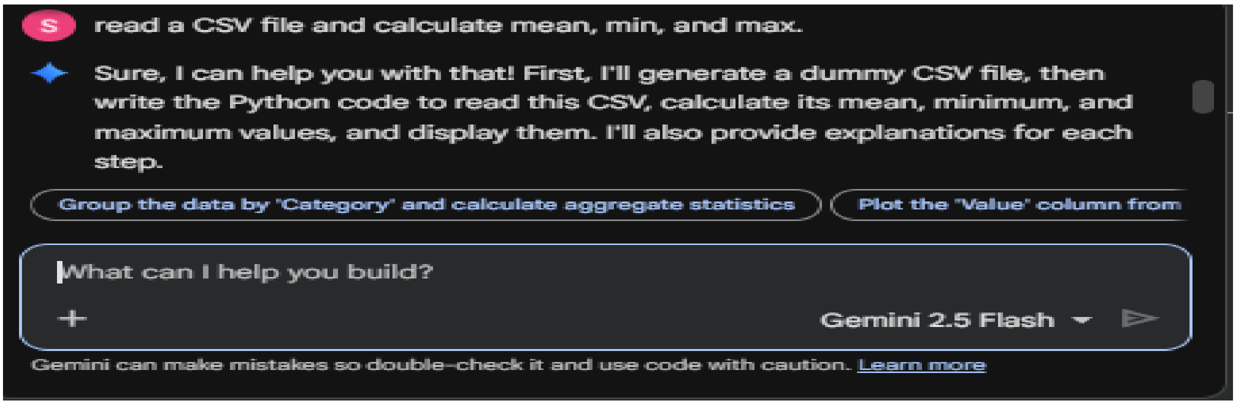
```
Creating 'my_sample_file.txt' and writing content...
Content written successfully.

Reading content from 'my_sample_file.txt':

Hello, this is a sample line.
This is the second line of text.
And this is the final line for demonstration.

File content displayed.
```

Code Explanation:

This program demonstrates basic file I/O operations in Python:

1. **filename = 'my_sample_file.txt'**: Defines the name of the text file that will be created and manipulated.

2. **Writing to the file**:
   - with open(filename, 'w', encoding='utf-8') as file:: This line opens the file named my_sample_file.txt in **write mode ('w')**. If the file doesn't exist, it will be created. If it does exist, its content will be truncated (emptied) before writing. encoding='utf-8' ensures proper handling of various characters.
   - The with statement ensures that the file is automatically closed even if errors occur.
   - file.write(...): These lines write the specified string content into the file. The \n character is used to add a new line.
3. **Reading from the file**:
   - with open(filename, 'r', encoding='utf-8') as file:: This line opens the same file in **read mode ('r')**.
   - content = file.read(): Reads the entire content of the file and stores it as a single string in the content variable.
   - print(content): Displays the read content to the console.

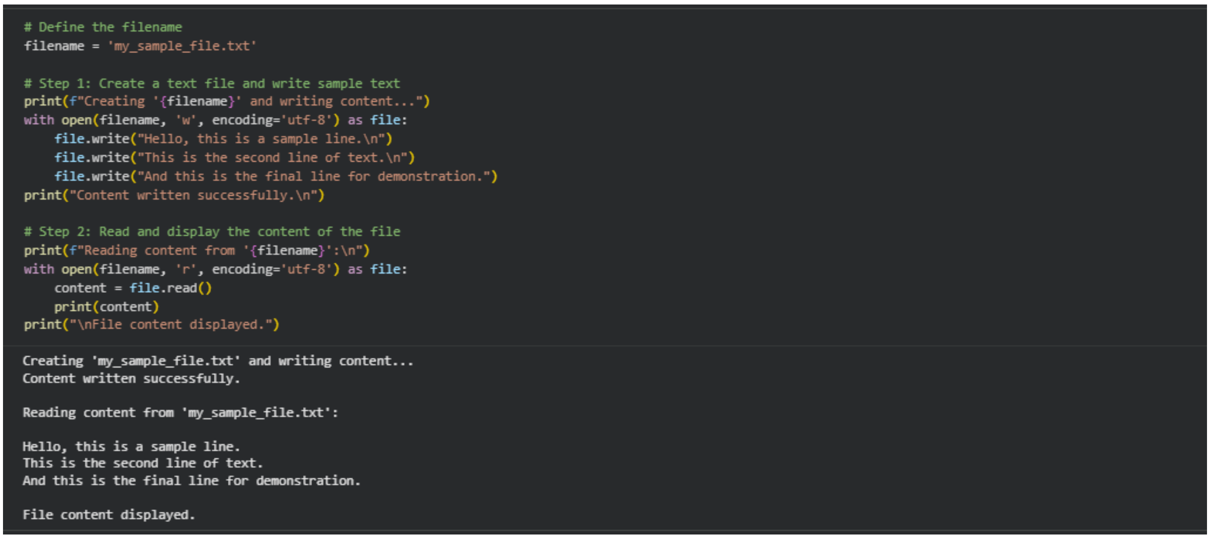


read a CSV file and calculate mean, min, and max.

Sure, I can help you with that! First, I'll generate a dummy CSV file, then write the Python code to read this CSV, calculate its mean, minimum, and maximum values, and display them. I'll also provide explanations for each step.

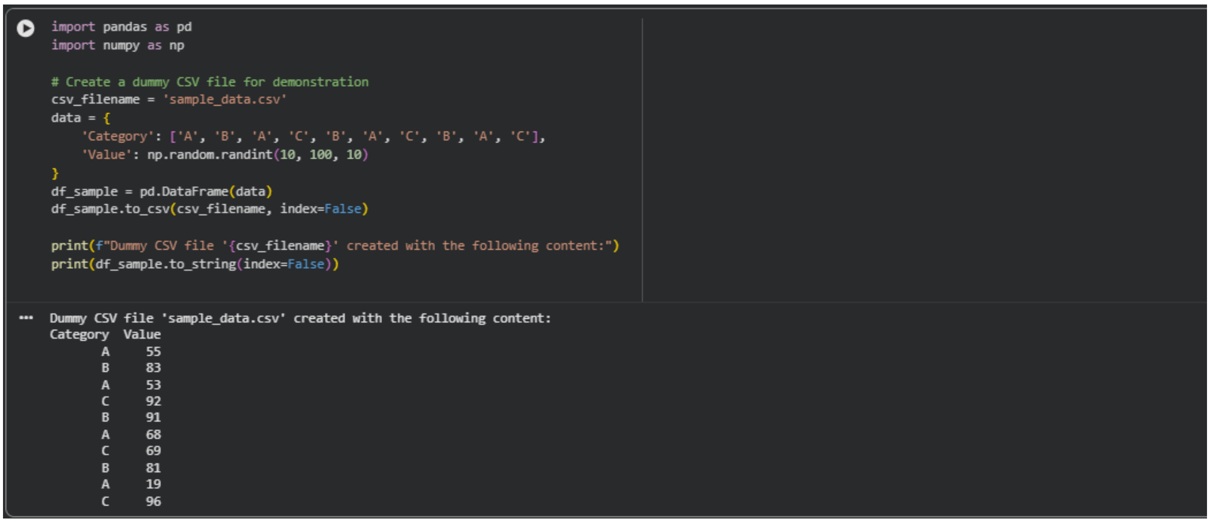
S
Group the data by 'Category' and calculate aggregate statistics
Plot the 'Value' column from
What can I help you build?
+
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more



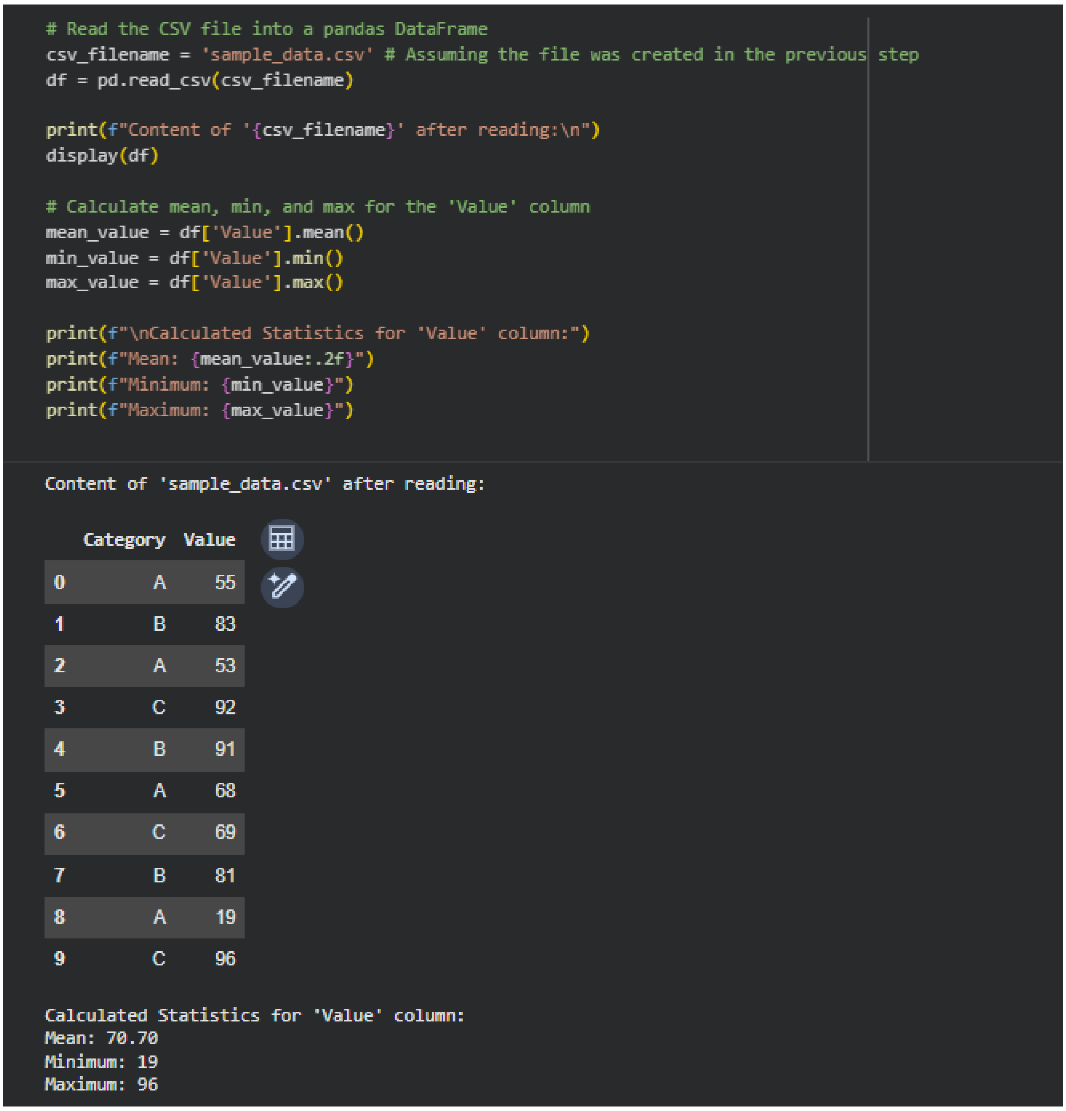
```python
# Define the filename
filename = 'my_sample_file.txt'

# Step 1: Create a text file and write sample text
print(f"Creating '{filename}' and writing content...")
with open(filename, 'w', encoding='utf-8') as file:
    file.write("Hello, this is a sample line.\n")
    file.write("This is the second line of text.\n")
    file.write("And this is the final line for demonstration.")
print("Content written successfully.\n")

# Step 2: Read and display the content of the file
print(f"Reading content from '{filename}':\n")
with open(filename, 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
print("\nFile content displayed.")
```

```
Creating 'my_sample_file.txt' and writing content...
Content written successfully.

Reading content from 'my_sample_file.txt':

Hello, this is a sample line.
This is the second line of text.
And this is the final line for demonstration.

File content displayed.
```



```python
import pandas as pd
import numpy as np

# Create a dummy CSV file for demonstration
csv_filename = 'sample_data.csv'
data = {
    'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'B', 'A', 'C'],
    'Value': np.random.randint(10, 100, 10)
}
df_sample = pd.DataFrame(data)
df_sample.to_csv(csv_filename, index=False)

print(f"Dummy CSV file '{csv_filename}' created with the following content:")
print(df_sample.to_string(index=False))
```

```
Dummy CSV file 'sample_data.csv' created with the following content:
Category  Value
       A     55
       B     83
       A     53
       C     92
       B     91
       A     68
       C     69
       B     81
       A     19
       C     96
```



```python
# Read the CSV file into a pandas DataFrame
csv_filename = 'sample_data.csv' # Assuming the file was created in the previous step
df = pd.read_csv(csv_filename)

print(f"Content of '{csv_filename}' after reading:\n")
display(df)

# Calculate mean, min, and max for the 'Value' column
mean_value = df['Value'].mean()
min_value = df['Value'].min()
max_value = df['Value'].max()

print(f"\nCalculated Statistics for 'Value' column:")
print(f"Mean: {mean_value:.2f}")
print(f"Minimum: {min_value}")
print(f"Maximum: {max_value}")
```

```
Content of 'sample_data.csv' after reading:
```

	Category	Value
0	A	55
1	B	83
2	A	53
3	C	92
4	B	91
5	A	68
6	C	69
7	B	81
8	A	19
9	C	96

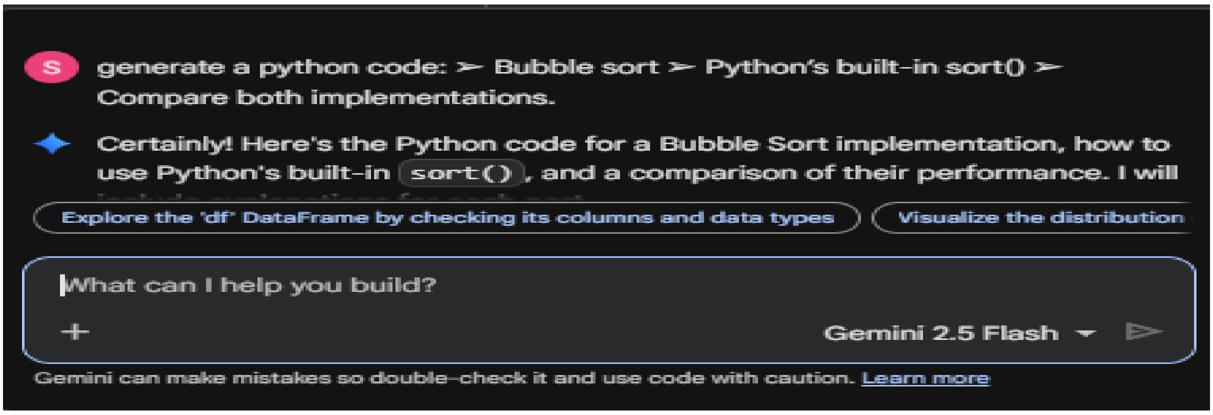
```
Calculated Statistics for 'Value' column:
Mean: 70.70
Minimum: 19
Maximum: 96
```

Code Explanation: Reading CSV and Calculating Statistics

This cell reads the previously created CSV file and then calculates the mean, minimum, and maximum values of its 'Value' column.

1. **csv_filename = 'sample_data.csv'**: Specifies the name of the CSV file to read.

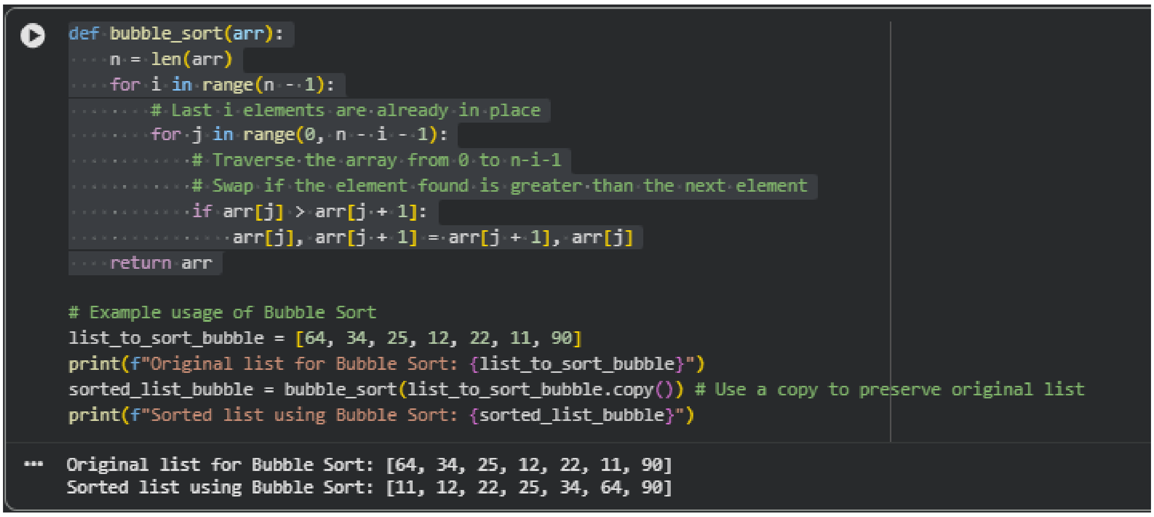
2. **df = pd.read_csv(csv_filename)**: Reads the CSV file into a pandas DataFrame named df.
3. **display(df)**: Prints the entire DataFrame to show its content after reading.
4. **mean_value = df['Value'].mean()**: Calculates the average of all values in the 'Value' column.
5. **min_value = df['Value'].min()**: Finds the smallest value in the 'Value' column.
6. **max_value = df['Value'].max()**: Finds the largest value in the 'Value' column.
7. Finally, the calculated mean, minimum, and maximum values are printed to the console.



generate a python code: ➢ Bubble sort ➢ Python's built-in sort() ➢ Compare both implementations.

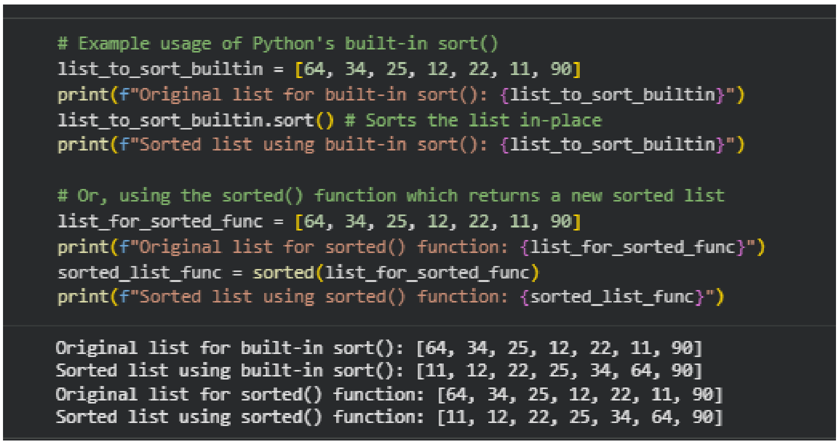
Certainly! Here's the Python code for a Bubble Sort implementation, how to use Python's built-in `sort()`, and a comparison of their performance. I will

S
Explore the 'df' DataFrame by checking its columns and data types
Visualize the distribution
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more



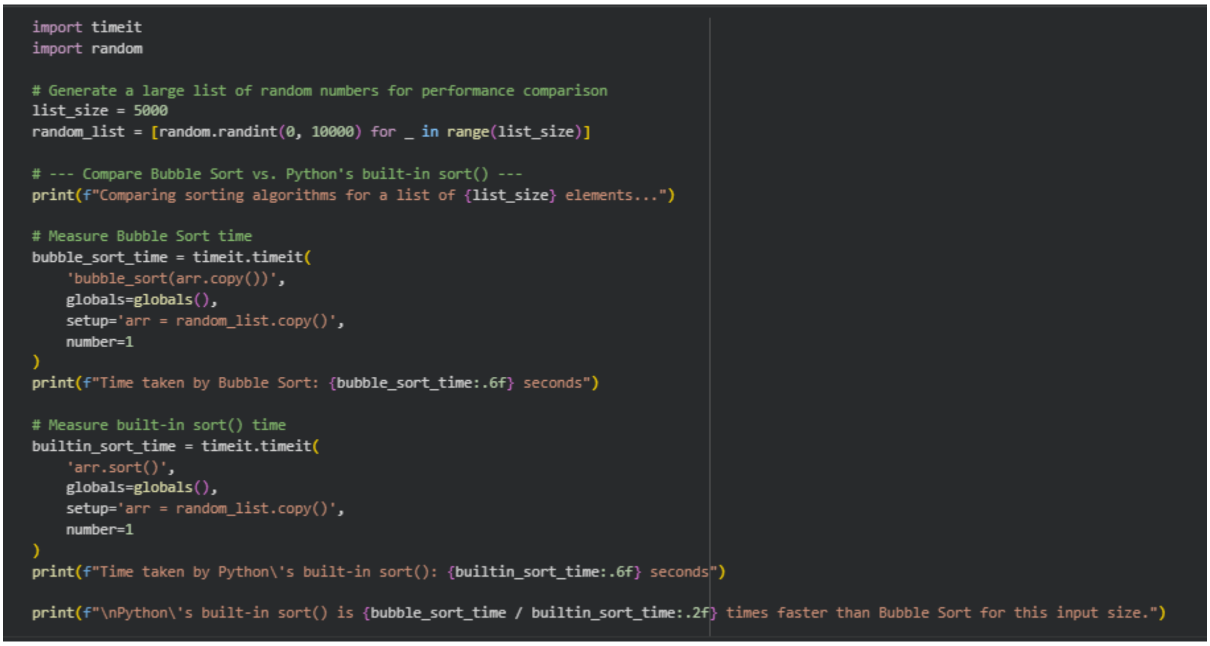
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# Example usage of Bubble Sort
list_to_sort_bubble = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for Bubble Sort: {list_to_sort_bubble}")
sorted_list_bubble = bubble_sort(list_to_sort_bubble.copy()) # Use a copy to preserve original list
print(f"Sorted list using Bubble Sort: {sorted_list_bubble}")
```

```
Original list for Bubble Sort: [64, 34, 25, 12, 22, 11, 90]
Sorted list using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
```



```python
# Example usage of Python's built-in sort()
list_to_sort_builtin = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for built-in sort(): {list_to_sort_builtin}")
list_to_sort_builtin.sort() # Sorts the list in-place
print(f"Sorted list using built-in sort(): {list_to_sort_builtin}")

# Or, using the sorted() function which returns a new sorted list
list_for_sorted_func = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for sorted() function: {list_for_sorted_func}")
sorted_list_func = sorted(list_for_sorted_func)
print(f"Sorted list using sorted() function: {sorted_list_func}")
```

```
Original list for built-in sort(): [64, 34, 25, 12, 22, 11, 90]
Sorted list using built-in sort(): [11, 12, 22, 25, 34, 64, 90]
Original list for sorted() function: [64, 34, 25, 12, 22, 11, 90]
Sorted list using sorted() function: [11, 12, 22, 25, 34, 64, 90]
```



```
import timeit
import random

# Generate a large list of random numbers for performance comparison
list_size = 5000
random_list = [random.randint(0, 10000) for _ in range(list_size)]

# --- Compare Bubble Sort vs. Python's built-in sort() ---
print(f"Comparing sorting algorithms for a list of {list_size} elements...")

# Measure Bubble Sort time
bubble_sort_time = timeit.timeit(
    'bubble_sort(arr.copy())',
    globals=globals(),
    setup='arr = random_list.copy()',
    number=1
)
print(f"Time taken by Bubble Sort: {bubble_sort_time:.6f} seconds")

# Measure built-in sort() time
builtin_sort_time = timeit.timeit(
    'arr.sort()',
    globals=globals(),
    setup='arr = random_list.copy()',
    number=1
)
print(f"Time taken by Python\'s built-in sort(): {builtin_sort_time:.6f} seconds")

print(f"\nPython\'s built-in sort() is {bubble_sort_time / builtin_sort_time:.2f} times faster than Bubble Sort for this input size.")
```

Code Explanation: Comparison of Implementations

This section compares the performance of the bubble_sort function against Python's built-in list.sort() method using the timeit module.

1. **import timeit and import random**: Imports necessary modules. timeit is for precise timing of small code snippets, and random is used to generate test data.
2. **Generate Test Data**: A large list (random_list) of list_size (e.g., 5000) random integers is created. This ensures a meaningful comparison, as the performance difference becomes more apparent with larger datasets.
3. **timeit.timeit()**:
    - This function measures the execution time of a small piece of Python code.
    - The first argument is the code string to be timed.
    - globals=globals(): Makes sure the bubble_sort function and random_list are available in the scope where timeit runs the code.
    - setup='arr = random_list.copy()': Before each timing run, a fresh copy of random_listis made and assigned toarr`. This is crucial because both sorting functions modify the list in-place, and we want to sort the same initial data for fair comparison.
    - number=1: Specifies that the code is executed once for timing. For very small operations, timeit often runs multiple times and takes an average, but for sorting a list of 5000 elements, one run is sufficient.
4. **Results**: The execution times for both bubble_sort and list.sort() are printed. You will typically observe that Python's built-in sort is significantly faster because it uses a highly optimized algorithm (Timsort) which has a much better average and worst-case time complexity (O(n log n)) compared to Bubble Sort (O(n^2)).