AI ASSISTED CODING

ASSIGNMENT 2.3

Name: A. Tejasri

Ht No: 2303A52455

Batch – 31

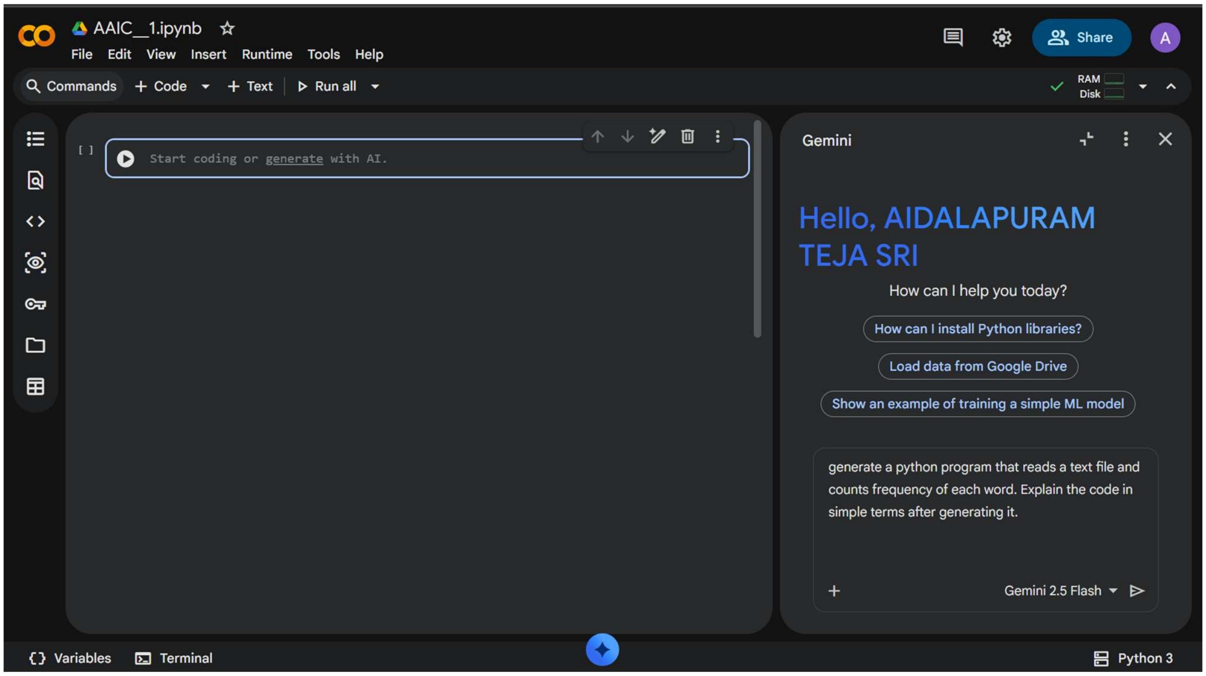
Task 1:

Word Frequency from Text File

Scenario: You are analyzing log files for keyword frequency.

Task: Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code

Prompt:



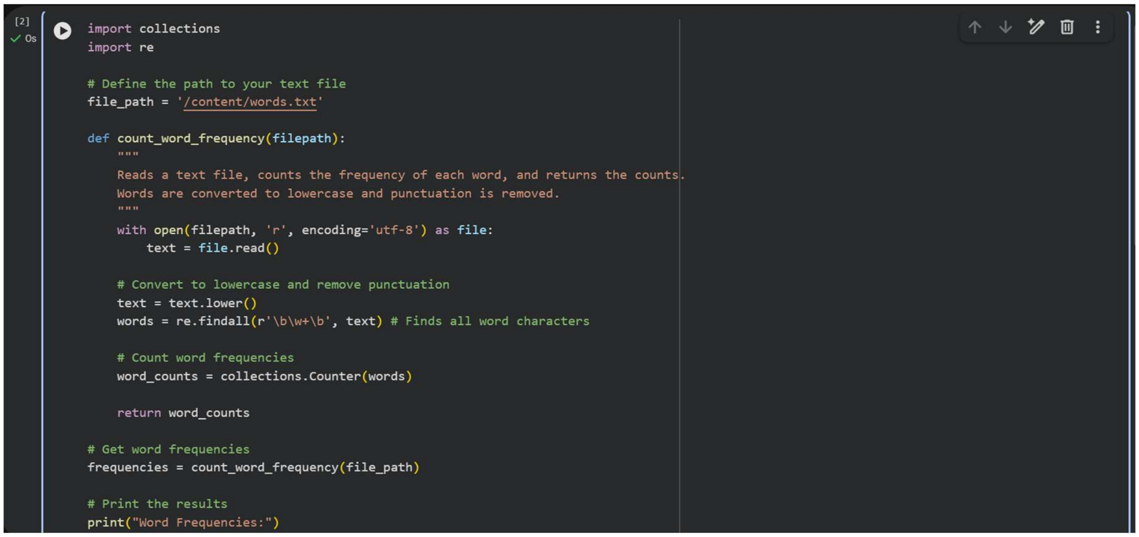
AAIC__1.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
Start coding or generate with AI.
Gemini
How can I install Python libraries?
Load data from Google Drive
Show an example of training a simple ML model
Gemini 2.5 Flash
Variables Terminal
Python 3

Hello, AIDALAPURAM TEJA SRI

How can I help you today?

generate a python program that reads a text file and counts frequency of each word. Explain the code in simple terms after generating it.

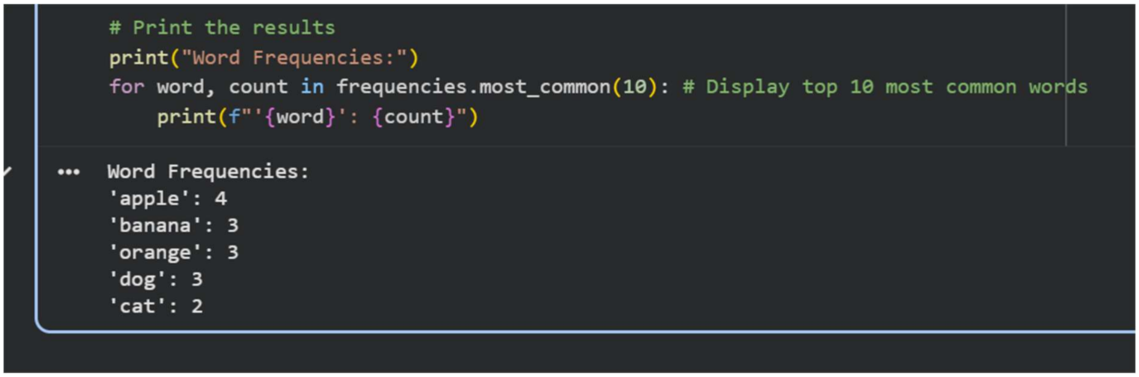
Code Screenshot:



```
[2]
✓ 0s
```

```python
import collections
import re

# Define the path to your text file
file_path = '/content/words.txt'

def count_word_frequency(filepath):
    """
    Reads a text file, counts the frequency of each word, and returns the counts.
    Words are converted to lowercase and punctuation is removed.
    """
    with open(filepath, 'r', encoding='utf-8') as file:
        text = file.read()

    # Convert to lowercase and remove punctuation
    text = text.lower()
    words = re.findall(r'\b\w+\b', text) # Finds all word characters

    # Count word frequencies
    word_counts = collections.Counter(words)

    return word_counts

# Get word frequencies
frequencies = count_word_frequency(file_path)

# Print the results
print("Word Frequencies:")
```

OUTPUT(O/P):



```python
# Print the results
print("Word Frequencies:")
for word, count in frequencies.most_common(10): # Display top 10 most common words
    print(f"'{word}': {count}")
```

```
Word Frequencies:
'apple': 4
'banana': 3
'orange': 3
'dog': 3
'cat': 2
```

Explanation:

This program demonstrates basic file handling in Python using Cursor. First, a text file is created and sample text is written into it. Then, the same file is opened in read mode and its contents are displayed on the screen. It shows how Python can be used to create, write, and read files easily. Such operations are useful in automation and data storage tasks.

Task 2: File Operations Using Cursor AI

Scenario: You are automating basic file operations.

Task: Use Cursor AI to generate a program that:

~Creates a text file

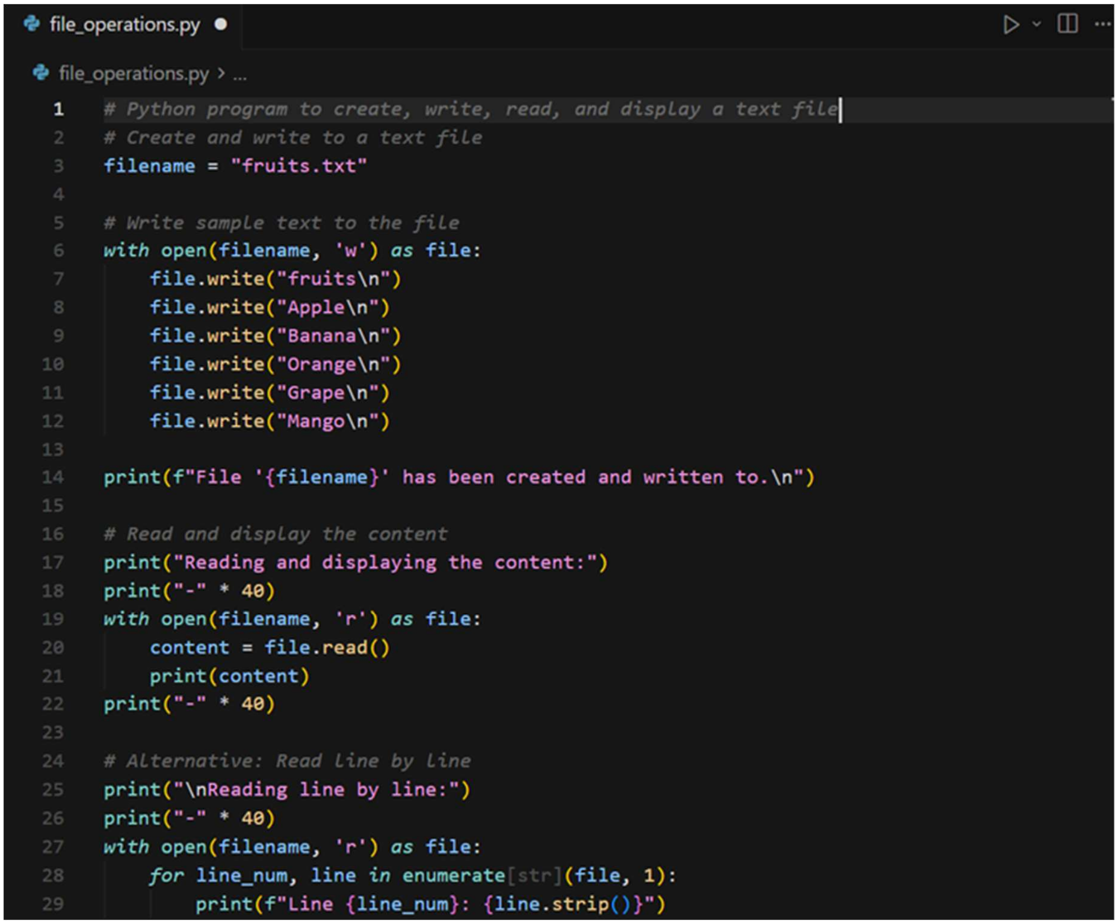
~Writes sample text

~Reads and displays the content

Prompt:

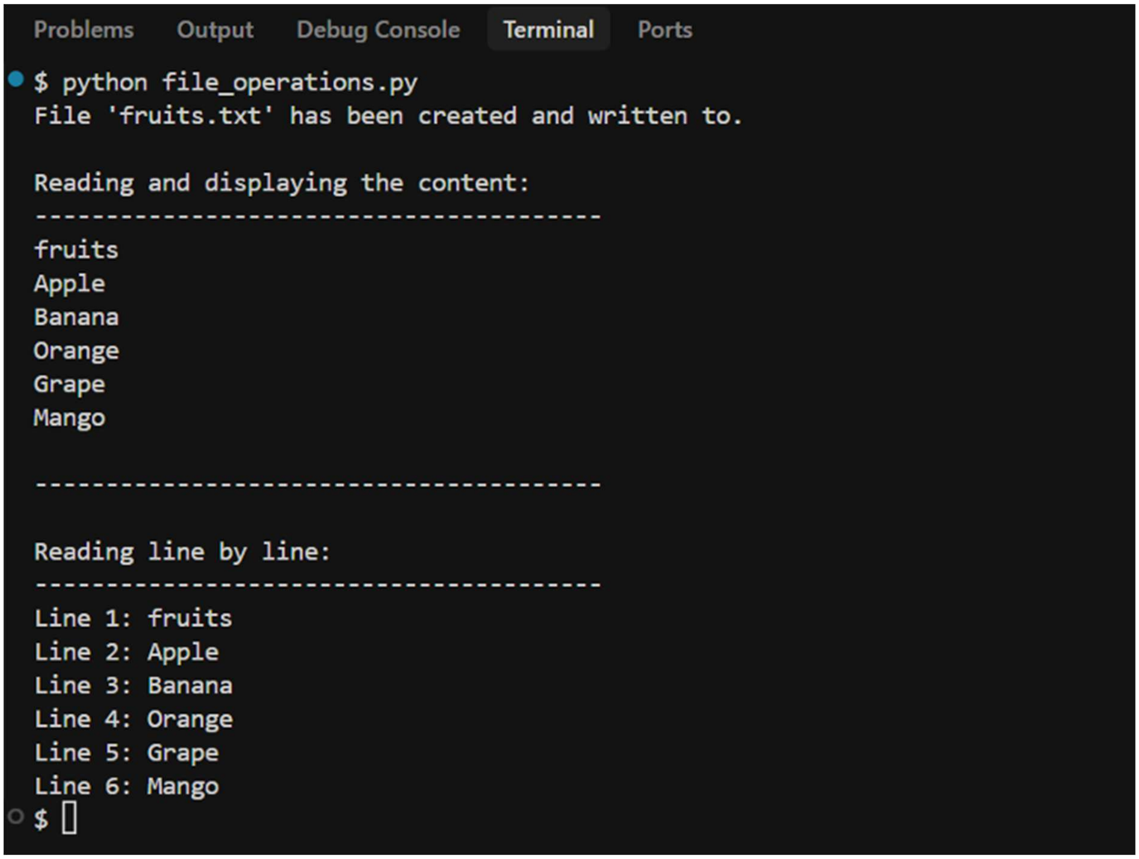
Generate a python program, which creates a text file, write sample txt eg.fruits in that, read and display that content.

Code Screenshot:



```python
# Python program to create, write, read, and display a text file
# Create and write to a text file
filename = "fruits.txt"

# Write sample text to the file
with open(filename, 'w') as file:
    file.write("fruits\n")
    file.write("Apple\n")
    file.write("Banana\n")
    file.write("Orange\n")
    file.write("Grape\n")
    file.write("Mango\n")

print(f"File '{filename}' has been created and written to.\n")

# Read and display the content
print("Reading and displaying the content:")
print("-" * 40)
with open(filename, 'r') as file:
    content = file.read()
    print(content)
print("-" * 40)

# Alternative: Read line by line
print("\nReading line by line:")
print("-" * 40)
with open(filename, 'r') as file:
    for line_num, line in enumerate[str](file, 1):
        print(f"Line {line_num}: {line.strip()}")
```

Output(O/P):



```
$ python file_operations.py
File 'fruits.txt' has been created and written to.

Reading and displaying the content:
----------------------------------------
fruits
Apple
Banana
Orange
Grape
Mango

----------------------------------------

Reading line by line:
----------------------------------------
Line 1: fruits
Line 2: Apple
Line 3: Banana
Line 4: Orange
Line 5: Grape
Line 6: Mango
$ 
```

Explanation:

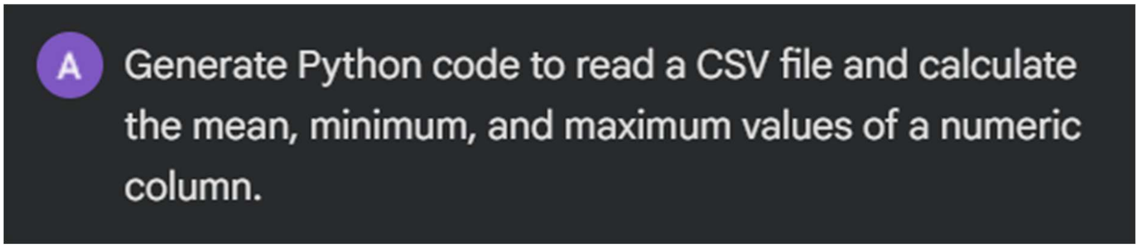
This program demonstrates basic file handling in Python using Cursor. First, a text file is created and sample text is written into it. Then, the same file is opened in read mode and its contents are displayed on the screen. It shows how Python can be used to create, write, and read files easily. Such operations are useful in automation and data storage tasks.

Task 3: CSV Data Analysis

Scenario: You are processing structured data from a CSV file.
Task: Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

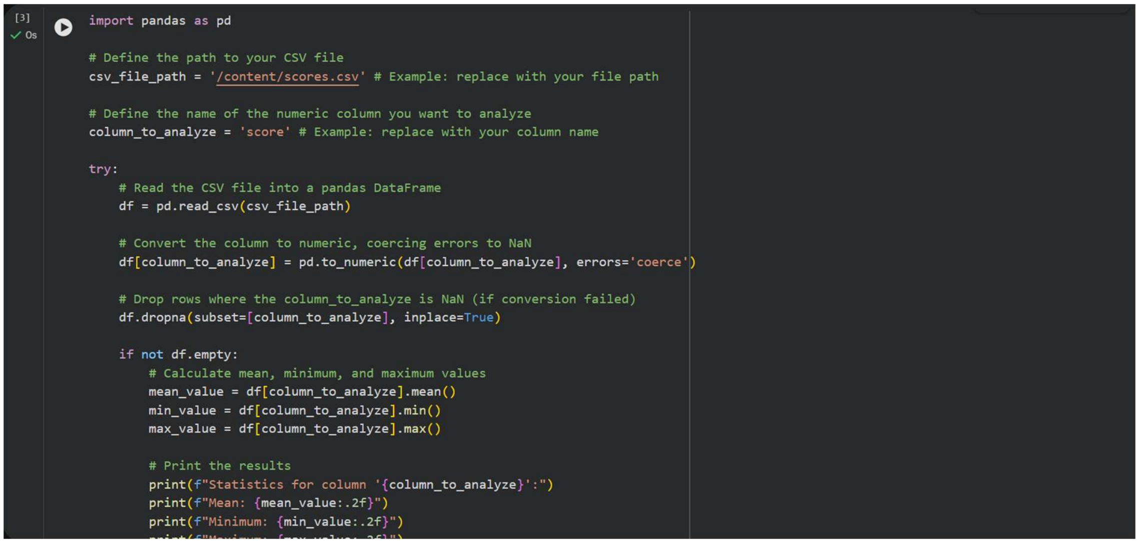
Prompt:

Generate Python code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.

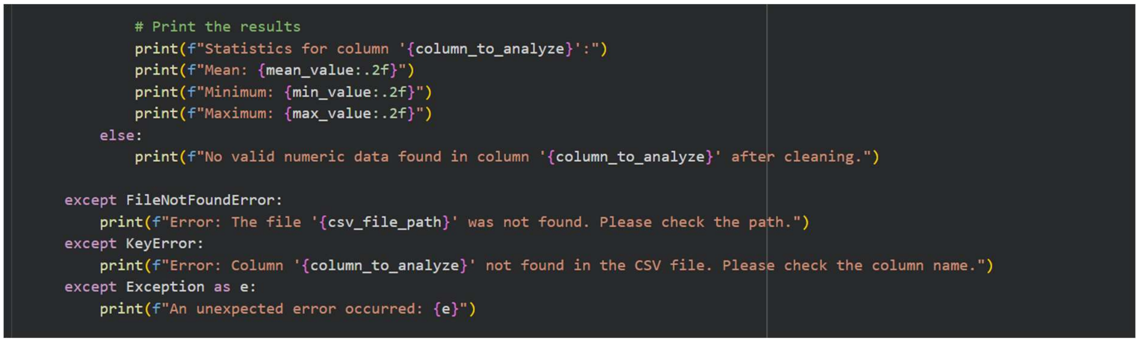


Generate Python code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.

Code Screenshot:

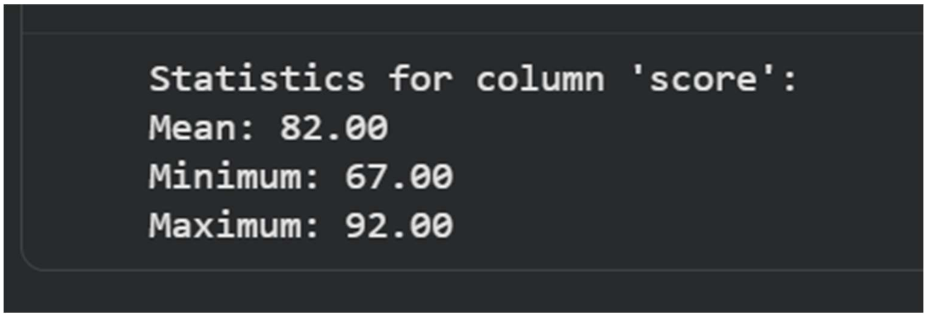


```python
import pandas as pd

# Define the path to your CSV file
csv_file_path = '/content/scores.csv' # Example: replace with your file path

# Define the name of the numeric column you want to analyze
column_to_analyze = 'score' # Example: replace with your column name

try:
    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(csv_file_path)

    # Convert the column to numeric, coercing errors to NaN
    df[column_to_analyze] = pd.to_numeric(df[column_to_analyze], errors='coerce')

    # Drop rows where the column_to_analyze is NaN (if conversion failed)
    df.dropna(subset=[column_to_analyze], inplace=True)

    if not df.empty:
        # Calculate mean, minimum, and maximum values
        mean_value = df[column_to_analyze].mean()
        min_value = df[column_to_analyze].min()
        max_value = df[column_to_analyze].max()

        # Print the results
        print(f"Statistics for column '{column_to_analyze}':")
        print(f"Mean: {mean_value:.2f}")
        print(f"Minimum: {min_value:.2f}")
```



```python
            # Print the results
            print(f"Statistics for column '{column_to_analyze}':")
            print(f"Mean: {mean_value:.2f}")
            print(f"Minimum: {min_value:.2f}")
            print(f"Maximum: {max_value:.2f}")
        else:
            print(f"No valid numeric data found in column '{column_to_analyze}' after cleaning.")

    except FileNotFoundError:
        print(f"Error: The file '{csv_file_path}' was not found. Please check the path.")
    except KeyError:
        print(f"Error: Column '{column_to_analyze}' not found in the CSV file. Please check the column name.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

OUTPUT(O/P):



```
Statistics for column 'score':
Mean: 82.00
Minimum: 67.00
Maximum: 92.00
```

Explanation:

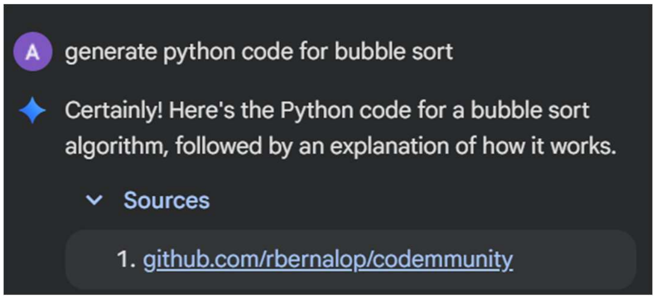
This program reads data from a CSV file using Python. It extracts numerical values from a column and calculates the mean, minimum, and maximum. CSV analysis is used in data processing and analytics applications.

Task 4: Sorting Lists Manual vs Built-in Scenario:

You are reviewing algorithm choices for efficiency.

Task: Use Gemini to generate: Bubble sort Python's built-in sort() Compare both implementations.

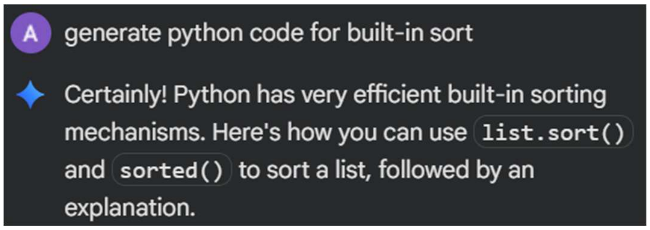
Prompt:



generate python code for bubble sort

Certainly! Here's the Python code for a bubble sort algorithm, followed by an explanation of how it works.

1. github.com/rbernalop/codemmunity

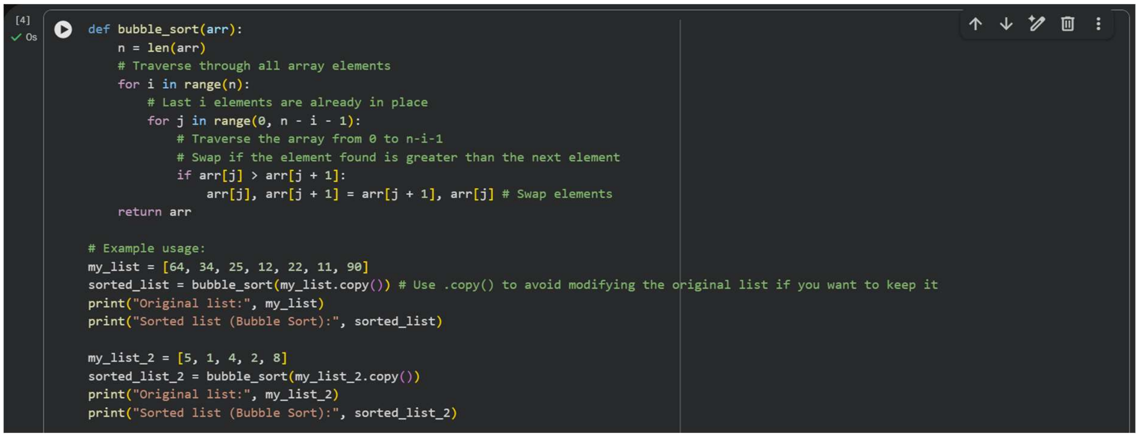


generate python code for built-in sort

Certainly! Python has very efficient built-in sorting mechanisms. Here's how you can use `list.sort()` and `sorted()` to sort a list, followed by an explanation.

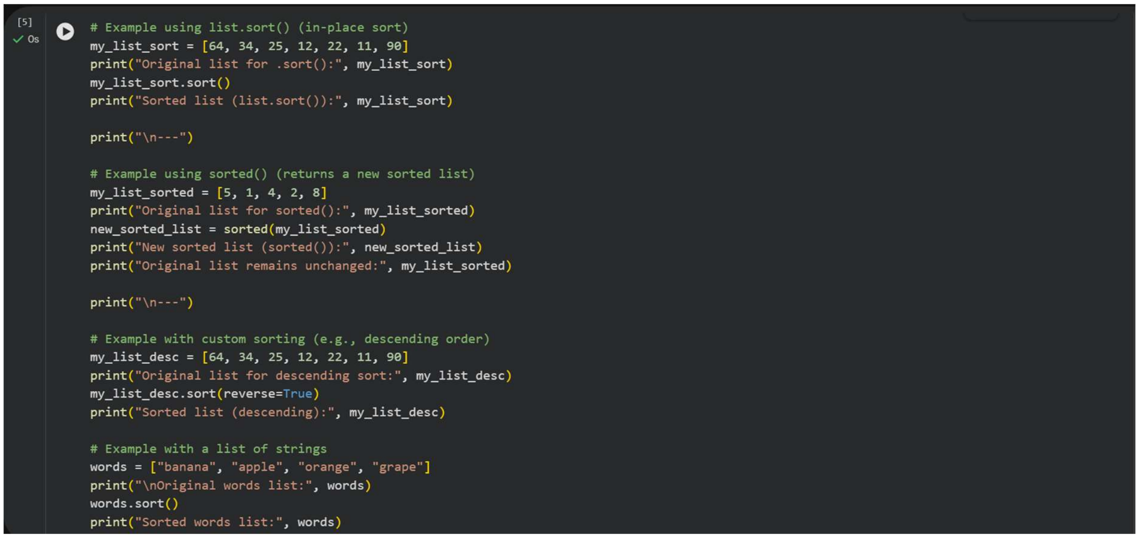
Code Screenshot:

Manual Bubble Sort code:



```python
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # 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] # Swap elements
    return arr

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
sorted_list = bubble_sort(my_list.copy()) # Use .copy() to avoid modifying the original list if you want to keep it
print("Original list:", my_list)
print("Sorted list (Bubble Sort):", sorted_list)

my_list_2 = [5, 1, 4, 2, 8]
sorted_list_2 = bubble_sort(my_list_2.copy())
print("Original list:", my_list_2)
print("Sorted list (Bubble Sort):", sorted_list_2)
```

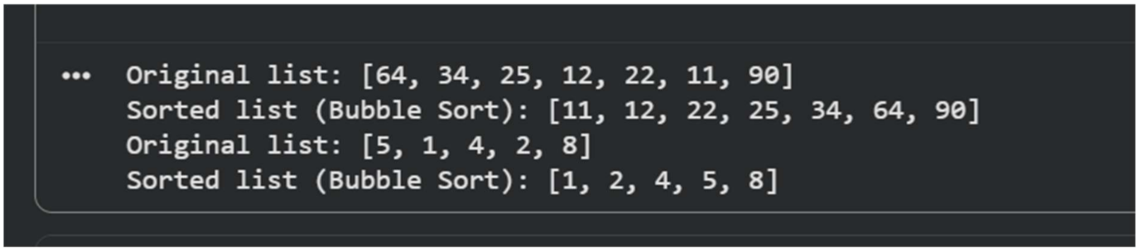
Sorting using sort() function:



```
# Example using list.sort() (in-place sort)
my_list_sort = [64, 34, 25, 12, 22, 11, 90]
print("Original list for .sort():", my_list_sort)
my_list_sort.sort()
print("Sorted list (list.sort()):", my_list_sort)

print("\n---")

# Example using sorted() (returns a new sorted list)
my_list_sorted = [5, 1, 4, 2, 8]
print("Original list for sorted():", my_list_sorted)
new_sorted_list = sorted(my_list_sorted)
print("New sorted list (sorted()):", new_sorted_list)
print("Original list remains unchanged:", my_list_sorted)

print("\n---")

# Example with custom sorting (e.g., descending order)
my_list_desc = [64, 34, 25, 12, 22, 11, 90]
print("Original list for descending sort:", my_list_desc)
my_list_desc.sort(reverse=True)
print("Sorted list (descending):", my_list_desc)

# Example with a list of strings
words = ["banana", "apple", "orange", "grape"]
print("\nOriginal words list:", words)
words.sort()
print("Sorted words list:", words)
```

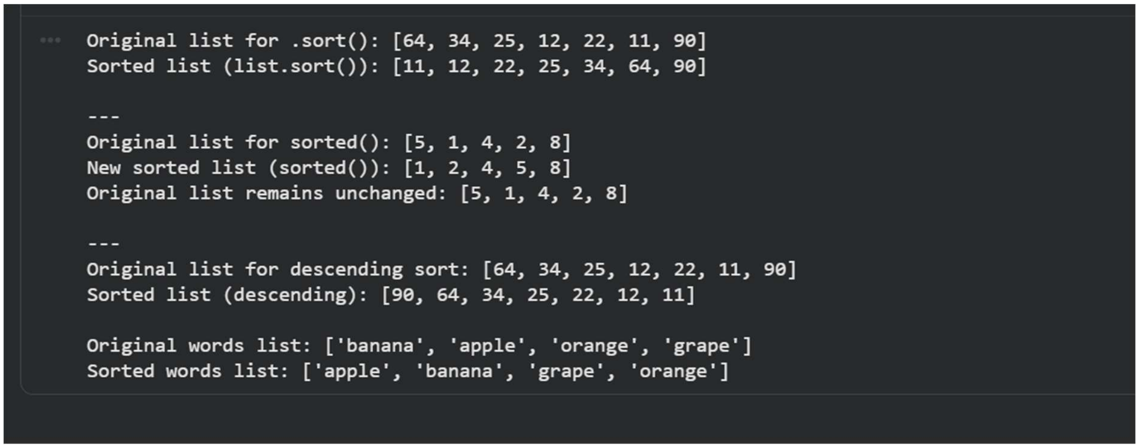
OUTPUT(O/P):

Bubble sort:



```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list (Bubble Sort): [11, 12, 22, 25, 34, 64, 90]
Original list: [5, 1, 4, 2, 8]
Sorted list (Bubble Sort): [1, 2, 4, 5, 8]
```

Sort Function:



```
Original list for .sort(): [64, 34, 25, 12, 22, 11, 90]
Sorted list (list.sort()): [11, 12, 22, 25, 34, 64, 90]

---
Original list for sorted(): [5, 1, 4, 2, 8]
New sorted list (sorted()): [1, 2, 4, 5, 8]
Original list remains unchanged: [5, 1, 4, 2, 8]

---
Original list for descending sort: [64, 34, 25, 12, 22, 11, 90]
Sorted list (descending): [90, 64, 34, 25, 22, 12, 11]

Original words list: ['banana', 'apple', 'orange', 'grape']
Sorted words list: ['apple', 'banana', 'grape', 'orange']
```

Explanation:

Bubble sort is a simple sorting algorithm that repeatedly compares and swaps adjacent elements. It is easy to understand but inefficient for large data sets. Python's built-in sort function is shorter, optimized and much faster. The built-in method should be preferred in real-world applications.