AI ASSISTED CODING

ASSIGNMENT-2.3

NAME: P.RUTHWICK

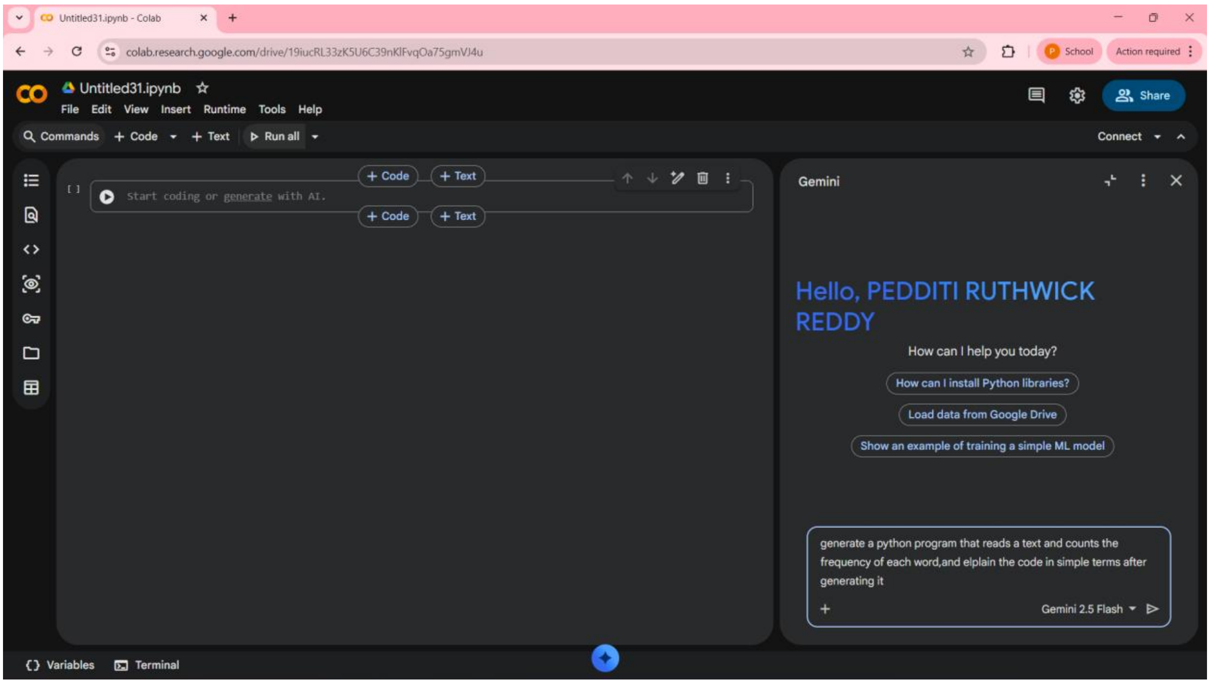
HT_NO:2303A52484

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

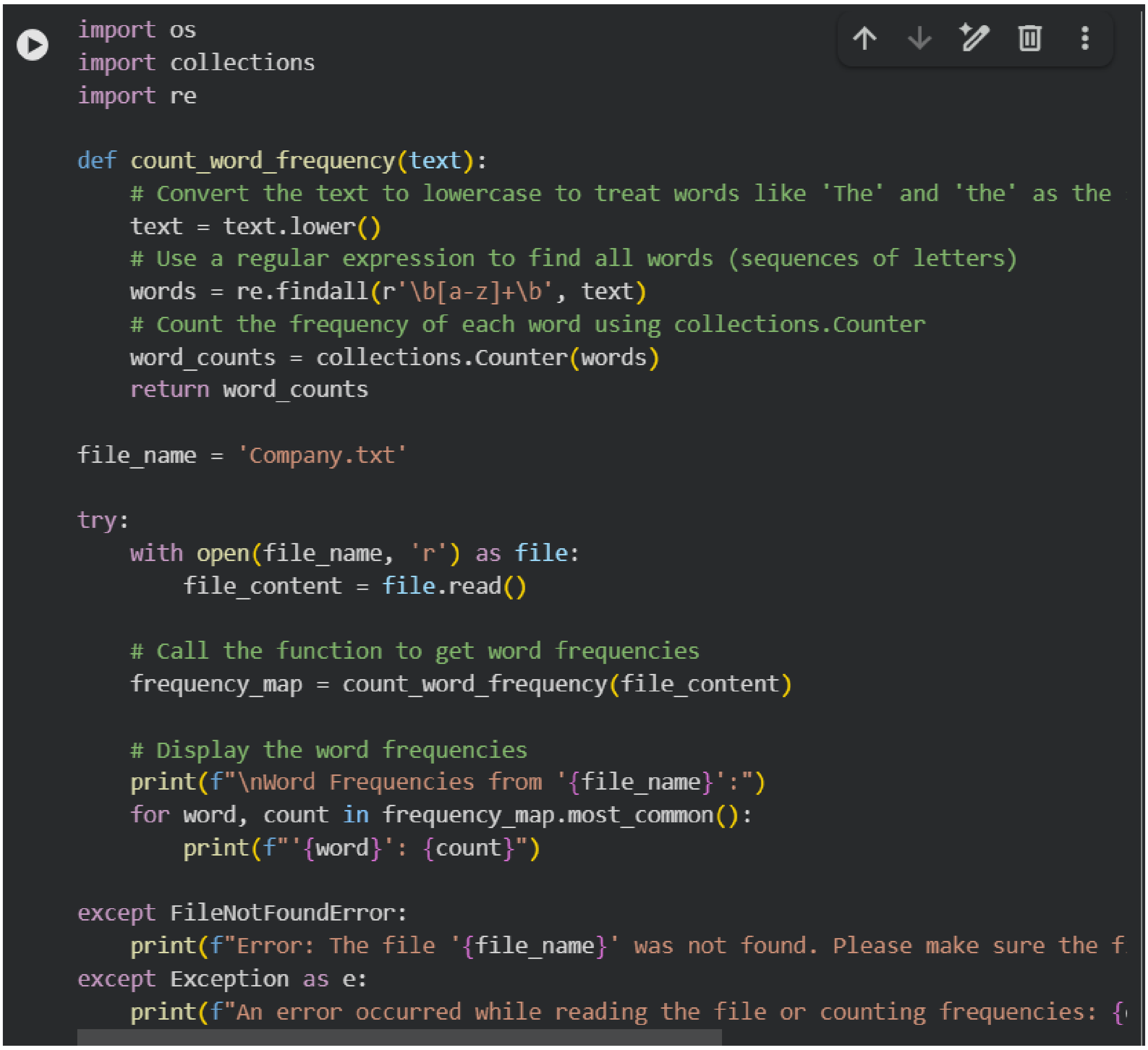
PROMPT:


Untitled31.ipynb - Colab
colab.research.google.com/drive/19iucRL33zK5U6C39nKIFvqOa75gmVJ4u
School
Action required
Untitled31.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Share
Commands
+ Code
+ Text
Run all
Connect
Start coding or generate with AI.
Gemini
Hello, PEDDITI RUTHWICK REDDY
How can I help you today?
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

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

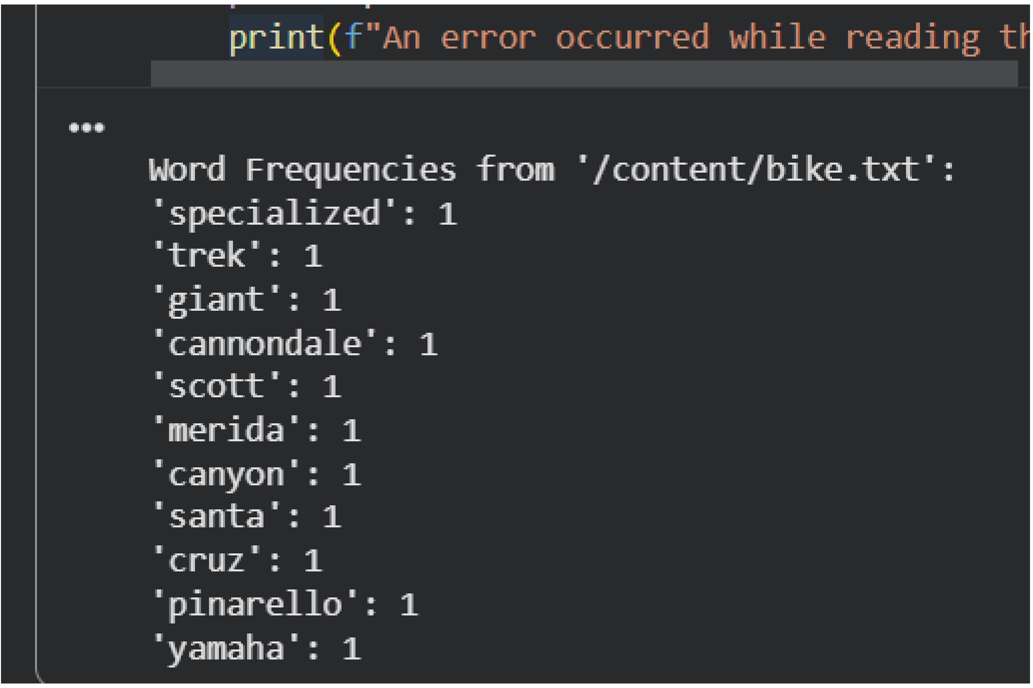
Create a Python program that creates a text file, writes sample text into it, then reads the file and displays its contents. Add clear comments in the code.

Code Screenshot:



```
import os
import collections
import re

def count_word_frequency(text):
    # Convert the text to lowercase to treat words like 'The' and 'the' as the
    text = text.lower()
    # Use a regular expression to find all words (sequences of letters)
    words = re.findall(r'\b[a-z]+\b', text)
    # Count the frequency of each word using collections.Counter
    word_counts = collections.Counter(words)
    return word_counts

file_name = 'Company.txt'

try:
    with open(file_name, 'r') as file:
        file_content = file.read()

    # Call the function to get word frequencies
    frequency_map = count_word_frequency(file_content)

    # Display the word frequencies
    print(f"\nWord Frequencies from '{file_name}':")
    for word, count in frequency_map.most_common():
        print(f"'{word}': {count}")

except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found. Please make sure the f
except Exception as e:
    print(f"An error occurred while reading the file or counting frequencies: {
```

OUTPUT



```
print(f"An error occurred while reading th
...
Word Frequencies from '/content/bike.txt':
'specialized': 1
'trek': 1
'giant': 1
'cannondale': 1
'scott': 1
'merida': 1
'canyon': 1
'santa': 1
'cruz': 1
'pinarello': 1
'yamaha': 1
```

:

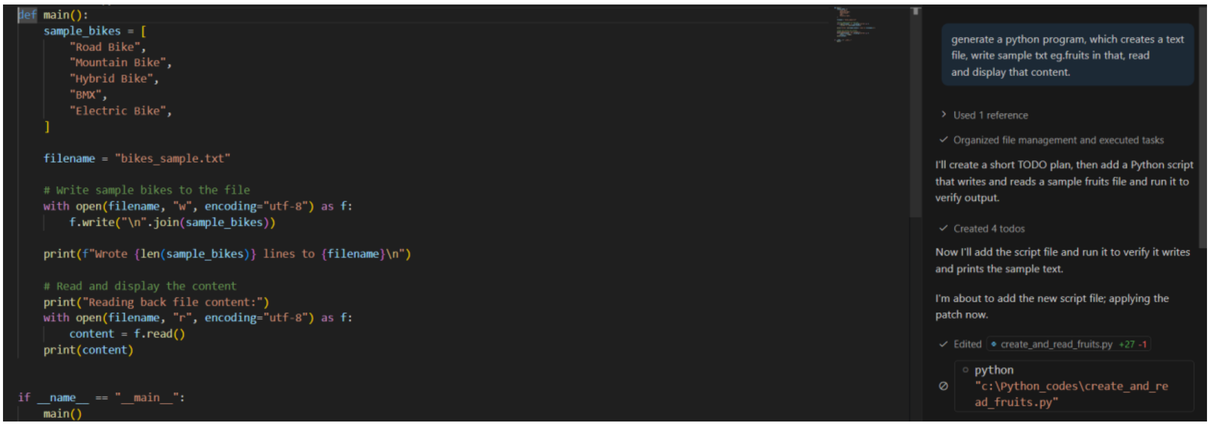
EXPLANATION:

This program reads a text file and splits the content into words. Each word is stored in a dictionary where the key is the word and the value is its count. If a word appears again, its count is increased. Finally, the program prints how many times each word appears in the file. This helps in analyzing text data and finding frequently used words.

Question: 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.



```python
def main():
    sample_bikes = [
        "Road Bike",
        "Mountain Bike",
        "Hybrid Bike",
        "BMX",
        "Electric Bike",
    ]

    filename = "bikes_sample.txt"

    # Write sample bikes to the file
    with open(filename, "w", encoding="utf-8") as f:
        f.write("\n".join(sample_bikes))

    print(f"Wrote {len(sample_bikes)} lines to {filename}\n")

    # Read and display the content
    print("Reading back file content:")
    with open(filename, "r", encoding="utf-8") as f:
        content = f.read()
    print(content)


if __name__ == "__main__":
    main()
```

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

> Used 1 reference

✓ Organized file management and executed tasks

I'll create a short TODO plan, then add a Python script that writes and reads a sample fruits file and run it to verify output.

✓ Created 4 todos

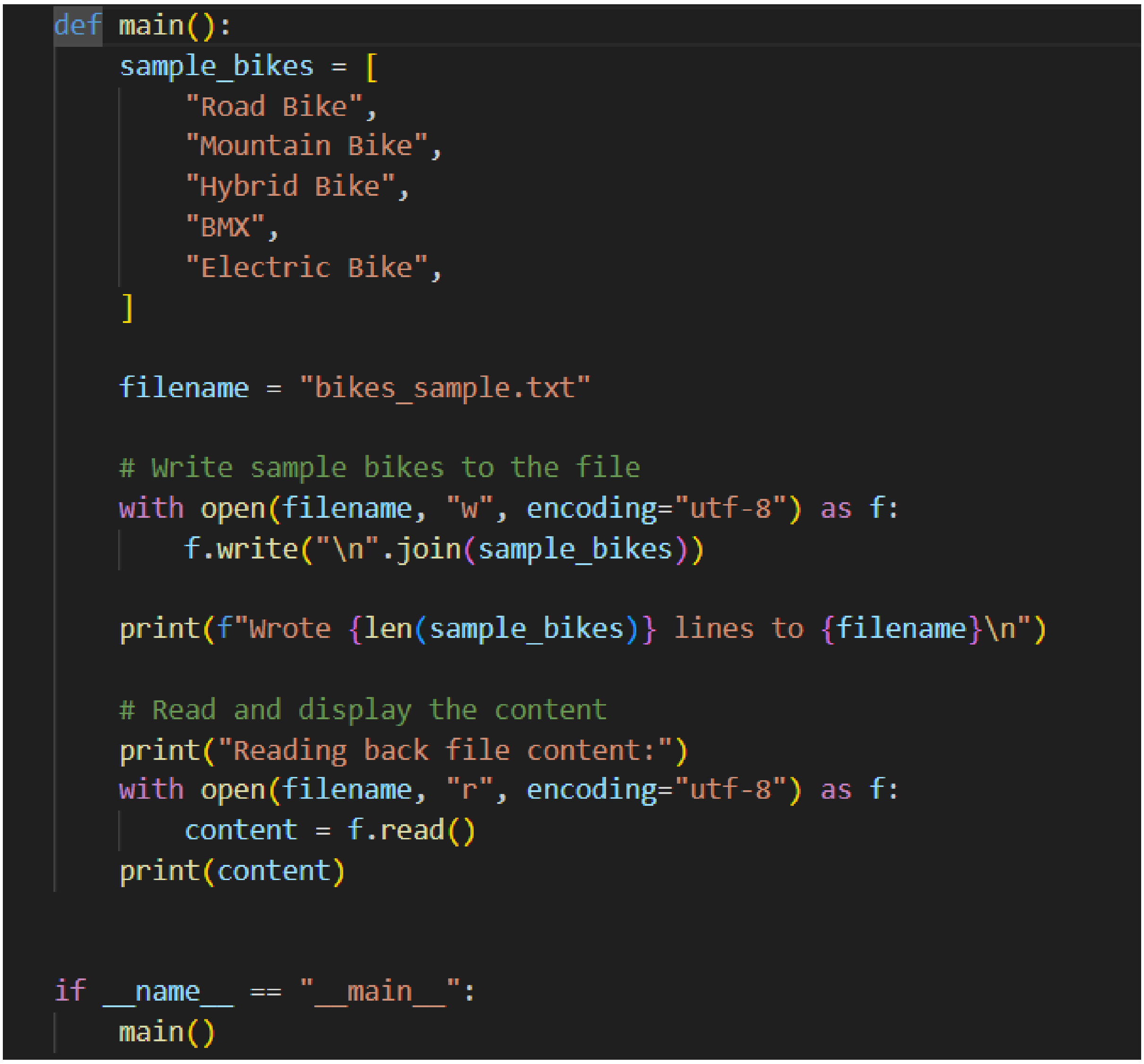
Now I'll add the script file and run it to verify it writes and prints the sample text.

I'm about to add the new script file; applying the patch now.

✓ Edited create_and_read_fruits.py +27 -1

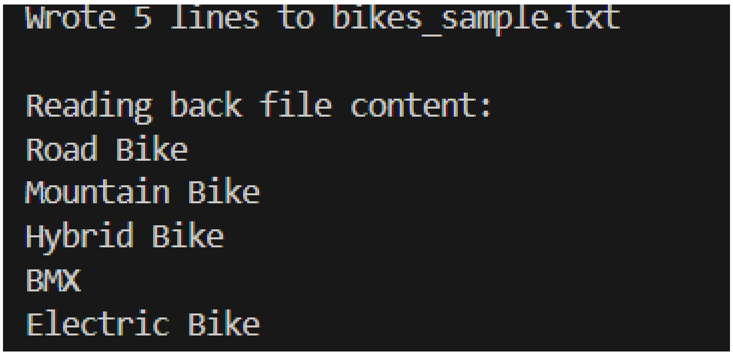
```
python "c:\Python_codes\create_and_read_fruits.py"
```

CODE SCREENSHOT:



```python
def main():
    sample_bikes = [
        "Road Bike",
        "Mountain Bike",
        "Hybrid Bike",
        "BMX",
        "Electric Bike",
    ]

    filename = "bikes_sample.txt"

    # Write sample bikes to the file
    with open(filename, "w", encoding="utf-8") as f:
        f.write("\n".join(sample_bikes))

    print(f"Wrote {len(sample_bikes)} lines to {filename}\n")

    # Read and display the content
    print("Reading back file content:")
    with open(filename, "r", encoding="utf-8") as f:
        content = f.read()
    print(content)


if __name__ == "__main__":
    main()
```

OUTPUT:



```
Wrote 5 lines to bikes_sample.txt

Reading back file content:
Road Bike
Mountain Bike
Hybrid Bike
BMX
Electric Bike
```

Explanation:

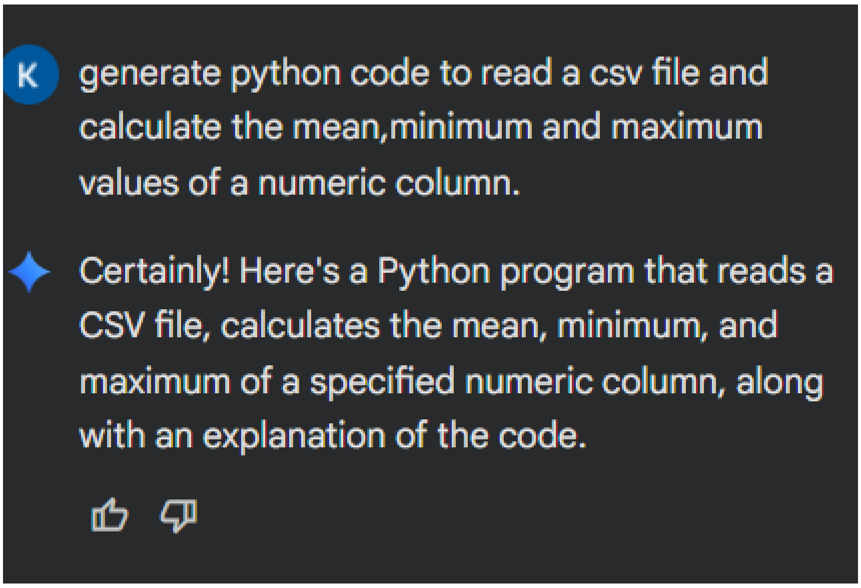
This program demonstrates basic file handling in Python using Cursor. First, a text file is created and sample text is written into it. Then, the same file is opened in read mode and its contents are displayed on the screen. It shows how Python can be used to create, write, and read files easily. Such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

Question:

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:

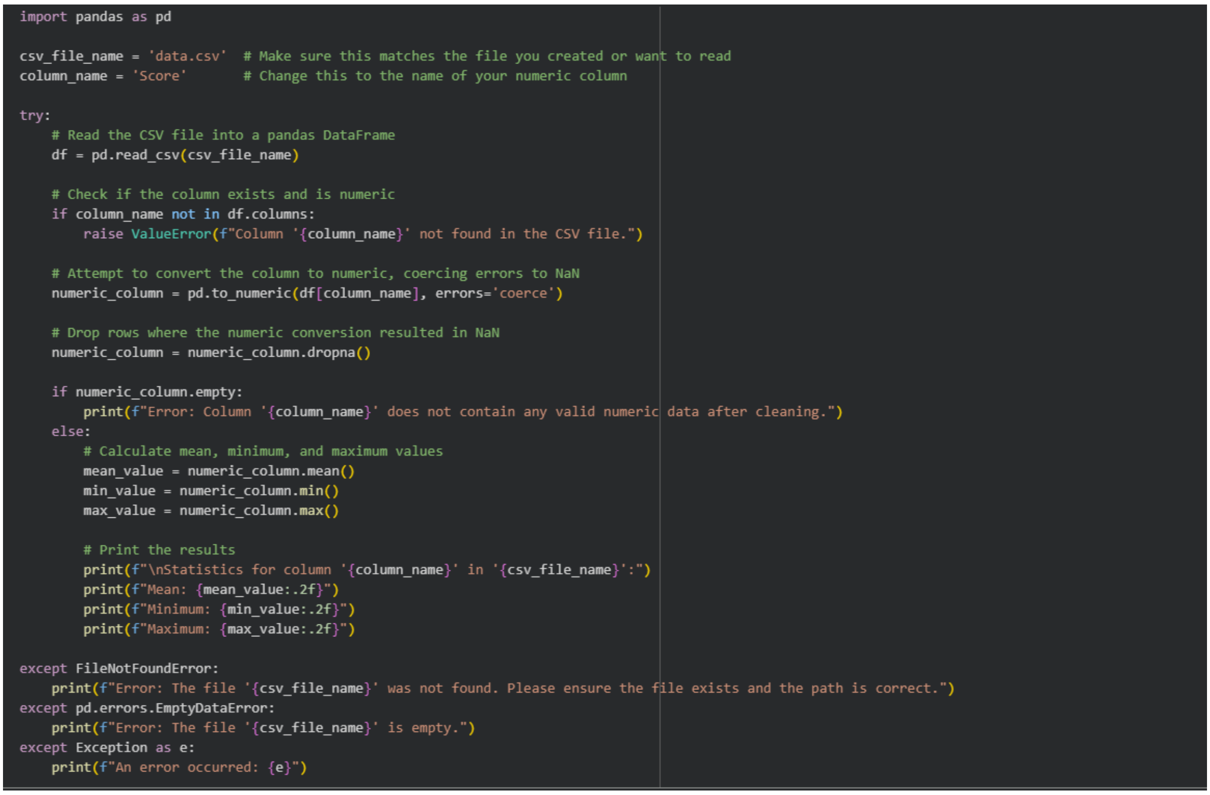
Generate Python code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.



generate python code to read a csv file and calculate the mean,minimum and maximum values of a numeric column.

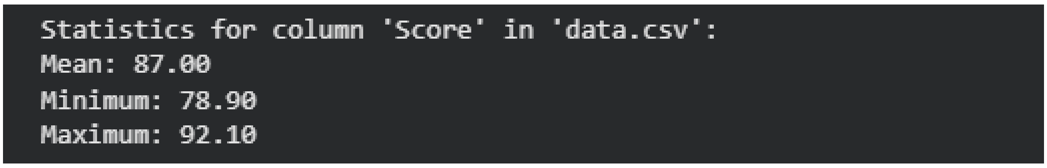
Certainly! Here's a Python program that reads a CSV file, calculates the mean, minimum, and maximum of a specified numeric column, along with an explanation of the code.

CODE SCREENCHOT:



```
import pandas as pd

csv_file_name = 'data.csv'  # Make sure this matches the file you created or want to read
column_name = 'Score'       # Change this to the name of your numeric column

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

    # Check if the column exists and is numeric
    if column_name not in df.columns:
        raise ValueError(f"Column '{column_name}' not found in the CSV file.")

    # Attempt to convert the column to numeric, coercing errors to NaN
    numeric_column = pd.to_numeric(df[column_name], errors='coerce')

    # Drop rows where the numeric conversion resulted in NaN
    numeric_column = numeric_column.dropna()

    if numeric_column.empty:
        print(f"Error: Column '{column_name}' does not contain any valid numeric data after cleaning.")
    else:
        # Calculate mean, minimum, and maximum values
        mean_value = numeric_column.mean()
        min_value = numeric_column.min()
        max_value = numeric_column.max()

        # Print the results
        print(f"\nStatistics for column '{column_name}' in '{csv_file_name}':")
        print(f"Mean: {mean_value:.2f}")
        print(f"Minimum: {min_value:.2f}")
        print(f"Maximum: {max_value:.2f}")

except FileNotFoundError:
    print(f"Error: The file '{csv_file_name}' was not found. Please ensure the file exists and the path is correct.")
except pd.errors.EmptyDataError:
    print(f"Error: The file '{csv_file_name}' is empty.")
except Exception as e:
    print(f"An error occurred: {e}")
```

OUTPUT:



```
Statistics for column 'Score' in 'data.csv':
Mean: 87.00
Minimum: 78.90
Maximum: 92.10
```

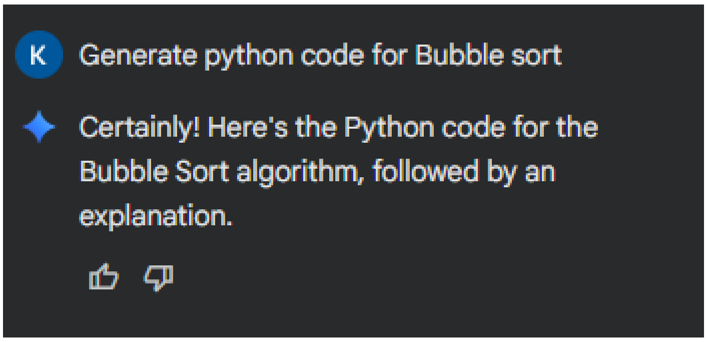
Explanation:

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.

Question:

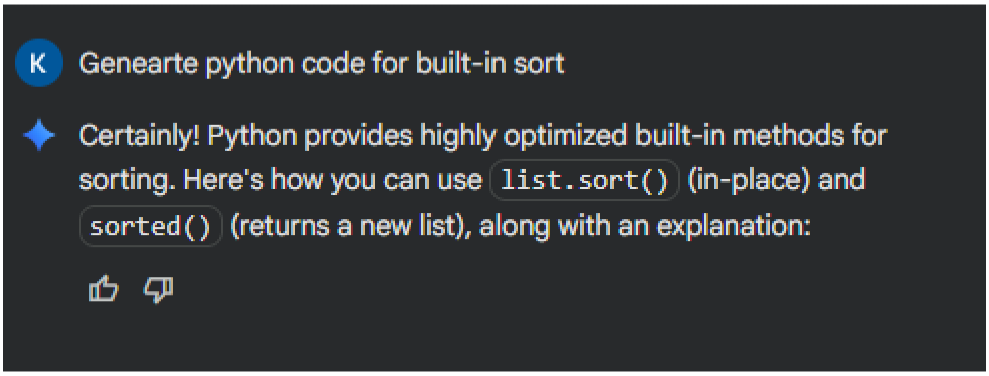
Task 4: Sorting Lists Manual vs Built-in Scenario: You are reviewing algorithm choices for efficiency. Task: Use Gemini to generate: Bubble sort Python's built-in sort() Compare both implementations.

Prompt:



K Generate python code for Bubble sort

Certainly! Here's the Python code for the Bubble Sort algorithm, followed by an explanation.

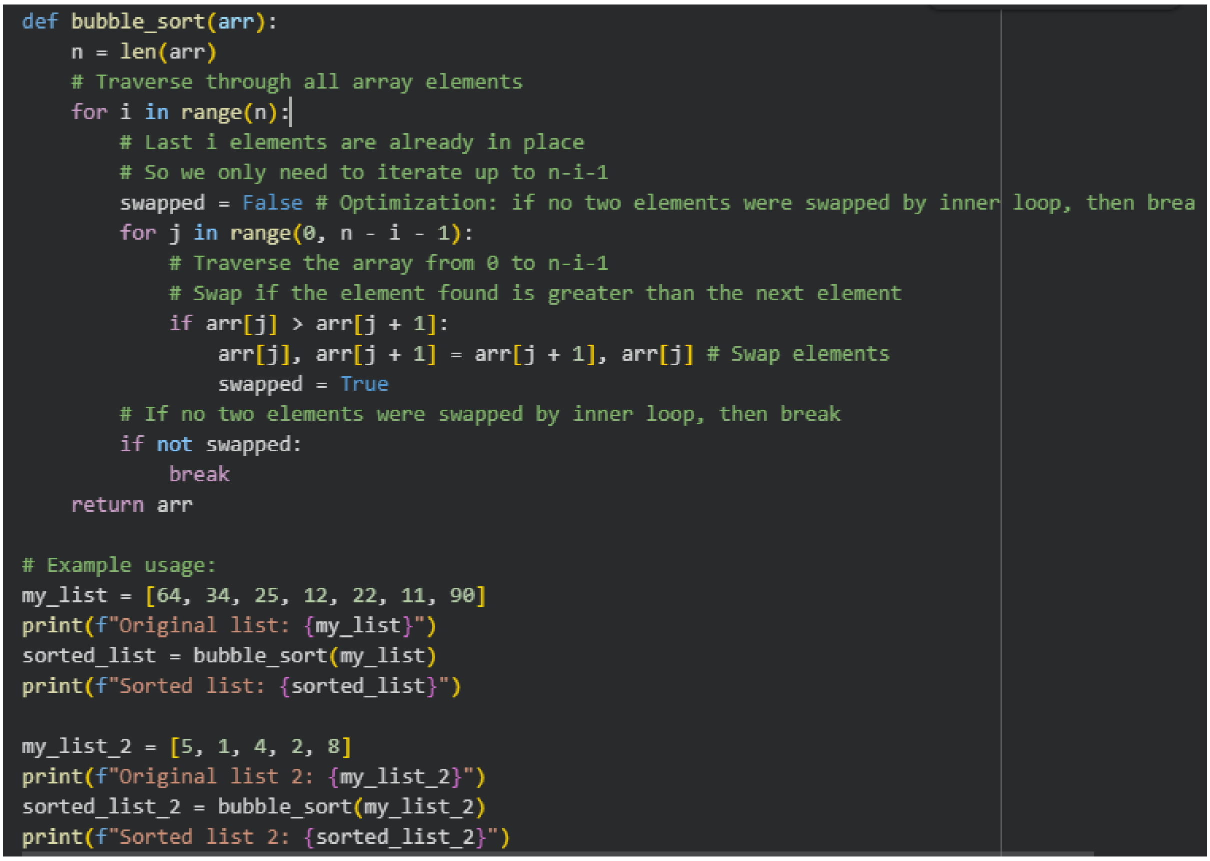


K Genearte python code for built-in sort

Certainly! Python provides highly optimized built-in methods for sorting. Here's how you can use `list.sort()` (in-place) and `sorted()` (returns a new list), along with an explanation:

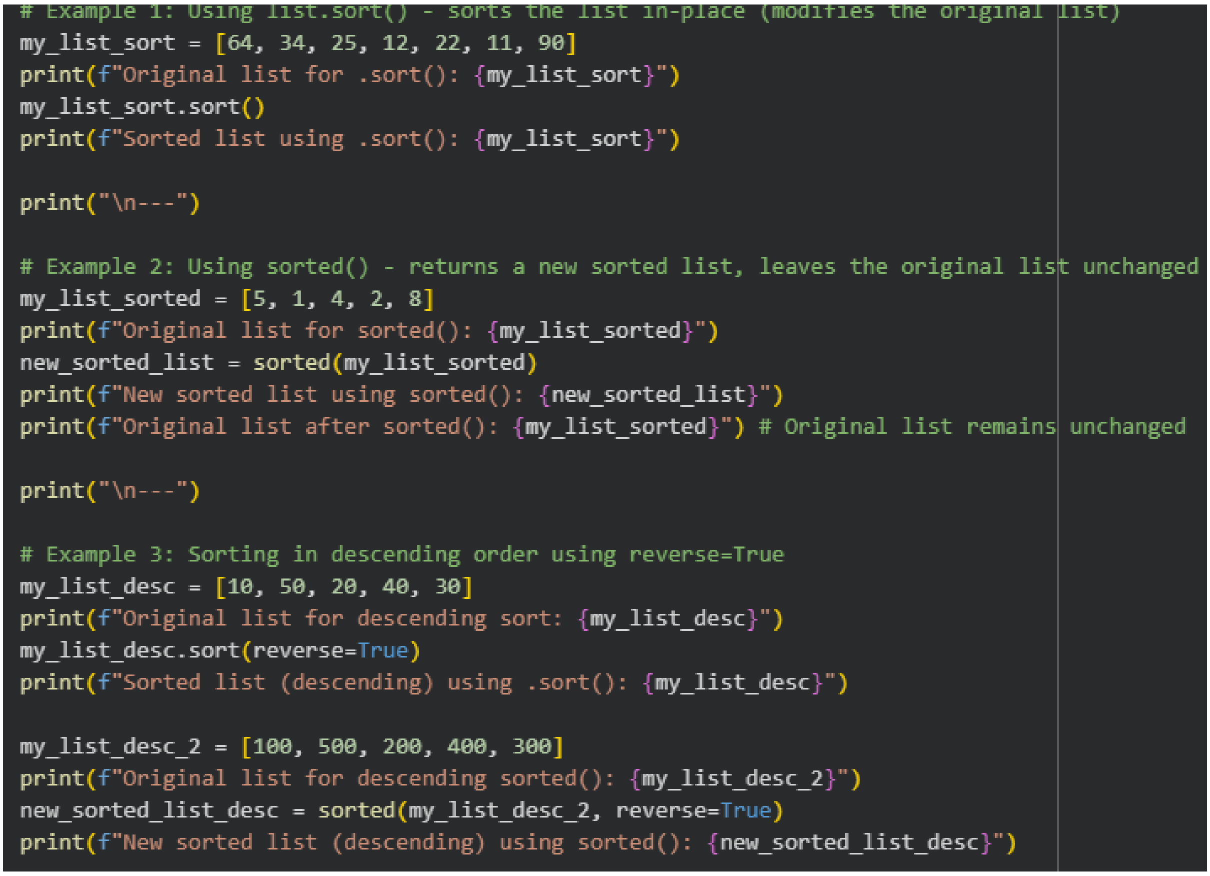
CODE SCREENSHOT:

Manual Bubble sort code:



```
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        # So we only need to iterate up to n-i-1
        swapped = False # Optimization: if no two elements were swapped by inner loop, then brea
        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
                swapped = True
        # If no two elements were swapped by inner loop, then break
        if not swapped:
            break
    return arr

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list: {my_list}")
sorted_list = bubble_sort(my_list)
print(f"Sorted list: {sorted_list}")

my_list_2 = [5, 1, 4, 2, 8]
print(f"Original list 2: {my_list_2}")
sorted_list_2 = bubble_sort(my_list_2)
print(f"Sorted list 2: {sorted_list_2}")
```

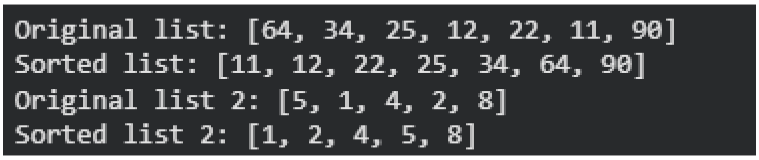
Sorting using sort() function:



```
# Example 1: Using list.sort() - sorts the list in-place (modifies the original list)
my_list_sort = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for .sort(): {my_list_sort}")
my_list_sort.sort()
print(f"Sorted list using .sort(): {my_list_sort}")

print("\n---")

# Example 2: Using sorted() - returns a new sorted list, leaves the original list unchanged
my_list_sorted = [5, 1, 4, 2, 8]
print(f"Original list for sorted(): {my_list_sorted}")
new_sorted_list = sorted(my_list_sorted)
print(f"New sorted list using sorted(): {new_sorted_list}")
print(f"Original list after sorted(): {my_list_sorted}") # Original list remains unchanged

print("\n---")

# Example 3: Sorting in descending order using reverse=True
my_list_desc = [10, 50, 20, 40, 30]
print(f"Original list for descending sort: {my_list_desc}")
my_list_desc.sort(reverse=True)
print(f"Sorted list (descending) using .sort(): {my_list_desc}")

my_list_desc_2 = [100, 500, 200, 400, 300]
print(f"Original list for descending sorted(): {my_list_desc_2}")
new_sorted_list_desc = sorted(my_list_desc_2, reverse=True)
print(f"New sorted list (descending) using sorted(): {new_sorted_list_desc}")
```

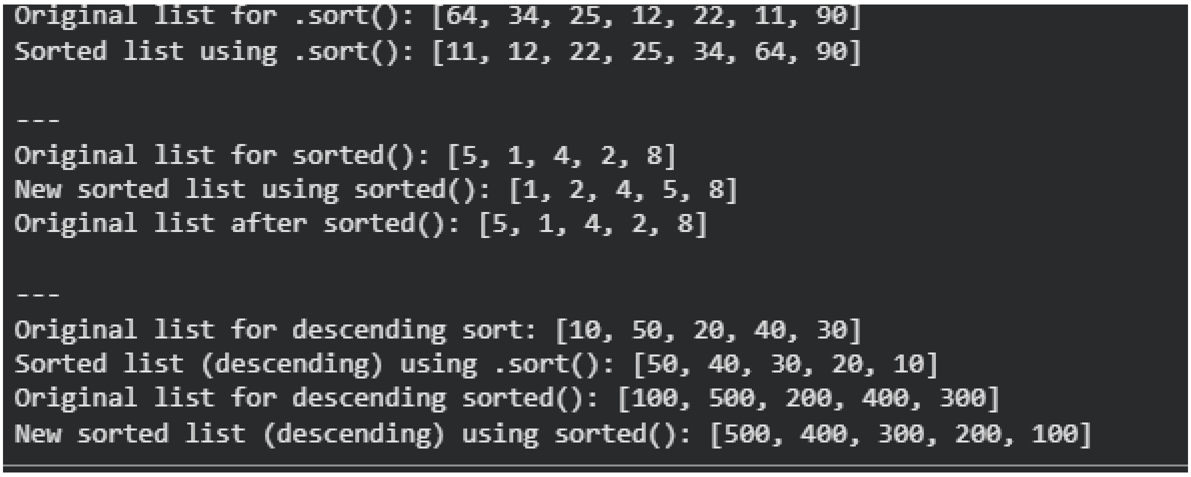
OUTPUT:

Bubble sort:



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

Sort function:



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

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

---
Original list for descending sort: [10, 50, 20, 40, 30]
Sorted list (descending) using .sort(): [50, 40, 30, 20, 10]
Original list for descending sorted(): [100, 500, 200, 400, 300]
New sorted list (descending) using sorted(): [500, 400, 300, 200, 100]
```

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.