Instituto Tecnologico de Monterrey

Maestría en Inteligencia Artificial Aplicada

Curso: Pruebas de Software y aseguramiento de la Calidad

Clave: TC4017.10

Profesor Título: Dr. Gerardo Padilla Zárate

Profesora Asistente: Mtra. Viridiana Rodríguez González

Estudiante: Francisco Javier Ramírez Arias

Matrícula: A01316379

##Actividad: Actividad 4.2

##Ejercicios de Programación #1

##Descripción: Cálculo de estadísticas.

```
## Esta linea nos permite guardar el codigo en
## un archivo físico que nombraremos computeStatistics

%%writefile computeStatistics.py

## Nos indica elinterprete que va a utilizar para
## Ejecutar el archivo, en este caso python3

#!/usr/bin/env python3

import sys    # Accesos a variables y funciones del interprete y del
sistema
import math   # Acceso a funciones matemáticas, indispensables para
estadísticas
import time   # Medición de tiempo de ejecución
```

```python
##Función que abre un archivo, lo recorre línea por línea, convierte
cada palabra
##en un número decimal. Si encuentra algo diferente a un número, lo
ignora y avisa
##al usuario en lugar deterner el programa.

##Carga y valida datos numericos desde un archivo de texto
def read_numbers_from_file(file_path):
    numbers = []

    try:
        with open(file_path, "r", encoding="utf-8") as file:
            for line_number, line in enumerate(file, start=1):
                value = line.strip()
                if not value:
                    continue
                try:
                    numbers.append(float(value))
                except ValueError:
                    print(
                        f"Warning: Invalid data on line {line_number}
-> '{value}'"
                    )
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
        sys.exit(1)

    return numbers

#Función para el cálculo del promedio
def compute_mean(numbers):
    total = 0.0                    #Inicializa la variable total
    for number in numbers:         #Recorrido de la lista
        total += number            #Suma el valor actual al total
acumulado
    return total / len(numbers)    #Regresa el promedio

#Función para el cálculo de la mediana
def compute_median(numbers):
    sorted_numbers = sorted(numbers)  #Ordena los elemenos de forma
ascendente
    count = len(sorted_numbers)       #Número de elementos en la lista
    mid = count // 2                  #Cálcula el punto medio

#Si el total de elementos es impar la mediana el número de en medio.
#Si es total de elementos es par, se toman dos valores centrales y se
promedia.
    if count % 2 == 0:
        return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2
    return sorted_numbers[mid]   #Regresa el valor de la mediana
```

```python
#Cálculo de la moda
def compute_mode(numbers):
    #Diccionario para contar la frecuencia de cada número
    frequency = {}
    for number in numbers:
        if number in frequency:
            frequency[number] += 1
        else:
            frequency[number] = 1
    #Inicializacion de variables
    max_frequency = 0
    mode_value = None
    #Identifica el número con la frecuencia más alta
    for number, count in frequency.items():
        if count > max_frequency:
            max_frequency = count
            mode_value = number

    return mode_value    #Regresa el valor de la moda

#Cálculo de la varianza
def compute_variance(numbers, mean):
    total = 0.0                             #Inicializa la variable
total
    for number in numbers:                  #Sumatoria de la diferencias
de cuadrado
        total += (number - mean) ** 2
    return total / len(numbers)             #División entre el número
total de elementos
                                            #Regresa la varianza
#Cálculo de la desviacion estandar
def compute_standard_deviation(variance):
    #Cálculo de la desviación estandar a partir de la raiz cuadrada de
la varianza
    return math.sqrt(variance)              #Regresa la desviación
estandar

#Escritura de los resultados en archivo *.txt
def write_results(results):
    #Se utiliza ´w´para sobreescribir el archivo y utf-8 para
compatibilidad
    with open("StatisticsResults.txt", "w", encoding="utf-8") as file:
        for key, value in results.items():
            file.write(f"{key}: {value}\n")     #Escribe cada
estadística y salto de línea

#Entrada principal del programa, el cual se encarga de manejar la
lógica o secuencia de
#ejecución del programa
```

```
def main():

    start_time = time.perf_counter()  # Inicio de medición de tiempo

    # Manejo de argumentos del sistema
    if len(sys.argv) == 2:
        file_path = sys.argv[1]
    # Fallback para ejecución interactiva en Google Colab
    elif len(sys.argv) == 1:
        print("No input file provided as command-line argument. Using
default file 'TC1.txt'.")
        file_path = "TC2.txt" # Using an existing file from the
kernel's file system
    else:
        print("Usage: python computeStatistics.py <input_file>")
        sys.exit(1)
    # Lectura de los datos del archivo
    numbers = read_numbers_from_file(file_path)
    # Validación del contenido de los datos
    if not numbers:
        print("Error: No valid numeric data found or file was empty.")
        sys.exit(1)
    # LLamado de funciones, para obtener las estadísticas de los
datos.
    mean = compute_mean(numbers)                    # Cálculo de
promedio
    median = compute_median(numbers)                # Cálculo de
mediana
    mode = compute_mode(numbers)                    # Cálculo de moda
    variance = compute_variance(numbers, mean)      # Cálculo de
varianza
    std_dev = compute_standard_deviation(variance)  # Cálculo de
desviación estandar


    end_time = time.perf_counter()           # Fin de medición
    elapsed_time = end_time - start_time     # Tiempo total

    # Las estadisticas las colocamos en un diccionario, para su
respectiva impresión
    results = {
        "Count": len(numbers),
        "Mean": mean,
        "Median": median,
        "Mode": mode,
        "Variance": variance,
        "Standard Deviation": std_dev,
        "Execution Time (seconds)": elapsed_time,
    }
    # Impresión de los resultados en consola
```

```
    for key, value in results.items():
        print(f"{key}: {value}")
    # Escritura de los datos en archivo
    write_results(results)


# Asegura que el script solo se ejecuta si se llama directamente
# y no se importa como módulo en otro archivo.
if __name__ == "__main__":
    main()
```

```
Overwriting computeStatistics.py
```

Estadísticas del archivo TC1.txt

```
!python computeStatistics.py TC1.txt
```

```
Warning: Invalid data on line 400 -> '405s'
Count: 399
Mean: 241.91228070175438
Median: 239.0
Mode: 393.0
Variance: 21086.30558853275
Standard Deviation: 145.21124470416453
Execution Time (seconds): 0.0003251499997531937
```

Estadísticas del archivo TC2.txt

```
!python computeStatistics.py TC2.txt
```

```
Count: 1977
Mean: 250.7840161861406
Median: 247.0
Mode: 230.0
Variance: 20785.369132479238
Standard Deviation: 144.17131868884059
Execution Time (seconds): 0.001116120000006049
```

Estadísticas del archivo TC3.txt

```
!python computeStatistics.py TC3.txt
```

```
Count: 12624
Mean: 249.77621989860583
Median: 249.0
Mode: 94.0
Variance: 21117.27747316329
Standard Deviation: 145.31784980917962
Execution Time (seconds): 0.010451209000166273
```

Estadísticas del archivo TC4.txt

```
!python computeStatistics.py TC4.txt

Count: 12624
Mean: 149.00267347908746
Median: 147.75
Mode: 123.75
Variance: 17007.920843018837
Standard Deviation: 130.41441961308894
Execution Time (seconds): 0.006751970000095753
```

Estadísticas del archivo TC5.txt

```
!python computeStatistics.py TC5.txt

Warning: Invalid data on line 5 -> 'ABA'
Warning: Invalid data on line 155 -> '23,45'
Warning: Invalid data on line 232 -> '11;54'
Warning: Invalid data on line 239 -> 'll'
Count: 307
Mean: 241.49511400651465
Median: 241.0
Mode: 393.0
Variance: 21160.021963097748
Standard Deviation: 145.46484786056646
Execution Time (seconds): 0.00037849000000278465
```

Estadísticas del archivo TC6.txt

```
!python computeStatistics.py TC6.txt

Count: 3000
Mean: 1.8790659927977473e+20
Median: 1.88008049965543e+20
Mode: 1.27620004531949e+20
Variance: 1.1530904699530647e+40
Standard Deviation: 1.0738205017381e+20
Execution Time (seconds): 0.0018766200000754907
```

Estadísticas del archivo TC7.txt

```
!python computeStatistics.py TC7.txt

Warning: Invalid data on line 183 -> 'ABBA'
Warning: Invalid data on line 229 -> 'ERROR'
Count: 12767
Mean: 2.474673954997149e+20
Median: 2.4664097307429e+20
Mode: 1.57638329490099e+20
```

```
Variance: 2.0910793147136484e+40
Standard Deviation: 1.4460564700984703e+20
Execution Time (seconds): 0.007965049999711482
```

Instalación de Pylint

```
pip install pylint

Requirement already satisfied: pylint in
/usr/local/lib/python3.12/dist-packages (4.0.4)
Requirement already satisfied: astroid<=4.1.dev0,>=4.0.2 in
/usr/local/lib/python3.12/dist-packages (from pylint) (4.0.3)
Requirement already satisfied: dill>=0.3.6 in
/usr/local/lib/python3.12/dist-packages (from pylint) (0.3.8)
Requirement already satisfied: isort!=5.13,<8,>=5 in
/usr/local/lib/python3.12/dist-packages (from pylint) (7.0.0)
Requirement already satisfied: mccabe<0.8,>=0.6 in
/usr/local/lib/python3.12/dist-packages (from pylint) (0.7.0)
Requirement already satisfied: platformdirs>=2.2 in
/usr/local/lib/python3.12/dist-packages (from pylint) (4.5.1)
Requirement already satisfied: tomlkit>=0.10.1 in
/usr/local/lib/python3.12/dist-packages (from pylint) (0.13.3)
```

#Análisis del Código

```
!pylint computeStatistics.py

************* Module computeStatistics
computeStatistics.py:98:0: C0303: Trailing whitespace (trailing-
whitespace)
computeStatistics.py:124:0: C0303: Trailing whitespace (trailing-
whitespace)
computeStatistics.py:1:0: C0114: Missing module docstring (missing-
module-docstring)
computeStatistics.py:1:0: C0103: Module name "computeStatistics"
doesn't conform to snake_case naming style (invalid-name)
computeStatistics.py:16:0: C0116: Missing function or method docstring
(missing-function-docstring)
computeStatistics.py:38:0: C0116: Missing function or method docstring
(missing-function-docstring)
computeStatistics.py:45:0: C0116: Missing function or method docstring
(missing-function-docstring)
computeStatistics.py:57:0: C0116: Missing function or method docstring
(missing-function-docstring)
computeStatistics.py:77:0: C0116: Missing function or method docstring
(missing-function-docstring)
computeStatistics.py:84:0: C0116: Missing function or method docstring
(missing-function-docstring)
```

```
computeStatistics.py:89:0: C0116: Missing function or method docstring
(missing-function-docstring)
computeStatistics.py:97:0: C0116: Missing function or method docstring
(missing-function-docstring)

--------------------------------------------------------------------
Your code has been rated at 8.52/10 (previous run: 8.52/10, +0.00)
```

#Código Mejorado

```
## Esta linea nos permite guardar el codigo en
## un archivo físico que nombraremos computeStatistics

%%writefile computeStatistics.py

## Nos indica elinterprete que va a utilizar para
## Ejecutar el archivo, en este caso python3

#!/usr/bin/env python3

"""
Programa para el cálculo de estadísticas desde archivos de texto.

Este script procesa datos numéricos para obtener estadísticas como la
media,
mediana, moda, varianza y desviación estándar, cumpliendo con los
estándares
de limpieza y documentación de PEP 8. Los datos son obtenidos de
archivos *.txt.
"""

import sys    # Accesos a variables y funciones del interprete y del
sistema
import math   # Acceso a funciones matemáticas, indispensables para
estadísticas
import time   # Medición de tiempo de ejecución

##Función que abre un archivo, lo recorre línea por línea, convierte
cada palabra
##en un número decimal. Si encuentra algo diferente a un número, lo
ignora y avisa
##al usuario en lugar deterner el programa.

##Carga y valida datos numericos desde un archivo de texto
def read_numbers_from_file(file_path):
    """
    Carga y valida datos numéricos desde un archivo de texto.
    """
    numbers = []
```

```
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            for line_number, line in enumerate(file, start=1):
                value = line.strip()
                if not value:
                    continue
                try:
                    numbers.append(float(value))
                except ValueError:
                    print(
                        f"Warning: Invalid data on line {line_number}
-> '{value}'"
                    )
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
        sys.exit(1)

    return numbers

#Definición de la función para el cálculo del promedio
def compute_mean(numbers):
    """
    Calcula el promedio de una lista de valores númericos.
    """
    total = 0.0                    #Inicializa la variable total
    for number in numbers:         #Recorrido de la lista
        total += number            #Suma el valor actual al total
acumulado
    return total / len(numbers)    #Regresa el promedio

#Función para el cálculo de la mediana
def compute_median(numbers):
    """
    Calcula la mediana de una lista de valores númericos.
    """
    sorted_numbers = sorted(numbers)  #Ordena los elemenos de forma
ascendente
    count = len(sorted_numbers)       #Número de elementos en la lista
    mid = count // 2                  #Cálcula el punto medio

#Si el total de elementos es impar la mediana el número de en medio.
#Si es total de elementos es par, se toman dos valores centrales y se
promedia.
    if count % 2 == 0:
        return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2
    return sorted_numbers[mid]   #Regresa el valor de la mediana

#Cálculo de la moda
def compute_mode(numbers):
```

```
    """
    Calcula la moda de una lista de valores números.
    """
    #Diccionario para contar la frecuencia de cada número
    frequency = {}
    for number in numbers:
        if number in frequency:
            frequency[number] += 1
        else:
            frequency[number] = 1
    #Inicializacion de variables
    max_frequency = 0
    mode_value = None
    #Identifica el número con la frecuencia más alta
    for number, count in frequency.items():
        if count > max_frequency:
            max_frequency = count
            mode_value = number

    return mode_value    #Regresa el valor de la moda

#Cálculo de la varianza
def compute_variance(numbers, mean):
    """
    Calcula la varianza de una lista de valores números.
    """
    total = 0.0                             #Inicializa la variable
total
    for number in numbers:                  #Sumatoria de la diferencias
de cuadrado
        total += (number - mean) ** 2
    return total / len(numbers)             #División entre el número
total de elementos
                                            #Regresa la varianza
#Cálculo de la desviacion estandar
def compute_standard_deviation(variance):
    """
    Calcula la desviación estandar de una lista de valores números.
    """
    #Cálculo de la desviación estandar a partir de la raiz cuadrada de
la varianza
    return math.sqrt(variance)              #Regresa la desviación
estandar

#Escritura de los resultados en archivo *.txt
def write_results(results):
    """
    Escribe los resultados en un archivo de texto.
    """
    #Se utiliza ´w´para sobreescribir el archivo y utf-8 para
```

```
compatibilidad
    with open("StatisticsResults.txt", "w", encoding="utf-8") as file:
        for key, value in results.items():
            file.write(f"{key}: {value}\n")     #Escribe cada
estadística y salto de línea

#Entrada principal del programa, el cual se encarga de manejar la
lógica o secuencia de
#ejecución del programa
def main():
    """
    Función principal del programa.
    """
    start_time = time.perf_counter()  # Inicio de medición de tiempo

    # Manejo de argumentos del sistema
    if len(sys.argv) != 2:
        print("Usage: python computeStatistics.py <input_file>")
        sys.exit(1)

    file_path = sys.argv[1]
    numbers = read_numbers_from_file(file_path)

    # Validación del contenido de los datos
    if not numbers:
        print("Error: No valid numeric data found or file was empty.")
        sys.exit(1)

    # LLamado de funciones, para obtener las estadísticas de los
datos.
    mean = compute_mean(numbers)                    # Cálculo de
promedio
    median = compute_median(numbers)                # Cálculo de
mediana
    mode = compute_mode(numbers)                    # Cálculo de moda
    variance = compute_variance(numbers, mean)      # Cálculo de
varianza
    std_dev = compute_standard_deviation(variance)  # Cálculo de
desviación estandar

    end_time = time.perf_counter()           # Fin de medición
    elapsed_time = end_time - start_time     # Tiempo total

    # Las estadisticas las colocamos en un diccionario, para su
respectiva impresión
    results = {
        "Count": len(numbers),
        "Mean": mean,
        "Median": median,
        "Mode": mode,
```

```
        "Variance": variance,
        "Standard Deviation": std_dev,
        "Execution Time (seconds)": elapsed_time,
    }
    # Impresión de los resultados en consola
    for key, value in results.items():
        print(f"{key}: {value}")
    # Escritura de los datos en archivo
    write_results(results)


# Asegura que el script solo se ejecuta si se llama directamente
# y no se importa como módulo en otro archivo.
if __name__ == "__main__":
    main()
```

```
Overwriting computeStatistics.py
```

```
!python computeStatistics.py TC7.txt
```

```
Warning: Invalid data on line 183 -> 'ABBA'
Warning: Invalid data on line 229 -> 'ERROR'
Count: 12767
Mean: 2.474673954997149e+20
Median: 2.4664097307429e+20
Mode: 1.57638329490099e+20
Variance: 2.0910793147136484e+40
Standard Deviation: 1.4460564700984703e+20
Execution Time (seconds): 0.007784289000028366
```

#Análisis del codigo mejorado

```
!pylint computeStatistics.py
```

```
************* Module computeStatistics
computeStatistics.py:1:0: C0103: Module name "computeStatistics"
doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 9.87/10 (previous run: 9.87/10, +0.00)
```

#Conclusiones

El desarrollo del programa de estadísticas descriptivas a partir de archivos de texto nos permitio manejar y recordar conceptos fundamentales de programación, análisis de datos y calidad de software. Con la implementación de algoritmos básicos se cálculo la media, mediana, moda, varianza, y desviación estándar sin depender de librerías estadísticas externas, esto no permite comprender los fundamentos matemáticos, así como los algoritmos que se encuentran involucrados.

El programa maneja erroores que permiten identficar y reportar datos invalidos sin interrumpir la ejecución, garantizando la continuidad del procesamiento. De igual manera el programa tiene la capacidad de procesar archivos con cientos o miles de registros, con lo que podemos mencionar de cierta forma que la solución es escalable y adecuada para diferentes tamaños de datos.

El poder contar con la medición del tiempo de ejecución y la norma PEP8, ahcen que nos encontremos un poco más apegados a un enfoque profesional en el desarrollo de software. Contamos con prácticas de código legible, mantenible, y evaluable bajo ciertpos criterios de calidad, en base a la librería **pylint**.

este ejemplo, presenta un problema aparentemente simple que puede abordarse con buenas prácticas de ingeniería de software, permitiendonos desarrollar competencias para el desarrollo de algoritmos o aplicaciones con mayor grado de complejidad, con un enfoque tanto en análisis de datos e inteligencia artificial.

Los requisitos que se mencionan en el ejercicio son los siguientes:

1. El programa será invocado desde la línea de comandos. El programa recibirá un archivo como parámetro. El archivo contendrá una lista de elementos (presumiblemente números). **Se cumple con el requisito**
2. El programa calculará todas las estadísticas descriptivas a partir de un archivo de números. Los resultados se mostrarán en pantallas y en un archivo llamado StatisticsResults.txt. Todos los cálculos deben realizarse utilizando los algoritmos básicos, no funciones, ni bibliotecas. Las estadisticas descriptivas son: media, mediana, moda, desviación estandar y varianza. **Se cumple con el requisito**
3. El programa incluira un mecanismo para gestionar datos no válidos en el archivo. Los errores se mostrarán en consola y la ejecución continuará. **Se cumple con el requisito**
4. El nombre del programa será computeStatistics.py. **Se cumple con el requisito**
5. El formato mínimo para invocar el programa será el suguiente: python computeStatistics.py fileWithdata.txt. **Se cumple con el requsito**
6. El programa gestionará archivos con un número de elementos que puede variar desde cientos hasta miles. **Se cumple con el requisito**
7. El programa debe incluir al final de la ejecución el tiempo transcurrido para la ejecución y el cálculo de los datos. Este número se incluira en el archivo de resultados y en la pantalla. **Se cumple con el requisito**
8. Cumplir con la norma PEP8. **Se cumple con el requisito**

Nota, correct pylint en el código sin mejoras nos envia el siguiente mensaje **"Module name "computeStatistics" doesn't conform to snake_case naming style (invalid-name)"**, al correr el código mejorado, el mensaje se mantiene, sin embargo decidimos dejar el nombre del programa como se menciona en el requisito número 4.

Los tiempos de ejecucion de los diferentes archivos los mostramos a continuación:

- TC1.txt ----->> 0.0003251 segundos
- TC2.txt ----->> 0.0011161 segundos
- TC3.txt ----->> 0.0104512 segundos
- TC4.txt ----->> 0.0067519 segundos

- TC5.txt ----->> 0.0003784 segundos
- TC6.txt ----->> 0.0018766 segundos
- TC7.txt ----->> 0.0079650 segundos

La calificación el primer código fue de 8.55/10, mientras que el código mejorado presenta una calificación de 9.87/10. La mejorá para está ocasión fue de 1.32 puntos, consideramos que es buena.