Universidad Nacional de Ingeniería

Facultad de Ciencias

CC112(D)

Práctica Calificada 5

SUPERVISOR: Eduardo Yauri Lozano

ALUMNOS:

20240474A Valera Oviedo, Marcelo Jesus
20240683J Cavero Benites, Flavia Fernanda

martes, 3 de diciembre de 2024
2024-1

December 3, 2024

1 Introducción

En este informe se presenta un sistema de gestión de tareas implementado en Python. Este sistema permite a los usuarios agregar, editar, eliminar y buscar tareas, así como marcar tareas como completadas y ordenar las tareas por prioridad.

2 Desarrollo

2.1 Parte a

El desarrollo del sistema se llevó a cabo utilizando la programación orientada a objetos en Python. Se implementaron varias funciones para gestionar las tareas y una clase enumerada (`Enum`) para definir las prioridades.

2.2 Definición de la clase `Prioridad`

La clase `Prioridad` utiliza el módulo `Enum` de Python para definir tres niveles de prioridad: baja, media y alta.

```
from enum import Enum

class Prioridad(Enum):
    BAJA = 0
    MEDIA = 1
    ALTA = 2
```

Listing 1: Definición de la clase Prioridad

2.3 Definición de la clase `Tarea`

La clase `Tarea` define los atributos de una tarea: descripción, prioridad y estado de completada.

```
class Tarea:
    def __init__(self, descripcion, prioridad):
        self.descripcion = descripcion
        self.prioridad = prioridad
        self.completada = False
```

Listing 2: Definición de la clase Tarea

2.4 Funciones para gestionar tareas

Se implementaron varias funciones para agregar, editar, eliminar, buscar y marcar tareas como completadas. También se creó una función para mostrar las tareas ordenadas por prioridad.

```
#funcion para agregar tareas leyendo los atributos ingresados por el usuario
def agregar_tarea(tareas):
    descripcion = input("Ingrese la descripci n de la tarea: ")
    prioridad = int(input("Ingrese la prioridad de la tarea (0 = Baja, 1 = Media, 2 = Alta): "))
    tarea = Tarea(descripcion, Prioridad(prioridad))
    tareas.append(tarea)

#funcion que busca la tarea por el  ndice  ingresado y cambia el atributo completada
def marcar_completada(tareas):
    indice = int(input("Ingrese el  ndice  de la tarea a marcar como completada: "))
    if 0 <= indice < len(tareas):
        tareas[indice].completada = True
    else:
        print(" ndice  inv lido.")

#funcion que ordena la lista de tareas por prioridad y las imprime
def mostrar_tareas(tareas):
    tareas_ordenadas = sorted(tareas, key=lambda x: x.prioridad.value, reverse=True)
    for i, tarea in enumerate(tareas_ordenadas):
        print(f"{i}. Descripci n: {tarea.descripcion}, Prioridad: {tarea.prioridad.name}, Completada: {'S ' if tarea.completada else 'No'}")

#funcion actualiza la descripci n y la prioridad ingresadas por el usuario
def editar_tarea(tareas):
    indice = int(input("Ingrese el  ndice  de la tarea a editar: "))
    if 0 <= indice < len(tareas):
        descripcion = input("Ingrese la nueva descripci n: ")
        prioridad = int(input("Ingrese la nueva prioridad (0 = Baja, 1 = Media, 2 = Alta): "))
        tareas[indice].descripcion = descripcion
        tareas[indice].prioridad = Prioridad(prioridad)
    else:
        print(" ndice  inv lido.")

#funcion que elimina un elemento del arreglo de tareas
def eliminar_tarea(tareas):
    indice = int(input("Ingrese el  ndice  de la tarea a eliminar: "))
    if 0 <= indice < len(tareas):
        tareas.pop(indice)
    else:
        print(" ndice  inv lido.")

```

```
#funcion que busca una tarea mediante el indice ingresado por el usuario
def buscar_tarea(tareas):
    keyword = input("Ingrese una palabra clave para buscar tareas: ")
    for i, tarea in enumerate(tareas):
        if keyword.lower() in tarea.descripcion.lower():
            print(f"{i}. Descripci n: {tarea.descripcion}, Prioridad: {tarea.prioridad.name}, Completada: {'S ' if tarea.completada else 'No'}")
```

Listing 3: Funciones para gestionar tareas

2.5 Función principal

La función principal (`main`) maneja la interacción con el usuario y permite ejecutar las diferentes funcionalidades del sistema de gestión de tareas.

```
def main():
    tareas = []  #creamos un arreglo vac o de tareas que se ir n llenando
    while True:
        print("\n1. Agregar tarea")
        print("2. Marcar tarea como completada")
        print("3. Ordenar y mostrar tareas por prioridad")
        print("4. Editar tarea")
        print("5. Eliminar tarea")
        print("6. Buscar tarea")
        print("7. Salir")
        opcion = int(input("Ingrese una opci n: "))

        if opcion == 1:
            agregar_tarea(tareas)
        elif opcion == 2:
            marcar_completada(tareas)
        elif opcion == 3:
            mostrar_tareas(tareas)
        elif opcion == 4:
            editar_tarea(tareas)
        elif opcion == 5:
            eliminar_tarea(tareas)
        elif opcion == 6:
            buscar_tarea(tareas)
        elif opcion == 7:
            print("Saliendo...")
            break
        else:
            print("Opci n no v lida.")

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

Listing 4: Función principal

3 Conclusiones

El sistema de gestión de tareas desarrollado en Python permite a los usuarios manejar sus tareas de manera eficiente. La implementación del sistema en Python utilizando programación orientada a objetos y funciones específicas facilita su uso y mantenimiento.

4 Documentación del Código

- **Clase** `Prioridad`: Define tres niveles de prioridad (baja, media, alta).
- **Clase** `Tarea`: Define los atributos de una tarea (descripción, prioridad, completada).
- **Funciones**:

– $\texttt{agregar}_{t}area$: $Agregaunanuevatareaalalista.$

- **Función principal** (`main`): Maneja la interacción con el usuario y permite ejecutar las diferentes funcionalidades del sistema.

5 Desarrollo

5.0.1 Parte b

En este análisis, procesamos un conjunto de datos de ventas electrónicas para extraer información útil, realizar análisis descriptivos y detectar valores atípicos. Utilizamos Python y varias bibliotecas como pandas, numpy, matplotlib y seaborn.

6 Carga de Datos

Leemos los datos desde un archivo CSV utilizando pandas.

```
import pandas as pd
df = pd.read_csv('electronica_ventas.csv')
```

7 Procesamiento de Fechas

Convertimos la columna `Fecha` a un tipo de datos datetime.

```
df['Fecha'] = pd.to_datetime(df['Fecha'], errors='coerce')
```

8 Detección de Valores Nulos

Verificamos si hay valores nulos en la columna `Fecha` después de la conversión.

```
if df['Fecha'].isnull().sum() > 0:
    print("Hay valores nulos en la columna 'Fecha'")
```

9 Cálculo de Ingresos

Calculamos los ingresos multiplicando las ventas por el precio.

```
df['Ingresos'] = df['Ventas'] * df['Precio']
```

10 Creación de la Columna Mes

Creamos una nueva columna que representa el mes de cada registro de ventas.

```
df['Mes'] = df['Fecha'].dt.month
```

11 Análisis Descriptivo

Mostramos estadísticas descriptivas del dataset.

```
print(df.describe())
```

12 Detección de Valores Atípicos

Identificamos valores atípicos en la columna `Ventas`.

```
q1 = df['Ventas'].quantile(0.25)
q3 = df['Ventas'].quantile(0.75)
iqr = q3 - q1
outliers = df[(df['Ventas'] < (q1 - 1.5 * iqr)) | (df['Ventas'] > (q3 + 1.5 * iqr))]
print(outliers)
```

13 Visualizaciones

Generamos varias visualizaciones para entender mejor los datos.

13.1 Distribución de Precios

Creamos un histograma para visualizar la distribución de los precios.

```
plt.figure(figsize=(8, 6))
plt.hist(df['Precio'], bins=20, color='purple', edgecolor='black')
plt.title('Distribución de Precios')
plt.xlabel('Precio')
plt.ylabel('Frecuencia')
plt.grid()
plt.show()
```

13.2 Relación entre Ventas e Ingresos

Generamos un diagrama de dispersión para mostrar la relación entre ventas e ingresos.

```
plt.figure(figsize=(8, 6))
plt.scatter(df['Ventas'], df['Ingresos'], alpha=0.5)
plt.title('Relación entre Ventas e Ingresos')
plt.xlabel('Ventas')
plt.ylabel('Ingresos')
plt.grid()
plt.show()
```

13.3 Ventas por Año

Creamos un gráfico de barras para mostrar las ventas por año.

```
df['Año'] = df['Fecha'].dt.year
ventas_anuales = df.groupby('Año')['Ventas'].sum()
plt.figure(figsize=(8, 6))
ventas_anuales.plot(kind='bar', title='Ventas por Año')
plt.ylabel('Ventas')
plt.xlabel('Año')
plt.grid()
plt.show()
```

13.4 Promedio de Precios por Producto

Creamos un gráfico de barras para mostrar el promedio de precios por producto.

```
promedio_precio_producto = df.groupby('Producto')['Precio'].mean()
plt.figure(figsize=(8, 6))
promedio_precio_producto.plot(kind='bar', color='green', title='Promedio de Precios por Proc
plt.ylabel('Precio Promedio')
plt.grid()
plt.show()
```

13.5 Ventas por Categoría

Si la columna `Categoría` existe, creamos un gráfico de barras para mostrar las ventas por categoría.

```
if 'Categoría' in df.columns:
    ventas_categoria = df.groupby('Categoría')['Ventas'].sum()
    plt.figure(figsize=(8, 6))
    ventas_categoria.plot(kind='bar', color='blue', title='Ventas por Categoría')
    plt.ylabel('Ventas')
    plt.grid()
    plt.show()
```

13.6 Evolución Mensual de Ingresos

Creamos un gráfico de líneas para mostrar la evolución mensual de ingresos.

```
ingresos_mensuales = df.groupby('Mes')['Ingresos'].sum()
plt.figure(figsize=(8, 6))
ingresos_mensuales.plot(kind='line', marker='o', color='red', title='Evolución Mensual de In
plt.ylabel('Ingresos')
plt.xlabel('Mes')
plt.grid()
plt.show()
```

13.7 Desviación Estándar de Ventas por Región

Creamos un gráfico de barras para mostrar la desviación estándar de ventas por región.

```
std_ventas_region = df.groupby('Region')['Ventas'].std()
plt.figure(figsize=(8, 6))
std_ventas_region.plot(kind='bar', color='cyan', title='Desviación Estándar de Ventas por Re
plt.ylabel('Desviación Estándar')
plt.grid()
plt.show()
```

13.8 Mapa de Calor de Correlación

Creamos un mapa de calor para visualizar la correlación entre todas las variables numéricas.

```
plt.figure(figsize=(10, 8))
numerical_df = df.select_dtypes(include=[np.number])
sns.heatmap(numerical_df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Mapa de Calor de Correlación')
plt.show()
```

14 Conclusiones

A través de este análisis, hemos podido obtener una comprensión más profunda de los datos de ventas electrónicas. Las visualizaciones generadas proporcionan una visión clara de las tendencias y patrones presentes en el conjunto de datos.