

# Instrucciones Copilot (S1 : req4, req6)

## Usuario

Hola requiero un script de python que lea la información de un archivo de documentación markdown y la muestre en consola, como ves el .md se divide en capítulos (titulares h2 o ##) requiero que el programa solicite al usuario la elección de un capítulo debe tener mensaje de bienvenida y despedida así como opción para finalizar el ciclo de peticiones al usuario, adapta automáticamente la ruta de ingreso del archivo.

## Copilot

Excelente

Tu requerimiento está claro: quieres un programa en Python que:

1. Lea el contenido del archivo Markdown ( .md ).
2. Identifique los capítulos (líneas que empiezan con ## ).
3. Permita al usuario elegir uno para mostrar su contenido.
4. Tenga un mensaje de bienvenida y despedida.
5. Funcione en bucle hasta que el usuario decida salir.

Aquí tienes el **script Python completo** y comentado, listo para ejecutarse:

```
# programa: leer_markdown.py
# Descripción: Lee el archivo Documentacion.md y permite al usuario elegir un
capítulo (##) para visualizar su contenido.

def leer_markdown(ruta_archivo):
    """Lee el archivo markdown y devuelve su contenido línea por línea."""
    with open(ruta_archivo, 'r', encoding='utf-8') as archivo:
        return archivo.readlines()

def obtener_capitulos(lineas):
    """Devuelve un diccionario con los títulos de nivel 2 (##) y su
    contenido."""
    capitulos = {}
    titulo_actual = None
    contenido_actual = []

    for linea in lineas:
        if linea.startswith("## "):
```

```

        # Si ya había un capítulo previo, lo guardamos antes de iniciar el
nuevo
        if titulo_actual:
            capitulos[titulo_actual] = "".join(contenido_actual).strip()
            contenido_actual = []
            titulo_actual = linea.strip().replace("## ", "")
        elif titulo_actual:
            contenido_actual.append(linea)

# Guarda el último capítulo
if titulo_actual:
    capitulos[titulo_actual] = "".join(contenido_actual).strip()

return capitulos

def mostrar_menu(capitulos):
    """Muestra el menú de capítulos disponibles."""
    print("\nCapítulos disponibles:\n")
    for i, capitulo in enumerate(capitulos.keys(), 1):
        print(f"{i}. {capitulo}")
    print(f"{len(capitulos) + 1}. Salir")

def main():
    print("👋 Bienvenido al lector de capítulos del archivo Documentacion.md
    📖")

    ruta = "Proyecto Aurelion/Documentacion.md"

    try:
        lineas = leer_markdown(ruta)
        capitulos = obtener_capitulos(lineas)

        if not capitulos:
            print("⚠️ No se encontraron capítulos (##) en el archivo.")
            return

        while True:
            mostrar_menu(capitulos)
            opcion = input("\nSeleccione un capítulo por número: ")

            try:
                opcion = int(opcion)
            except ValueError:
                print("❌ Por favor ingrese un número válido.\n")
                continue

```

```

        if opcion == len(capitulos) + 1:
            print("\n👋 Gracias por usar el lector de Markdown. ¡Hasta pronto!")
            break
        elif 1 <= opcion <= len(capitulos):
            titulo = list(capitulos.keys())[opcion - 1]
            print(f"\n--- {titulo} ---\n")
            print(capitulos[titulo])
            print("\n" + "-" * 50 + "\n")
        else:
            print("⚠️ Opción fuera de rango.\n")

    except FileNotFoundError:
        print("❌ No se encontró el archivo README.md en el directorio actual.")

if __name__ == "__main__":
    main()

```

## Usuario

indicame como verifico si los valores de una columna son unicos

## Copilot

Para verificar si los valores de una columna son únicos en un DataFrame de pandas, hay varias formas. Te muestro las más comunes:

### 1. Usando el método `is_unique`:

```

# Retorna True si todos los valores son únicos, False si hay duplicados
columna_es_unica = df_clientes['nombre_columna'].is_unique
print(columna_es_unica)

```

### 2. Usando `nunique()` vs `len()`:

```

# Compara el número de valores únicos con el total de registros
total_valores = len(df_clientes['nombre_columna'])
valores_unicos = df_clientes['nombre_columna'].nunique()
print(f"Total valores: {total_valores}")
print(f"Valores únicos: {valores_unicos}")
print(f"¿Son todos únicos?: {total_valores == valores_unicos}")

```

### 3. Para ver los valores duplicados:

```
# Muestra las filas que tienen valores duplicados en la columna
duplicados = df_clientes[df_clientes['nombre_columna'].duplicated()]
print(duplicados)
```

4. Para contar cuántas veces se repite cada valor:

```
# Muestra un conteo de cada valor
conteo = df_clientes['nombre_columna'].value_counts()
print(conteo)
```

 **Usuario**

recuérdame como hago un heatmap de correlaciones

 **Copilot**

Usa seaborn sobre la matriz de correlación de las columnas numéricas. Ejemplos cortos:

```
numeric = aurelion.select_dtypes(include=['int64', 'float64'])
corr = numeric.corr()

# heatmap simple
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", vmin=-1, vmax=1,
linewidths=.5)
plt.title("Matriz de correlación")
plt.show()
```

```
# heatmap con máscara (oculta triángulo superior)
import numpy as np
mask = np.triu(np.ones_like(corr, dtype=bool))

plt.figure(figsize=(10,8))
sns.heatmap(corr, mask=mask, annot=True, fmt=".2f", cmap="vlag", center=0,
linewidths=.5)
plt.title("Heatmap (triángulo superior oculto)")
plt.show()
```

 **Usuario**

recuérdame como hago boxplot con seaborn



Ejemplos cortos para hacer boxplots con seaborn (reemplaza aurelion/columnas según tu DataFrame):

```
# 1) Boxplot simple de una columna numérica
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=aurelion['importe'].dropna())
plt.title('Boxplot de importe')
plt.show()
```

```
# 2) Boxplot por categoría (comparar distribución por grupo)
sns.boxplot(x='categoria', y='importe', data=aurelion)
plt.xticks(rotation=45)
plt.title('Importe por categoría')
plt.tight_layout()
plt.show()
```

```
# 3) Horizontal + ocultar outliers (showfliers=False)
sns.boxplot(x='importe', y='categoria', data=aurelion, showfliers=False)
plt.title('Importe por categoría (sin outliers)')
plt.tight_layout()
plt.show()
```

```
# 4) Grid de boxplots por otra variable con catplot (facets)
sns.catplot(kind='box', x='categoria', y='importe', col='medio_pago',
            data=aurelion, col_wrap=3, sharey=True)
plt.tight_layout()
plt.show()
```



**Usuario**

recuerdame como hago boxplot con seaborn



Ejemplos cortos para hacer boxplots con seaborn (reemplaza aurelion/columnas según tu DataFrame):

```
# 1) Boxplot simple de una columna numérica
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.boxplot(x=aurelion['importe'].dropna())
plt.title('Boxplot de importe')
plt.show()
```

```
# 2) Boxplot por categoría (comparar distribución por grupo)
sns.boxplot(x='categoria', y='importe', data=aurelion)
plt.xticks(rotation=45)
plt.title('Importe por categoría')
plt.tight_layout()
plt.show()
```

```
# 3) Horizontal + ocultar outliers (showfliers=False)
sns.boxplot(x='importe', y='categoria', data=aurelion, showfliers=False)
plt.title('Importe por categoría (sin outliers)')
plt.tight_layout()
plt.show()
```

```
# 4) Grid de boxplots por otra variable con catplot (facets)
sns.catplot(kind='box', x='categoria', y='importe', col='medio_pago',
            data=aurelion, col_wrap=3, sharey=True)
plt.tight_layout()
plt.show()
```

## Usuario

ayudame con un grafico de dispersion de la relacion entre precio y cantidad vendida

## Copilot

```
plt.figure(figsize=(8,6))
sns.scatterplot(x='precio_unitario', y='cantidad', data=aurelion,
               hue='categoria', alpha=0.7)
plt.title('Relación entre precio unitario y cantidad vendida')
plt.xlabel('Precio unitario')
plt.ylabel('Cantidad vendida')
plt.tight_layout()
plt.show()
```

```
correlacion = aurelion['precio_unitario'].corr(aurelion['cantidad'])
print(f"📊 Correlación precio vs cantidad: {correlacion:.2f}")
```

## Usuario

necesito generar una grafica donde se representen los producto con menor rotación

## Copilot

```
productos_menos_vendidos = ventas_por_producto.tail(10)
print("Productos con menor rotación:")
print(productos_menos_vendidos)

plt.figure(figsize=(10,6))
sns.barplot(x='cantidad', y='producto_comprado',
data=productos_menos_vendidos, palette='rocket')
plt.title('Productos con menor rotación')
plt.xlabel('Cantidad vendida')
plt.ylabel('Producto')
plt.tight_layout()
plt.show()
```

## Usuario

teniendo en cuenta lo que llevo avanzado, como prosigo? cual es el flujo de trabajo?

## Copilot

Flujo de trabajo propuesto (breve, accionable):

1. Reafirmar objetivo — elige UNA tarea prioritaria:
  - Regresión: predecir importe por venta.
  - Series temporales: forecast de ingresos (agregar por fecha).
  - Segmentación: RFM + KMeans para clientes.
  - Detección de anomalías: aislar outliers en importe.
2. Data quality & limpieza (completar / documentar):
  - Ver valores nulos y tipos (ya hecho).
  - Decidir qué hacer con outliers (cap, eliminar o mantener).
  - Confirmar integridad de identificadores/fechas.
3. Feature engineering (crear variables que el modelo necesite):
  - Fechas: day/month/weekday/recency.

- Agregados por cliente si la tarea lo requiere (freq, total, avg).
- One-hot / target encoding para categóricas; escalar numéricas según el modelo.

#### 4. Baseline modeling + validación:

- Split train/test temporal o aleatorio según la tarea.
- Crear baseline simple (mean predictor para regresión; KMeans para segmentación).
- Evaluar con métricas claras (MAE/RMSE/R2 para regresión, MAPE para series, silhouette para clustering).

#### 5. Mejora y selección de modelos:

- Probar modelos más robustos (RandomForest, XGBoost/LightGBM; Prophet/ARIMA para series).
- Cross-validation y búsqueda de hiperparámetros.
- Interpretabilidad: feature importance, SHAP si aplica.

#### 6. Entrega: documentación, visualizaciones clave, guardado de artefactos (model.pkl), notebook limpio y reproducible.

#### 7. Producción / seguimiento: pipeline de inferencia, pruebas, monitorización de performance.

Ejemplo rápido — si quieres empezar por segmentación RFM, añade esta celda (crea df RFM y KMeans 3 clusters):

```
# 4. Sección de aplicación de algoritmos de Machine Learning

### 4.1 Segmentación de clientes con KMeans (RFM)

# Importar librerías necesarias
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Calcular RFM por cliente
# Referencia de fecha: última fecha de venta en dataset
reference_date = df_aurelion['fecha_venta'].max() + pd.Timedelta(days=1)

# Crear tabla RFM
rfm = df_aurelion.groupby('nombre_cliente').agg({
    'fecha_venta': lambda x: (reference_date - x.max()).days, # Recency (días desde última compra)
    'nombre_cliente': 'count', # Frequency (número de transacciones)
    'importe': 'sum' # Monetary (suma de importes)
}).rename(columns={
    'fecha_venta': 'recency',
    'nombre_cliente': 'frequency',
```



```

    'importe': 'monetary'
}).reset_index()

print("=== Tabla RFM ===")
print(rfm.head(10))
print(f"\nDimensiones RFM: {rfm.shape}")
print("\nEstadísticas RFM:")
print(rfm.describe())

# Preparar datos para KMeans (normalización)
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm[['recency', 'frequency', 'monetary']])

print("\n=== Datos normalizados (primeras 5 filas) ===")
print(rfm_scaled[:5])

# Determinar número óptimo de clusters (método del codo)
inertias = []
silhouette_scores = []
k_range = range(2, 11)

for k in k_range:
    kmeans_temp = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans_temp.fit(rfm_scaled)
    inertias.append(kmeans_temp.inertia_)
    silhouette_scores.append(silhouette_score(rfm_scaled,
kmeans_temp.labels_))

# Visualizar método del codo
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Gráfico de inercia
axes[0].plot(k_range, inertias, 'bo-', linewidth=2, markersize=8)
axes[0].set_xlabel('Número de clusters (k)')
axes[0].set_ylabel('Inercia')
axes[0].set_title('Método del Codo')
axes[0].grid(True, alpha=0.3)

# Gráfico de silhouette
axes[1].plot(k_range, silhouette_scores, 'ro-', linewidth=2, markersize=8)
axes[1].set_xlabel('Número de clusters (k)')
axes[1].set_ylabel('Silhouette Score')
axes[1].set_title('Silhouette Score por k')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()

```

```

plt.show()

# Aplicar KMeans con k=3 (puedes ajustar según gráficos)
k_optimal = 3
kmeans = KMeans(n_clusters=k_optimal, random_state=42, n_init=10)
rfm['cluster'] = kmeans.fit_predict(rfm_scaled)

print(f"\n=== KMeans aplicado con k={k_optimal} ===")
print(f"Silhouette Score: {silhouette_score(rfm_scaled, rfm['cluster']):.4f}")
print(f"Davies-Bouldin Index: {davies_bouldin_score(rfm_scaled,
rfm['cluster']):.4f}")

# Distribución de clientes por cluster
print("\nDistribución de clientes por cluster:")
print(rfm['cluster'].value_counts().sort_index())

# Perfilar cada cluster
print("\n=== Perfil de cada cluster ===")
cluster_profile = rfm.groupby('cluster')[['recency', 'frequency',
'monetary']].agg(['mean', 'median', 'min', 'max'])
print(cluster_profile)

# Visualizar clusters (gráfico 3D simulado con pairplot)
plt.figure(figsize=(10, 6))
scatter = plt.scatter(rfm['recency'], rfm['frequency'], c=rfm['cluster'],
                      s=rfm['monetary']/5, cmap='viridis', alpha=0.6,
                      edgecolors='black')
plt.xlabel('Recency (días)')
plt.ylabel('Frequency (transacciones)')
plt.title('Segmentación de clientes KMeans (tamaño = Monetary)')
plt.colorbar(scatter, label='Cluster')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Visualizar distribución de clusters por característica
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

axes[0].boxplot([rfm[rfm['cluster']==i]['recency'].values for i in
range(k_optimal)],
                labels=[f'Cluster {i}' for i in range(k_optimal)])
axes[0].set_ylabel('Recency (días)')
axes[0].set_title('Recency por Cluster')
axes[0].grid(True, alpha=0.3)

axes[1].boxplot([rfm[rfm['cluster']==i]['frequency'].values for i in

```

```

range(k_optimal)],
        labels=[f'Cluster {i}' for i in range(k_optimal)])
axes[1].set_ylabel('Frequency (transacciones)')
axes[1].set_title('Frequency por Cluster')
axes[1].grid(True, alpha=0.3)

axes[2].boxplot([rfm[rfm['cluster']==i]['monetary'].values for i in
range(k_optimal)],
        labels=[f'Cluster {i}' for i in range(k_optimal)])
axes[2].set_ylabel('Monetary (importe total)')
axes[2].set_title('Monetary por Cluster')
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Interpretación de clusters
print("\n=== Interpretación de Clusters ===")
for i in range(k_optimal):
    cluster_data = rfm[rfm['cluster']==i]
    print(f"\nCluster {i} ({len(cluster_data)} clientes):")
    print(f" - Recency promedio: {cluster_data['recency'].mean():.1f} días")
    print(f" - Frequency promedio: {cluster_data['frequency'].mean():.1f} transacciones")
    print(f" - Monetary promedio: ${cluster_data['monetary'].mean():.2f}")

```

## Usuario

como elegir valor optimo de k según gráfico de silueta?

## Copilot

```

# ...existing code...
# Determinar k óptimo por silhouette (ya tienes inertias y silhouette_scores)
import numpy as np
best_k = list(k_range)[int(np.argmax(silhouette_scores))]
print(f"Mejor k por silhouette: {best_k} (silhouette=
{max(silhouette_scores):.4f})")

# comparar k candidatos cercanos
for k in k_range:
    kmeans_tmp = KMeans(n_clusters=k, random_state=42,
n_init=10).fit(rfm_scaled)
    labels_tmp = kmeans_tmp.labels_
    s = silhouette_score(rfm_scaled, labels_tmp)

```

```

    sizes = np.bincount(labels_tmp)
    print(f"k={k}: silhouette={s:.4f}, tamaños={sizes}")

# Usar best_k
k_optimal = best_k
kmeans = KMeans(n_clusters=k_optimal, random_state=42, n_init=10)
rfm['cluster'] = kmeans.fit_predict(rfm_scaled)

print(f"\n=== KMeans aplicado con k={k_optimal} ===")
print(f"Silhouette Score: {silhouette_score(rfm_scaled, rfm['cluster']):.4f}")
print(f"Davies-Bouldin Index: {davies_bouldin_score(rfm_scaled,
rfm['cluster']):.4f}")
print("\nDistribución de clientes por cluster:")
print(rfm['cluster'].value_counts().sort_index())

# Perfil y verificación rápida
cluster_profile = rfm.groupby('cluster')[['recency', 'frequency',
'monetary']].agg(['mean', 'median', 'min', 'max', 'count'])
print("\nPerfil de cada cluster:")
print(cluster_profile)
# ...existing code...

```

## Usuario

```

=== Interpretación de Clusters === Cluster 0 (22 clientes): - Recency
promedio: 124.5 días - Frequency promedio: 3.2 transacciones - Monetary
promedio: $23425.09 Cluster 1 (12 clientes): - Recency promedio: 31.1 días -
Frequency promedio: 7.8 transacciones - Monetary promedio: $56831.33 Cluster 2
(3 clientes): - Recency promedio: 12.7 días - Frequency promedio: 12.0
transacciones - Monetary promedio: $104460.33 Cluster 3 (6 clientes): -
Recency promedio: 127.2 días - Frequency promedio: 9.0 transacciones -
Monetary promedio: $65942.33 Cluster 4 (20 clientes): - Recency promedio: 34.1
días - Frequency promedio: 4.1 transacciones - Monetary promedio: $29522.95

```

que nombres de segmentación le pondrías a dichos cluster basados en metodo rfm (recency, frequency, monetary) y que estrategias promocionales me sugieres, actúa como experto en marketing y fidelización

## Copilot

Aquí tienes **nombres de segmentación RFM profesionales**, adecuados para tus datos, y **estrategias promocionales** claras para cada grupo.

Los nombres están basados en prácticas reales de *Customer Segmentation*, *CRM* y *Lifecycle Marketing*.

---

# Interpretación y Nombres de Segmentos RFM + Estrategias

## Cluster 2 — "Clientes VIP / Súper Leales"

**Recency:** muy reciente (12 días)

**Frequency:** altísima (12)

**Monetary:** altísimo (104.460)

### Perfil

Tus mejores clientes. Compran seguido, reciente y gastan más que todos.

### Estrategias

- **Programa VIP exclusivo:** acceso anticipado a productos, descuentos especiales.
- **Ofertas personalizadas premium** basadas en su historial.
- **Regalos por lealtad** (cupones, muestras gratuitas).
- **Contacto directo por WhatsApp o asesoría personalizada.**
- **Invitarlos a referir amigos** (tienen más probabilidad de hacerlo).

---

## Cluster 1 — "Leales de Alto Valor"

**Recency:** reciente (31 días)

**Frequency:** alta (7.8)

**Monetary:** alto (56.831)

### Perfil

Clientes valiosos que compran con frecuencia pero no al nivel VIP.

### Estrategias

- **Upselling y cross-selling** basado en su categoría favorita.
- **Promos moderadas** (ej.: 10–15% por compra recurrente).
- **Email de “gracias por tu lealtad” + cupón especial.**
- **Reforzar programas de puntos.**
- **Contenidos de valor** (tips, guías, recomendaciones).

---

## **Cluster 4 — "Clientes Activos de Valor Medio"**

**Recency:** reciente (34 días)

**Frequency:** moderada (4.1)

**Monetary:** medio (29.522)

### **Perfil**

Clientes activos, compran con cierta recurrencia y buen ticket promedio.

### **Estrategias**

- **Campañas de activación:** "vuelve y recibe un 10%".
- **Recomendaciones personalizadas** para aumentar la frecuencia.
- **Combos o paquetes** para incrementar el ticket.
- **Recordatorios suaves** ("Hace un mes no compras, mira estas novedades").

---

## **Cluster 3 — "Grandes Gastadores Infrecuentes" (High Monetary / Low Recency)**

**Recency:** *muy* bajo (127 días → mucho tiempo sin comprar)

**Frequency:** alta (9)

**Monetary:** alto (65.942)

### **Perfil**

Clientes que **antes eran TOP**, pero llevan mucho sin comprar.

Valiosos, pero están en riesgo.

### **Estrategias**

- **Campañas de reactivación agresivas:** 20–30% off por regreso.
- **Notificar lanzamientos nuevos** (pueden ser sensibles a la novedad).
- **Ofertas de "solo por hoy"** para generar urgencia.
- **Encuesta corta** para entender por qué se alejaron.

**Meta:** reactivarlos antes de que se pierdan definitivamente.

---

# **Cluster 0 — "Clientes Dormidos de Bajo Valor"**

**Recency:** viejo (124 días)

**Frequency:** baja (3.2)

**Monetary:** bajo (23.425)

## **Perfil**

Clientes con poco valor y que llevan mucho sin comprar.

## **Estrategias**

- **Campañas de reactivación masivas de bajo costo** (mailing general).
- **Descuentos amplios o cupones** (para incrementar conversión).
- **Mostrarles productos económicos** (barreras de entrada más bajas).
- **Remarketing digital** con ofertas.
- Si no responden → **segmentarlos para campañas pasivas**, evitando gasto excesivo.

Cluster	Nombre	Tipo de cliente	Acción clave
2	VIP / Súper Leales	Máximo valor	Retener con exclusividad
1	Leales de Alto Valor	Frecuentes y rentables	Fidelizar + upselling
4	Activos de Valor Medio	Recientes, moderados	Activar más frecuencia
3	Grandes Gastadores Infrecuentes	Alto gasto pero inactivos	Reactivar urgente
0	Dormidos de Bajo Valor	Bajo valor, inactivos	Recuperar con campañas masivas