

UNIVERSIDAD ANDINA DEL CUSCO
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



APRENDIZAJE NO SUPERVISADO -DBSCAN, FP

Docente : Mg.Ing. HUGO ESPETIA HUAMANGA

Presentado por: ANAYA RODRIGUEZ SEBASTIAN RODRIGO
DEL CASTILLO MOLINA GINO GALOIS
HUANCA PILARES, JEAN CARLOS
ZAMATA SOTEC JAMIL
ZAVALETA HANDA FERNADO HIROSHI

CUSCO-PERÚ
2024

Introducción

Este informe analiza el dataset *datos_casa_cusco.csv*, que incluye información sobre 2000 departamentos en Cusco y sus alrededores. Las variables clave abarcan el precio, locación, área construida y características de las propiedades.

Se llevó a cabo un proceso de limpieza y preprocesamiento de datos, junto con la generación de datos sintéticos mediante inteligencia artificial. Se utilizaron técnicas de agrupamiento con el algoritmo DBSCAN para identificar clusters y el algoritmo FP-Growth para descubrir relaciones entre variables. A través de este análisis, buscamos ofrecer una comprensión clara de los patrones en el mercado inmobiliario de Cusco.

.

Descripción del Dataset

El dataset utilizado para este proyecto, *datos_casa_cusco.csv*, contiene información detallada de 2000 viviendas en diversas zonas de Cusco y áreas cercanas. Las columnas incluidas son:

- Precio: Valor en soles de la vivienda.
- Locación: Ubicación geográfica de la propiedad (San Jerónimo, Cusco, Wanchaq, etc.).
- Área: Metros cuadrados construidos.
- Dormitorios: Cantidad de habitaciones.
- Baños: Cantidad de baños.
- Piso: Nivel del piso donde se encuentra la propiedad.
- Zona: Clasificación del tipo de zona (Urbanización, Residencial, Avenida, etc.).

Se llevó a cabo un análisis exploratorio para entender las distribuciones y patrones dentro de los datos.

Limpieza y Preprocesamiento de los Datos

El dataset contenía algunas variables categóricas que fueron convertidas a variables numéricas mediante codificación one-hot. Además:

- Se eliminaron las filas con valores nulos.
- Se aplicaron transformaciones a las variables categóricas Locación y Zona para garantizar su uso en los modelos de machine learning.

El preprocesamiento se realizó con el objetivo de obtener un dataset limpio y listo para ser utilizado en la implementación de los modelos.

390820	San Jeronimo	76	2	1	3	Urbanización
303990	San Jeronimo	69	2	1	4	Residencial
306990	Cusco	84	3	2	1	Avenida
684500	Cusco	103	3	2	2	Calle
525400	Cusco	75	3	2	2	Barrio
572605	San Sebastian	120	3	2	3	Urbanización
517500	San Jeronimo	110	3	2	2	Urbanización
440000	Wanchaq	107	3	2	2	Residencial
628000	Cusco	112	3	1	2	Calle
536250	Cusco	102	3	2	3	Calle
993750	Cusco	177	3	2	2	Urbanización
1200000	Cusco	235	3	5	7	Condominio
348750	Cusco	90	3	1	5	Residencial
573750	Cusco	109	3	2	4	Avenida
342000	San Jeronimo	102	3	2	5	Residencial
506250	Santiago	100	3	3	6	Residencial
1350000	Santiago	265	4	3	2	Residencial
300000	Santiago	75	3	1	2	Urbanización
918750	Wanchaq	146	3	3	2	Urbanización
900000	Wanchaq	168	3	3	5	Urbanización
354375	San Sebastian	85	3	2	3	Urbanización
562500	San Sebastian	115	3	2	4	Barrio
525000	San Jeronimo	118	3	2	3	Urbanización
278640	Wanchaq	78	2	1	4	Barrio
285000	San Sebastian	82	2	2	2	Urbanización

El dataset se amplió con datos sintéticos con ayuda de la IA chat gpt, para poder generar 2000 datos en nuestra base de datos a la cual llamamos datos_casas_cusco.csv

1. Agrupamiento (Clustering)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN es un algoritmo de agrupamiento basado en densidad, que identifica clusters en grandes conjuntos de datos según la densidad de puntos en el espacio. A diferencia de otros métodos de clustering (como K-means), no necesita que se especifique el número de clusters con anticipación.

¿Cómo funciona DBSCAN?

- Epsilon (ϵ): Radio que define una región alrededor de cada punto. Si otros puntos están dentro de este radio, se consideran vecinos.
- MinPts: Número mínimo de puntos que debe tener una región para considerarse un cluster.
- **Puntos Core, Border y Noise:**
 - Core Points: Puntos que tienen al menos MinPts vecinos dentro de su radio ϵ .
 - Border Points: Puntos que están dentro del radio ϵ de un punto core pero que tienen menos de MinPts vecinos.
 - Noise Points: Puntos que no pertenecen a ningún cluster (no son puntos core ni border).

Características clave:

- Identificación de ruido: DBSCAN marca los puntos que no encajan bien en ningún cluster como ruido, permitiendo una mejor identificación de patrones.
- Clusters de forma arbitraria: Es eficaz para detectar clusters de cualquier forma, ya que se basa en la densidad de puntos.
- No requiere especificar el número de clusters. A diferencia de K-means, el número de clusters se deriva del parámetro ϵ .

Ventajas:

- Flexibilidad en formas de clusters: DBSCAN detecta clusters de formas arbitrarias, como clusters no esféricos o con distribuciones complejas.
- Detección automática de ruido.
- No requiere predefinir el número de clusters.

Desventajas:

- Elección sensible de parámetros: Elegir un valor inapropiado para ϵ o MinPts puede afectar significativamente el rendimiento del algoritmo.

- Problemas en datos de alta dimensión: En espacios de alta dimensión, puede ser difícil encontrar una buena distancia ϵ debido a la "maldición de la dimensionalidad."

Aplicaciones:

- Detección de anomalías en datos espaciales.
- Análisis de agrupamiento en datos geoespaciales.
- Agrupamiento de datos de redes sociales para identificar comunidades.

APLICACIÓN

1. Base de datos de departamentos en cusco .csv

Precio,Locacion,Arean2,NroDormitorios,NroBanos,NroPisos,Zona		
289311, San Sebastian,125,2,1,6,Urb		
529492, San Jeronimo,150,2,1,6,Urb		
529820, Saylla,111,2,1,1,Institucion		

2. Importar pandas y librerías de clustering

```
# Instalar scikit-learn si no lo tienes instalado
!pip install -U scikit-learn

# Importar las librerías necesarias
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import seaborn as sns
```

3. Cargar archivos

```
df = pd.read_csv("datos_casas_cusco.csv")
print(df.head())
```

	Precio	Locacion	Arean2	NroDormitorios	NroBanos	NroPisos	\
0	289311	San Sebastian	125	2	1	6	
1	529492	San Jeronimo	150	2	1	6	
2	529820	Saylla	111	2	1	1	
3	612894	Santiago	249	3	3	8	
4	778810	Wanchaq	242	5	5	3	
		Zona					
0		Urbanizacion					
1		Urbanizacion					
2		Institucion					

4. Normalización de datos

```

# Seleccionar las columnas relevantes para el clustering
columnas_relevantes = ['Precio', 'Aream2', 'NroDormitorios', 'NroBanos', 'NroPisos']
datos = df[columnas_relevantes]

# Normalizar los datos para que tengan media 0 y varianza 1
scaler = StandardScaler()
datos_normalizados = scaler.fit_transform(datos)

# Mostrar un resumen de los datos normalizados
print("Datos normalizados:\n", datos_normalizados[:5])

```

Datos normalizados:

```

[[-1.34612217 -0.39489029 -1.34232    -1.43231906  0.36498958]
 [-0.65314851 -0.01691804 -1.34232    -1.43231906  0.36498958]
 [-0.65220216 -0.60655476 -1.34232    -1.43231906 -1.53106019]
 [-0.41251587  1.47985209 -0.45248241 -0.01942726  1.12340949]

```

5. Aplicar DBSCAN para la generación de clusters

```

# Configurar y aplicar DBSCAN
dbscan = DBSCAN(eps=0.85, min_samples=5) # Ajusta estos parámetros según sea necesario
etiquetas = dbscan.fit_predict(datos_normalizados)

# Agregar las etiquetas de los clusters al DataFrame original
df['Cluster'] = etiquetas

# Ver cuántos clusters se han encontrado (ignora el ruido, que es -1)
n_clusters = len(set(etiquetas)) - (1 if -1 in etiquetas else 0)
print(f"Número de clusters encontrados: {n_clusters}")
print(df['Cluster'].value_counts())

```

Número de clusters encontrados: 6

```

Cluster
2      506
0      486
3      482
1      469
-1      51
5         3
4         3
Name: count, dtype: int64

```

6. Generación de gráficos

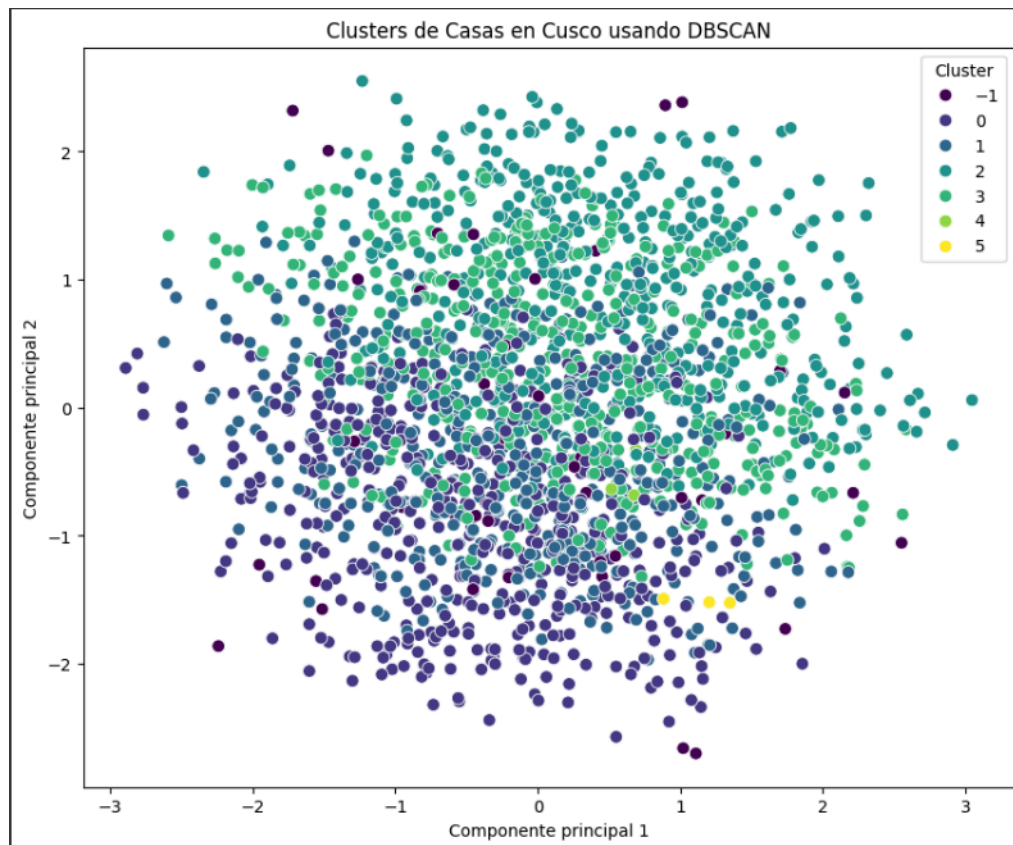
```

from sklearn.decomposition import PCA

# Reducir la dimensionalidad a 2D usando PCA para visualización
pca = PCA(n_components=2)
datos_reducidos = pca.fit_transform(datos_normalizados)

# Crear un scatter plot de los clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(x=datos_reducidos[:, 0], y=datos_reducidos[:, 1], hue=df['Cluster'], palette='viridis', s=60)
plt.title('Clusters de Casas en Cusco usando DBSCAN')
plt.xlabel('Componente principal 1')
plt.ylabel('Componente principal 2')
plt.legend(title='Cluster')
plt.show()

```



7. Estadísticas descriptivas por cluster

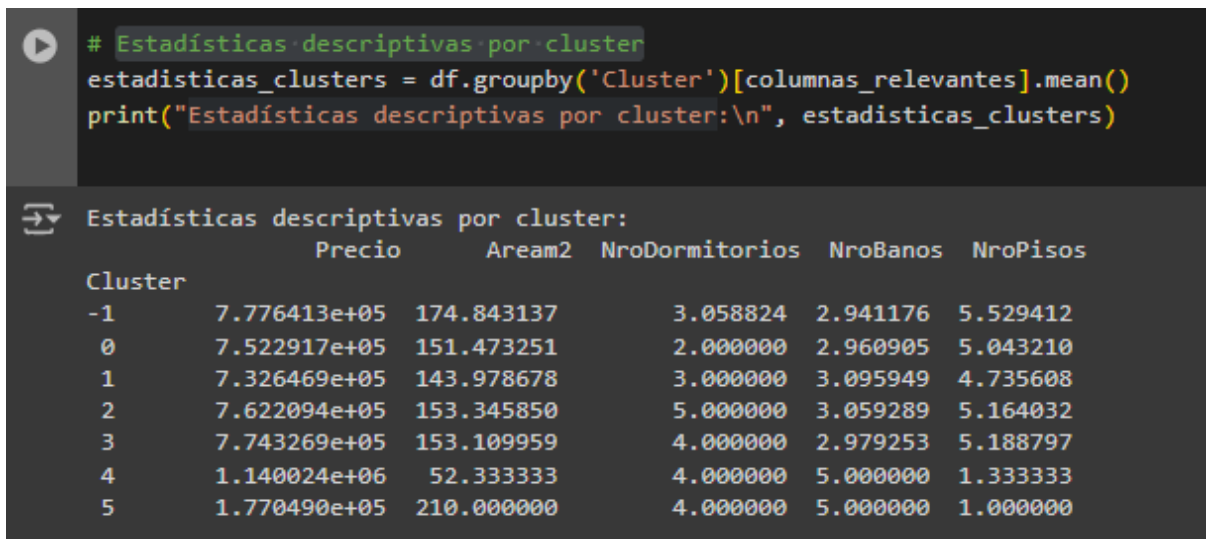
Estas estadísticas descriptivas muestran los promedios de las características de las casas en cada cluster, según el análisis de DBSCAN. Cada fila representa un cluster y cada columna muestra la media de una característica (Precio, Aream2, NroDormitorios, NroBaños y NroPisos). Vamos a interpretar los resultados:

- Cluster -1 (Ruido) Precio promedio: 777,641.3 Área promedio: 174.8 m² Número de dormitorios: 3.06 Número de baños: 2.94 Número de pisos: 5.53 Este cluster representa el "ruido" o los puntos que no pudieron ser agrupados en clusters principales. Estas propiedades tienden a tener un precio alto, un área grande y un número elevado de pisos. Podrían ser casas que no se ajustan a un patrón común en los otros clusters.
- Cluster 0 Precio promedio: 752,291.7 Área promedio: 151.5 m² Número de dormitorios: 2 Número de baños: 2.96 Número de pisos: 5.04 Las casas en este cluster tienen un precio intermedio, con un área relativamente grande y suelen tener 2 dormitorios. El número de baños y pisos es alto, lo que podría indicar que estas casas están diseñadas para aprovechar mejor el espacio vertical.
- Cluster 1 Precio promedio: 732,646.9 Área promedio: 144 m² Número de dormitorios: 3 Número de baños: 3.1 Número de pisos: 4.74 Aquí, el precio y

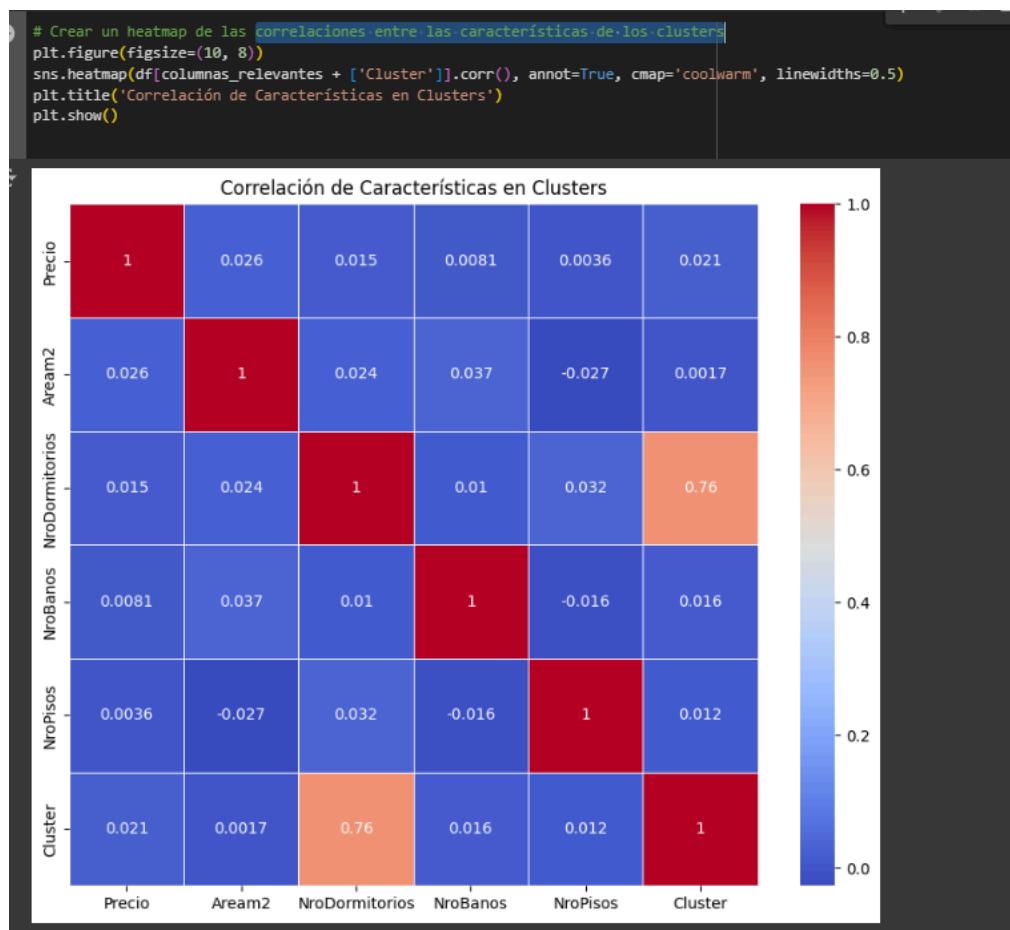
el área son un poco más bajos que en el cluster 0, pero estas casas tienen un promedio de 3 dormitorios y 3 baños. El número de pisos es ligeramente menor.

- Cluster 2 Precio promedio: 762,209.4 Área promedio: 153.3 m² Número de dormitorios: 5 Número de baños: 3.06 Número de pisos: 5.16 Las casas en este cluster tienden a ser más grandes, con un promedio de 5 dormitorios, lo que podría indicar propiedades más adecuadas para familias grandes. El precio también es más alto, y el número de pisos es elevado.
- Cluster 3 Precio promedio: 774,326.9 Área promedio: 153.1 m² Número de dormitorios: 4 Número de baños: 2.98 Número de pisos: 5.19 Las casas en el cluster 3 tienen características intermedias con un promedio de 4 dormitorios y 3 baños, similar al cluster 2 en cuanto a área, pero con un precio ligeramente superior.
- Cluster 4 Precio promedio: 1,140,024 Área promedio: 52.3 m² Número de dormitorios: 4 Número de baños: 5 Número de pisos: 1.33 Estas casas son más caras, pero tienen un área mucho menor en comparación con otros clusters. Sin embargo, tienen muchos baños (5 en promedio), lo que sugiere que están diseñadas con un enfoque especial, posiblemente viviendas lujosas o con un propósito específico, como oficinas o alojamientos.
- Cluster 5 Precio promedio: 177,049 Área promedio: 210 m² Número de dormitorios: 4 Número de baños: 5 Número de pisos: 1 Las propiedades en este cluster son notablemente más baratas y tienen un área grande, lo que podría indicar que son viviendas más sencillas o en ubicaciones menos cotizadas, pero con características adecuadas para familias grandes (4 dormitorios, 5 baños).

Conclusiones Cluster -1 representa el ruido, con propiedades que no encajan bien en otros clusters. Clusters 0, 1, 2 y 3 tienen características similares en cuanto a precio, área y número de habitaciones, con algunas diferencias sutiles. Clusters 4 y 5 parecen ser casos extremos con características muy específicas: uno con precios muy altos y un área pequeña (posiblemente propiedades de lujo), y el otro con precios bajos pero un área grande (posiblemente viviendas más económicas).



8. correlaciones entre las características de los clusters



link colab

https://colab.research.google.com/drive/1klu7o5uJOm11qG_HoLU88HRK3J3BySgw?usp=sharing#scrollTo=_jf

3. Asociación

Modelos de Asociación (Algoritmo Apriori o FP-Growth)

Los algoritmos de reglas de asociación, como FP-Growth, son técnicas que buscan descubrir relaciones interesantes entre variables en grandes bases de datos. Son especialmente útiles para análisis de transacciones, como en supermercados, donde se busca encontrar qué productos suelen comprarse juntos (ej. el análisis de "market basket").

¿Cómo funciona FP-Growth?

- FP-Growth Construye una estructura de árbol compacta (FP-tree) para evitar el recuento reiterado de itemsets frecuentes, mejorando la eficiencia en grandes bases de datos.

Características clave:

- Análisis exploratorio de grandes bases de datos: Proporciona reglas que pueden ayudar a comprender mejor los patrones ocultos.
- Eficiencia en bases de datos grandes: FP-Growth es más eficiente manejando grandes bases de datos.

Ventajas:

Mejora la eficiencia en bases de datos grandes, ya que evita la generación innecesaria de conjuntos de ítems.

Desventajas:

Aunque más rápido, su implementación es más compleja y requiere más memoria.

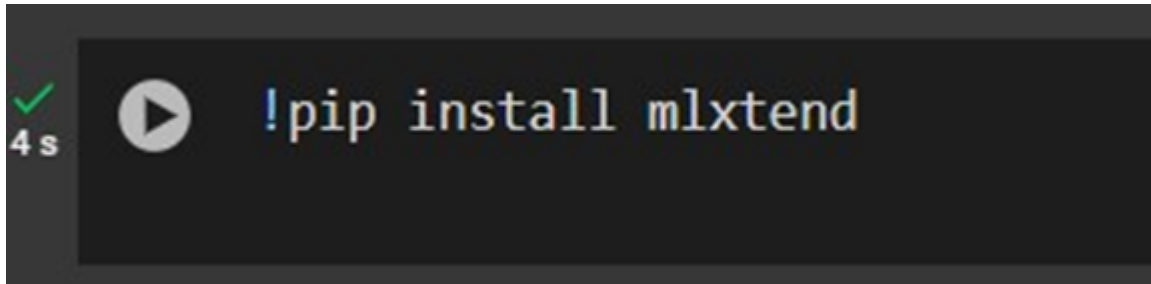
Aplicaciones:

- Análisis de ventas en retail: Descubre qué productos suelen comprarse juntos, para promociones o disposición de productos en tiendas.
- Optimización de catálogos: Ayuda a comprender qué productos tienden a ser solicitados juntos para mejorar el stock y las ofertas.
- Medicina: Descubre correlaciones entre síntomas y enfermedades o entre tratamientos y resultados.

APLICACIÓN

1. Instalar las librerías necesarias

Usaremos la librería mlxtend que tiene implementaciones útiles para minería de patrones frecuentes, incluido FP-Growth. Además, trabajaremos con pandas para cargar y manipular los datos



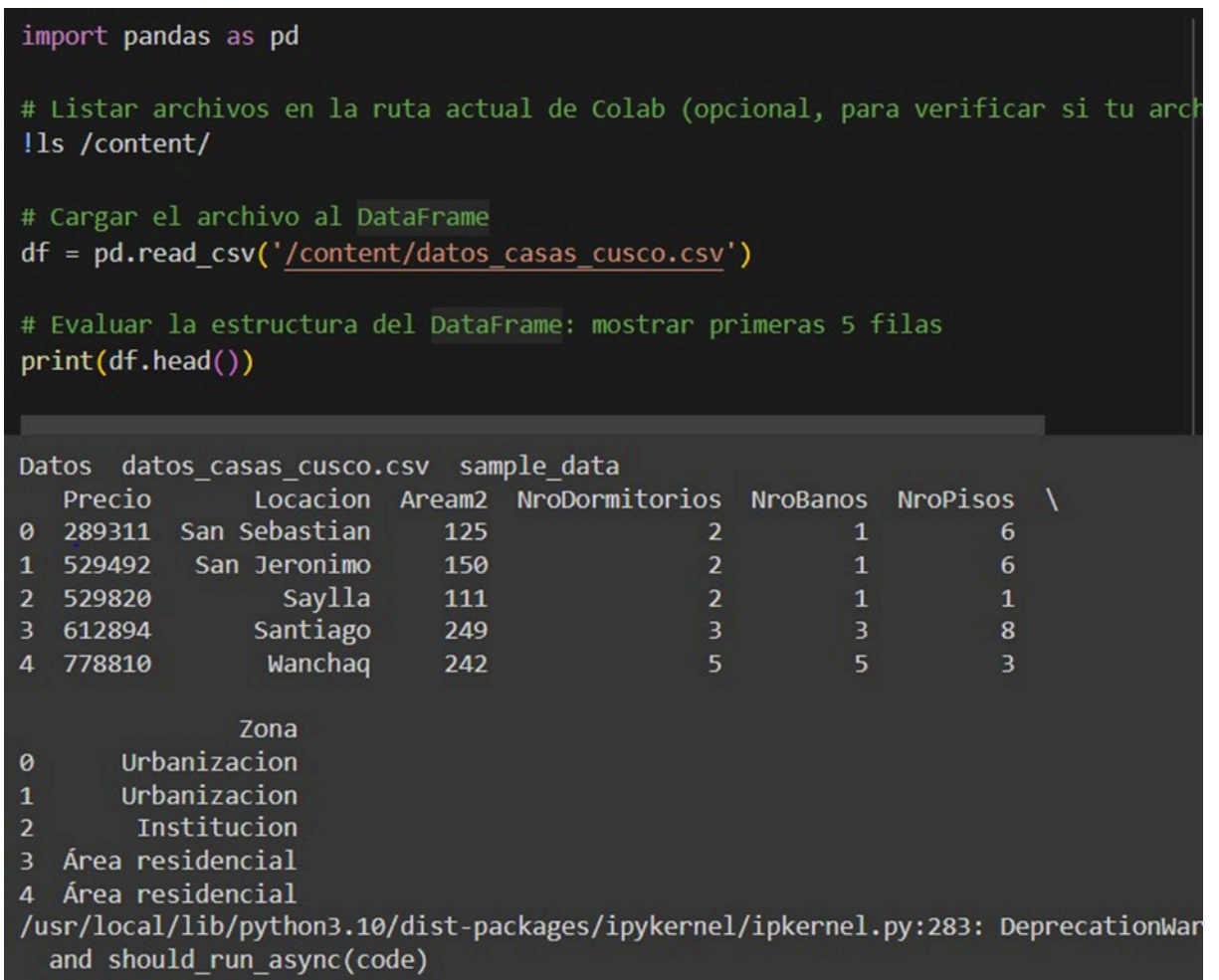
```
✓ 4 s !pip install mlxtend
```

2. Importar las librerías



```
✓ 0 s [2] import pandas as pd
      from mlxtend.frequent_patterns import fpgrowth, association_rules
```

3. Cargar los archivos que vamos a usar



```
import pandas as pd

# Listar archivos en la ruta actual de Colab (opcional, para verificar si tu arch
!ls /content/

# Cargar el archivo al DataFrame
df = pd.read_csv('/content/datos_casas_cusco.csv')

# Evaluar la estructura del DataFrame: mostrar primeras 5 filas
print(df.head())
```

	Datos datos_casas_cusco.csv			sample_data			
	Precio	Locacion	Arean2	NroDormitorios	NroBanos	NroPisos	\
0	289311	San Sebastian	125	2	1	6	
1	529492	San Jeronimo	150	2	1	6	
2	529820	Saylla	111	2	1	1	
3	612894	Santiago	249	3	3	8	
4	778810	Wanchaq	242	5	5	3	

```

      Zona
0  Urbanizacion
1  Urbanizacion
2  Institucion
3  Área residencial
4  Área residencial
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
and should_run_async(code)
```

Cargamos los archivos y verificamos que estén correctamente cargados haciendo en print de los primeros 4 datos de la tabla

4. Convertir los datos a formato binario

Como estamos trabajando con categorías (e.g., Locación, Zona) y algunos valores numéricos, podemos transformar estas columnas en categorías binarias, lo que permitirá aplicar FP-Growth.

```
[6] # Convertir columnas categóricas (Locacion y Zona) a formato binario
df_binarizado = pd.get_dummies(df[['Locacion', 'Zona']])

# Opcional: Binarizar los datos numéricos en rangos específicos
df_binarizado['Area_Grande'] = (df['Arean2'] > 150).astype(int)
df_binarizado['Muchos_Dormitorios'] = (df['NroDormitorios'] >= 3).astype(int)

# Visualizamos los primeros datos binarizados
print(df_binarizado.head())
```

```

Locacion_Cusco  Locacion_San Jeronimo  Locacion_San Sebastian \
0              False                  False                  True
1              False                  True                   False
2              False                  False                  False
3              False                  False                  False
4              False                  False                  False

Locacion_Santiago  Locacion_Saylla  Locacion_Wanchaq  Zona_Avenida \
0              False              False              False          False
1              False              False              False          False
2              False              True               False          False
3              True               False              False          False
4              False              False              True           False

Zona_Barrio  Zona_Calle  Zona_Condominio  Zona_Institucion  Zona_Paradero \
0          False        False              False              False          False
1          False        False              False              False          False
2          False        False              False              True           False
3          False        False              False              False          False
4          False        False              False              False          False

Zona_Parque  Zona_Pasaje  Zona_Residencial  Zona_Urbanizacion \
```

5. Aplicar FP-Growth

```
[7] from mlxtend.frequent_patterns import fpgrowth, association_rules

# Ejecutar FP-Growth con soporte mínimo del 20% (0.2)
patrones_frecuentes = fpgrowth(df_binarizado, min_support=0.2, use_colnames=True)

# Mostrar los patrones frecuentes encontrados
print(patrones_frecuentes)

# Generar reglas de asociación con un umbral de confianza del 50%
reglas = association_rules(patrones_frecuentes, metric="confidence", min_threshold=0.5)

# Mostrar las reglas de asociación
print(reglas)
```

```

support      itemsets
0  0.7485      (Muchos_Dormitorios)
1  0.4905      (Area_Grande)
2  0.3675  (Area_Grande, Muchos_Dormitorios)

  antecedents  consequents  antecedent support \
0  (Area_Grande)  (Muchos_Dormitorios)      0.4905

  consequent support  support  confidence  lift  leverage  conviction \
0              0.7485  0.3675   0.749235  1.000983  0.000361  1.002933

  zhangs_metric
0      0.001927
```

6. Generación de gráficos para explicar más detalladamente

Gráficos de barras

Muestra el soporte de cada patrón frecuente para ver cuáles combinaciones de ítems ocurren más.

```
# Generar un gráfico de barras de los soportes de los patrones frecuentes
plt.figure(figsize=(10, 6))
patrones_frecuentes.plot(kind='bar', x='itemsets', y='support', legend=False, ax=plt)
plt.title('Patrones Frecuentes - Gráfico de Barras')
plt.xlabel('Conjuntos de Ítems')
plt.ylabel('Soporte')
plt.xticks(rotation=45, ha='right')
plt.show()
```

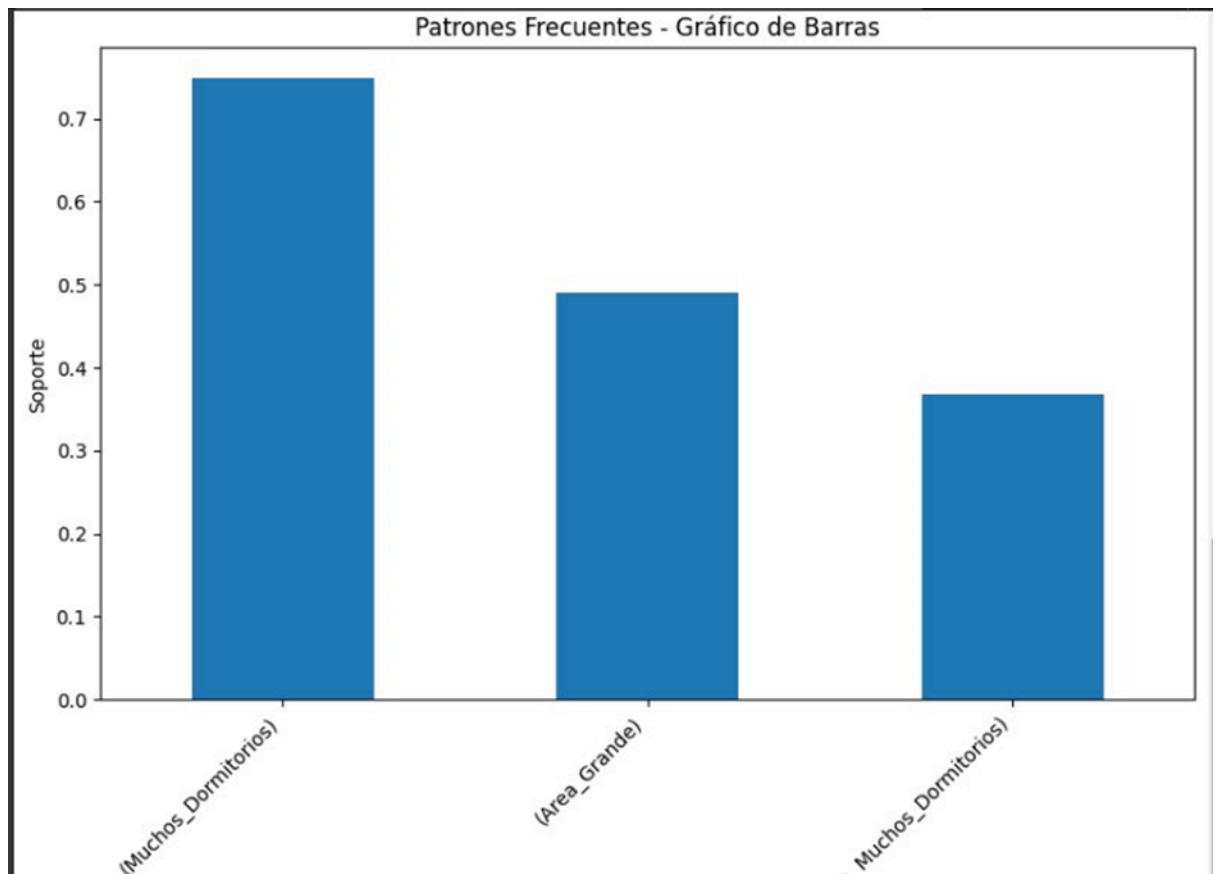


Gráfico de pastel

Representa la distribución del soporte de los patrones frecuentes como proporciones del total.

```
# Gráfico de pastel para mostrar distribución de soportes
plt.figure(figsize=(8, 8))
plt.pie(patrones_frecuentes['support'], labels=patrones_frecuentes['itemsets'], autopct='%1.1f%%')
plt.title('Distribución de Soporte de los Patrones Frecuentes')
plt.show()
```

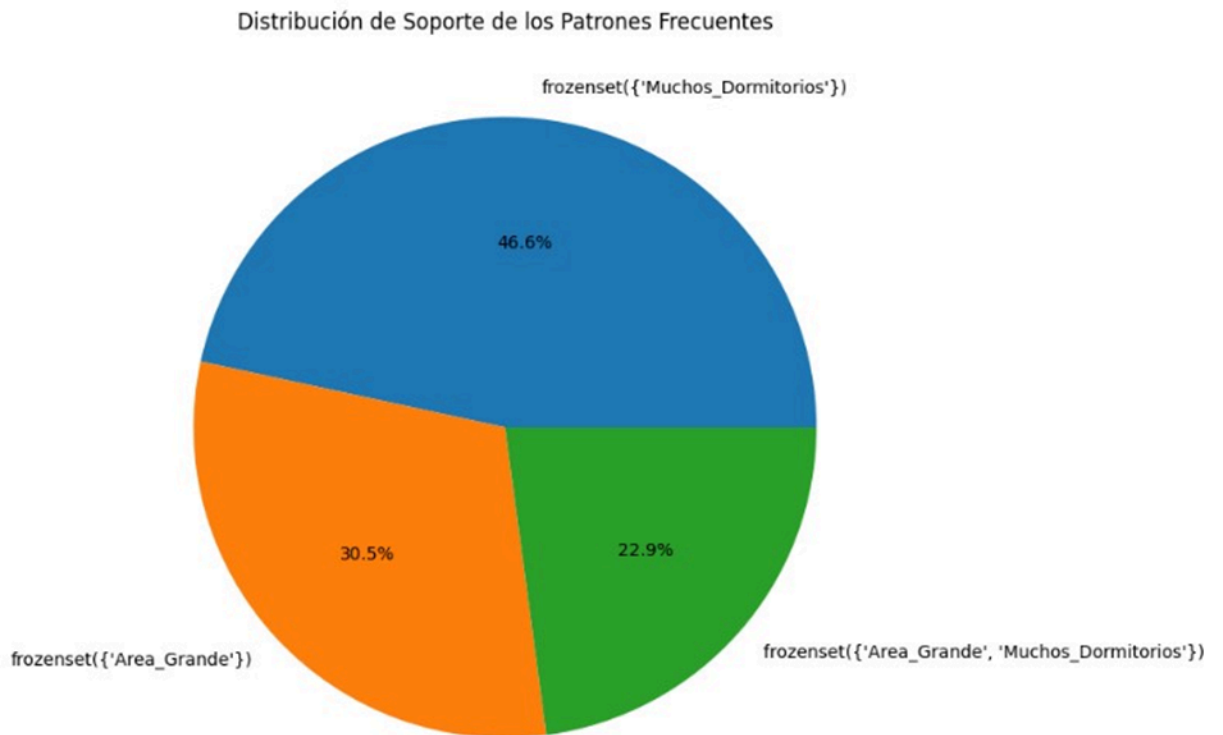
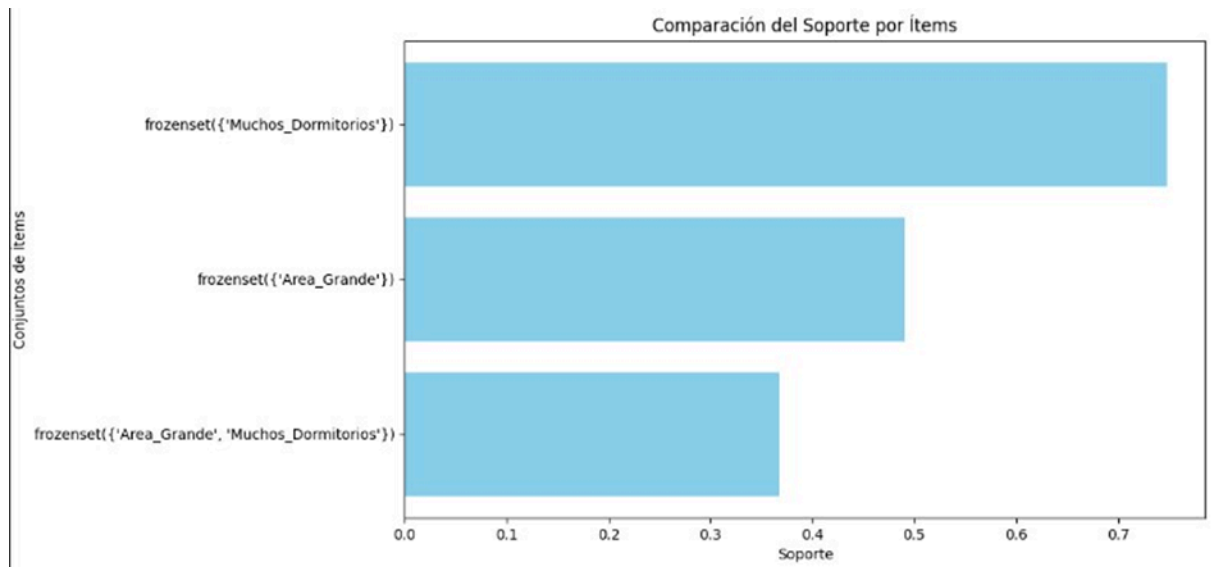



Gráfico comparativo

Compara los ítems más frecuentes para identificar cuáles tienen mayor peso en los patrones.

```
# Gráfico comparativo entre los ítems y su soporte
plt.figure(figsize=(10, 6))
plt.barh(patrones_frecuentes['itemsets'].astype(str), patrones_frecuentes['support'])
plt.title('Comparación del Soporte por Ítems')
plt.xlabel('Soporte')
plt.ylabel('Conjuntos de Ítems')
plt.gca().invert_yaxis() # Invertir el eje para una mejor lectura
plt.show()
```

Estos gráficos te permiten:

Identificar patrones frecuentes en los datos (cuáles ocurren más).

Visualizar la distribución del soporte (cuáles dominan y cuáles son minoritarios). Comparar patrones de forma clara para destacar combinaciones relevantes.

link colab

https://colab.research.google.com/drive/17wtMBh9FEaEoRRDy-50Fo7_oBXM8E__S?auth=1#scrollTo=6A3bUzwNF9a6

Proyecto

<https://github.com/JeanCar2003/APRENDIZAJE-NO-SUPERVISADO--DBSCAN-FP/tree/main/APRENDIZAJE%20NO%20SUPERVISADO%20-DBSCAN%2C%20FP>