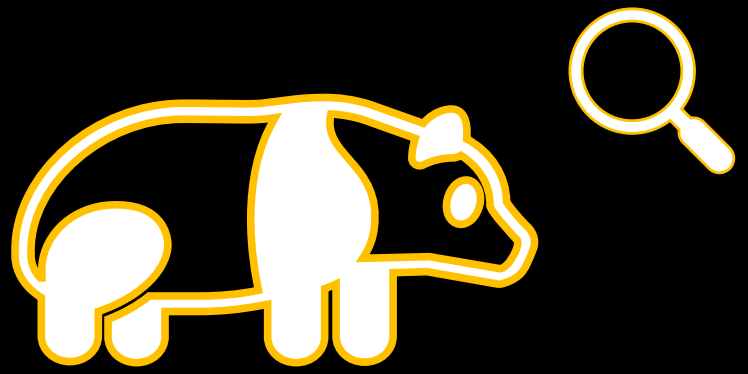
Análisis exploratorio con pandas



Por Alma Ahumada

¿Qué son los pandas?

Pandas es una biblioteca open-source de Python diseñada para el análisis y manipulación de datos. Permite trabajar con información en forma de tablas de manera sencilla y eficiente, similar a Excel pero con mucha más potencia. Está construida sobre NumPy y se ha convertido en una herramienta esencial para científicos de datos.

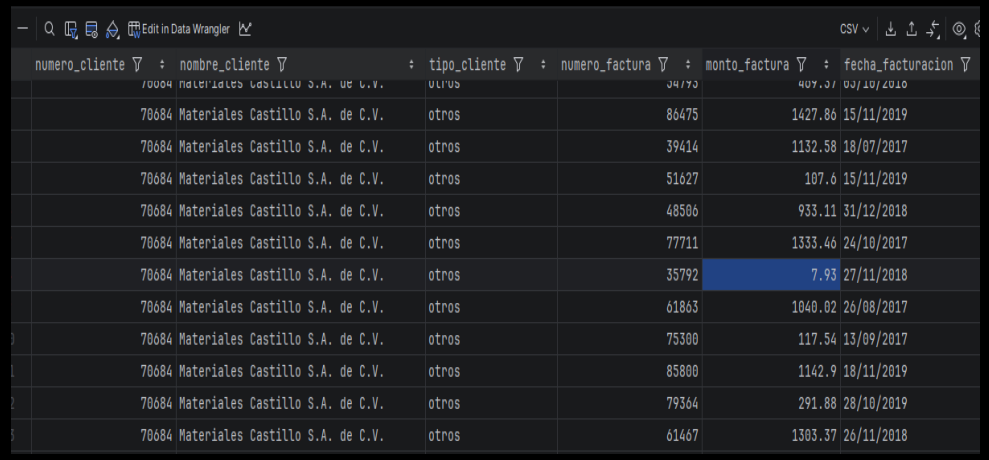
Sus estructuras principales son las Series (una columna) y los DataFrames (tablas completas). Con Pandas es fácil cargar archivos, limpiar datos, filtrar, agrupar y preparar información para análisis o machine learning.

¿Qué es el Análisis Exploratorio de Datos (EDA)?

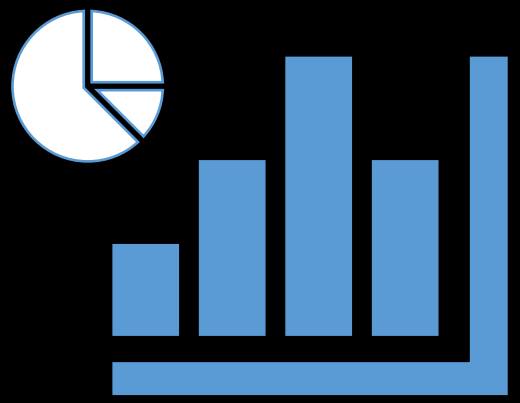
El Análisis Exploratorio de Datos (EDA) consiste en explorar y entender los datos antes de construir cualquier modelo. En Python se realiza principalmente con Pandas para resumir y limpiar la información, y con Matplotlib o Seaborn para crear gráficos que revelan patrones, valores atípicos y relaciones entre variables. Es una etapa esencial porque ayuda a tomar mejores decisiones en todo el proyecto.

¿Por qué pandas para EDA?

- Pandas es la mejor opción para hacer Análisis Exploratorio de Datos (EDA) en Python porque permite cargar, limpiar, resumir y explorar información con muy pocas líneas de código. Es rápido, flexible y se integra perfectamente con Matplotlib y Seaborn para crear gráficos de forma sencilla.
- Por eso, la mayoría de analistas y científicos de datos lo usan como primera herramienta en cualquier proyecto.



numero_cliente	nombre_cliente	tipo_cliente	numero_factura	monto_factura	fecha_facturacion
70684	Materiales Castillo S.A. de C.V.	otros	86475	1427.86	15/11/2019
70684	Materiales Castillo S.A. de C.V.	otros	39414	1132.58	18/07/2017
70684	Materiales Castillo S.A. de C.V.	otros	51627	107.6	15/11/2019
70684	Materiales Castillo S.A. de C.V.	otros	48506	933.11	31/12/2018
70684	Materiales Castillo S.A. de C.V.	otros	77711	1333.46	24/10/2017
70684	Materiales Castillo S.A. de C.V.	otros	35792	7.93	27/11/2018
70684	Materiales Castillo S.A. de C.V.	otros	61863	1040.02	26/08/2017
70684	Materiales Castillo S.A. de C.V.	otros	75300	117.54	13/09/2017
70684	Materiales Castillo S.A. de C.V.	otros	85800	1142.9	18/11/2019
70684	Materiales Castillo S.A. de C.V.	otros	79364	291.88	28/10/2019
70684	Materiales Castillo S.A. de C.V.	otros	61467	1303.37	26/11/2018

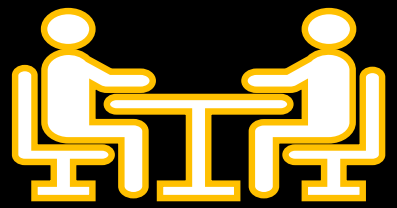


Los 5 pasos clave del EDA

- Carga y preparación 
- Exploración inicial 
- Limpieza de datos
- Análisis y visualizaciones 
- Insights y recomendaciones

problema

- En el sector de materiales eléctricos y luminarias en México, muchas empresas facturan miles de pedidos al año, pero no saben realmente dónde están ganando dinero de verdad ni por qué están perdiendo clientes.
- Se invierte en marketing, stock y visitas sin tener claridad: ¿en qué estados estamos creciendo? ¿Qué tipo de clientes se van primero? ¿Cuántos ya están inactivos y quiénes son los que más nos convendría recuperar?

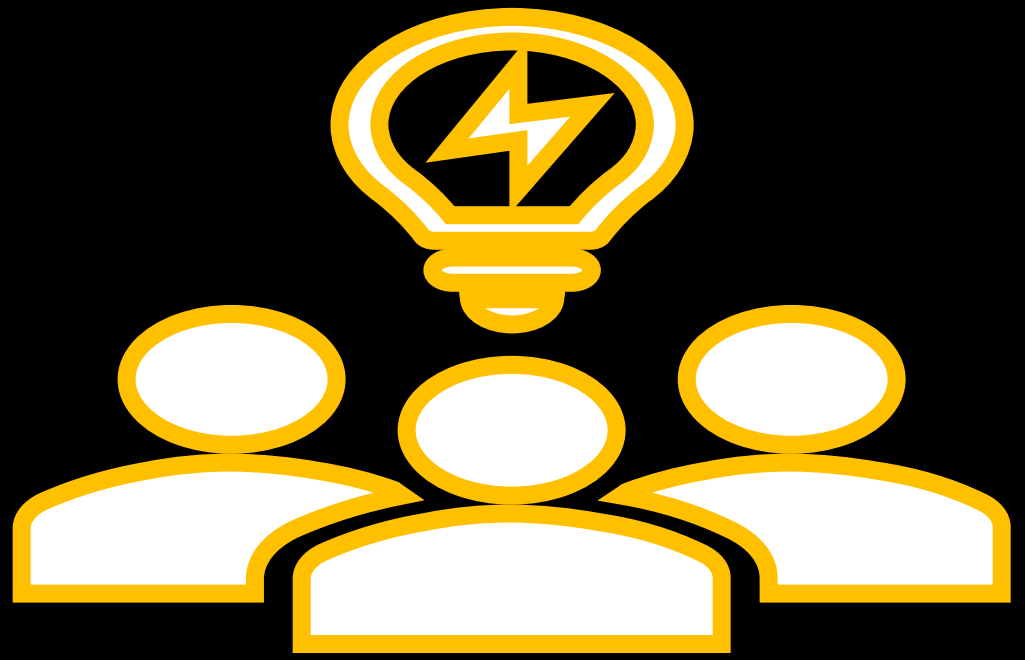


Objetivo del EDA

Este análisis transforma una base cruda de facturas en respuestas claras y accionables.

Con Pandas y unas cuantas gráficas vamos a descubrir exactamente dónde ganamos, quién se está yendo y a quién debemos llamar primero para recuperar ventas.

¿Y cómo lo hacemos?



Análisis exploratorio de datos con pandas

```
1
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import seaborn as sns
6
7
8  # Configuración visual
9  sns.set_style("whitegrid")
10 plt.rcParams['figure.figsize'] = (12, 6)
11 plt.rcParams['font.size'] = 12
12
13 print("Librerías cargadas correctamente. Listo para empezar el EDA. :) ")
```

✓ [16] 60ms

```
Librerías cargadas correctamente. Listo para empezar el EDA. :)
```

```python
# Usamos encoding utf-8.
df = pd.read_csv('data_python.csv', encoding='utf-8')

print("Dataset cargado correctamente.")
print(f"Dimensiones del DataFrame: {df.shape[0]} filas y {df.shape[1]} columnas")
print("\nPrimeras 5 filas para visualizar datos:")
display(df.head())
```

✓ [46] 48ms

```
Dataset cargado correctamente.
Dimensiones del DataFrame: 3042 filas y 9 columnas

Primeras 5 filas para visualizar datos:
```

5 rows × 9 cols — Static Output

	numero_cli…	nombre_cliente	tipo_cl…	numero_factura	monto_factura	fecha	municipio_cli…	state_cliente	rfc
0	30682	Isabel Sánchez Martínez	reventa	30286.0	0.00	25/04/2017	Naucalpan de Juá…	Estado de México	HO6F
1	30682	Isabel Sánchez Martínez	reventa	53633.0	153616.89	08/01/2018	Naucalpan de Juá…	Estado de México	HO6F
2	30682	Isabel Sánchez Martínez	reventa	37842.0	0.00	02/05/2018	Naucalpan de Juá…	Estado de México	HO6F
3	30682	Isabel Sánchez Martínez	reventa	94294.0	149923.46	22/07/2019	Naucalpan de Juá…	Estado de México	HO6F
4	30682	Isabel Sánchez Martínez	reventa	90057.0	10542.65	13/11/2019	Naucalpan de Juá…	Estado de México	HO6F

```python
# Esta celda es solo para explorar: tipos de datos, valores nulos, estadisticas básicas.
print("=== Información general del DataFrame ===")
df.info()

print("\n=== Estadísticas descriptivas de monto_factura ===")
display(df.describe())

print("\n=== Valores únicos por columna (para ver categorías) ===")
for col in ['tipo_cliente', 'state_cliente']:
    print(f"{col}: {df[col].nunique()} valores únicos")
    print(df[col].value_counts().head(5))
```

✓ [51] 103ms

```
=== Información general del DataFrame ===
<class 'pandas.DataFrame'>
Index: 1274 entries, 1 to 3041
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   numero_cliente     1274 non-null   int64
 1   nombre_cliente     1273 non-null   str
 2   tipo_cliente       1273 non-null   str
 3   numero_factura     1273 non-null   float64
 4   monto_factura      1274 non-null   float64
 5   fecha              1274 non-null   datetime64[us]
 6   municipio_cliente  1274 non-null   str
 7   state_cliente      1274 non-null   str
 8   rfc                1273 non-null   str
dtypes: datetime64[us](1), float64(2), int64(1), str(5)
memory usage: 99.5 KB
```

```
=== Estadísticas descriptivas de monto_factura ===
```

8 rows × 4 cols

	numero_cliente	numero_factura	monto_factura	fecha
count	1274.000000	1273.000000	1.274000e+03	1274
mean	59461.135793	66511.915161	3.115826e+05	2018-12-31 23:02:21.287284
min	30682.000000	30061.000000	6.000000e+00	2017-03-02 00:00:00
25%	42972.750000	49031.000000	4.998703e+04	2018-03-06 12:00:00
50%	59368.000000	67215.000000	1.003292e+05	2019-02-11 00:00:00
75%	74873.000000	84603.000000	1.913124e+05	2020-01-02 00:00:00
max	89005.000000	99997.000000	2.393121e+06	2020-03-31 00:00:00
std	16989.835188	20296.503065	4.873614e+05	NaN

```
=== Valores únicos por columna (para ver categorías) ===
tipo_cliente: 7 valores únicos
tipo_cliente
reventa        208
comercial      202
otros          200
constructor    174
industrial     173
Name: count, dtype: int64
state_cliente: 13 valores únicos
state_cliente
Jalisco             391
```

```python
# Eliminamos filas donde monto_factura esté en cero o vacío.
# También convertimos la fecha al formato correcto (dd/mm/yyyy) y eliminamos fechas inválidas.
# Reemplazamos valores raros en state_cliente (como "estado_cliente") por "Desconocido" para no perder filas útiles.
# Finalmente eliminamos duplicados por si existen facturas repetidas.

# Paso 1: Eliminar filas con monto_factura <= 0 o nulo
df = df.dropna(subset=['monto_factura'])
df = df[df['monto_factura'] > 0]

# Paso 2: Convertir fecha a datetime
df['fecha'] = pd.to_datetime(df['fecha'], format='%d/%m/%Y', errors='coerce')
df = df.dropna(subset=['fecha'])  # eliminamos filas con fecha inválida

# Paso 3: Limpiar state_cliente
df['state_cliente'] = df['state_cliente'].replace('estado_cliente', np.nan)
df['state_cliente'] = df['state_cliente'].fillna('Desconocido')

# Paso 4: Eliminar duplicados exactos
df = df.drop_duplicates()

print(f"Dataset después de limpieza: {df.shape[0]} filas")
print("Limpieza completada: filas con monto 0 o vacío eliminadas, fechas convertidas.")
```

✓ [53] 31ms

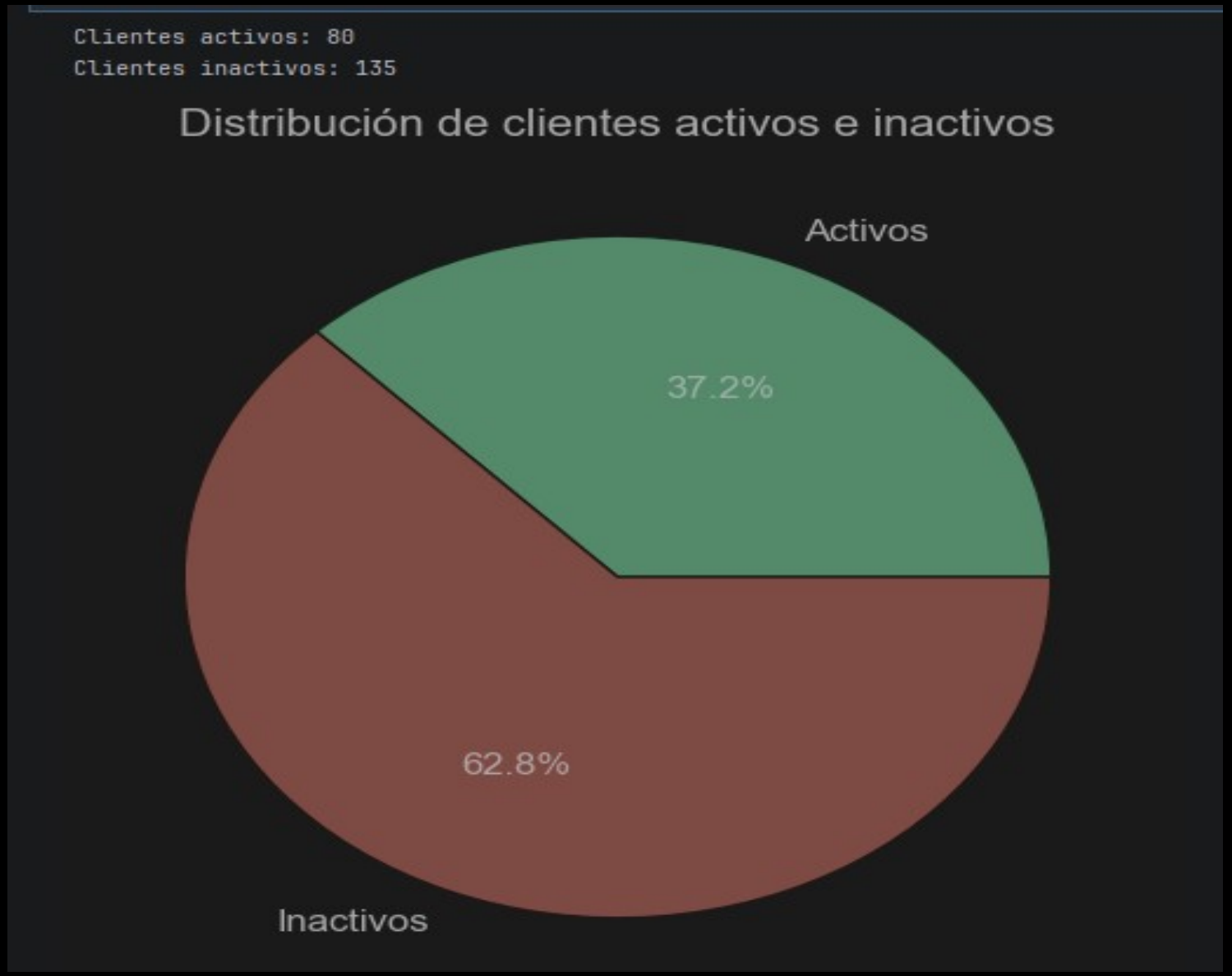
```
Dataset después de limpieza: 1274 filas
Limpieza completada: filas con monto 0 o vacío eliminadas, fechas convertidas.
```

```python
# Definimos fecha de referencia como 1 de abril 2020 (los datos terminan en marzo 2020).
# Calculamos la última compra por cliente y los días sin comprar.
# Si >105 días el cliente se considera inactivo.

reference_date = pd.to_datetime('2020-04-01')

last_purchase = (df.groupby('numero_cliente')['fecha']
                 .max()
                 .reset_index()
                 .rename(columns={'fecha': 'ultima_compra'}))

last_purchase['dias_sin_compra'] = (reference_date - last_purchase['ultima_compra']).dt.days
last_purchase['estado'] = np.where(last_purchase['dias_sin_compra'] <= 105, 'Activo', 'Inactivo')

activos = last_purchase['estado'].value_counts().get('Activo', 0)
inactivos = last_purchase['estado'].value_counts().get('Inactivo', 0)

print(f"Clientes activos: {activos}")
print(f"Clientes inactivos: {inactivos}")

# Gráfica de pastel
plt.figure()
plt.pie([activos, inactivos], labels=['Activos', 'Inactivos'], autopct='%1.1f%%', colors=['#2ecc71', '#e74c3c'])
plt.title('Distribución de clientes activos vs inactivos')
plt.show()
```

✓ [57] 80ms

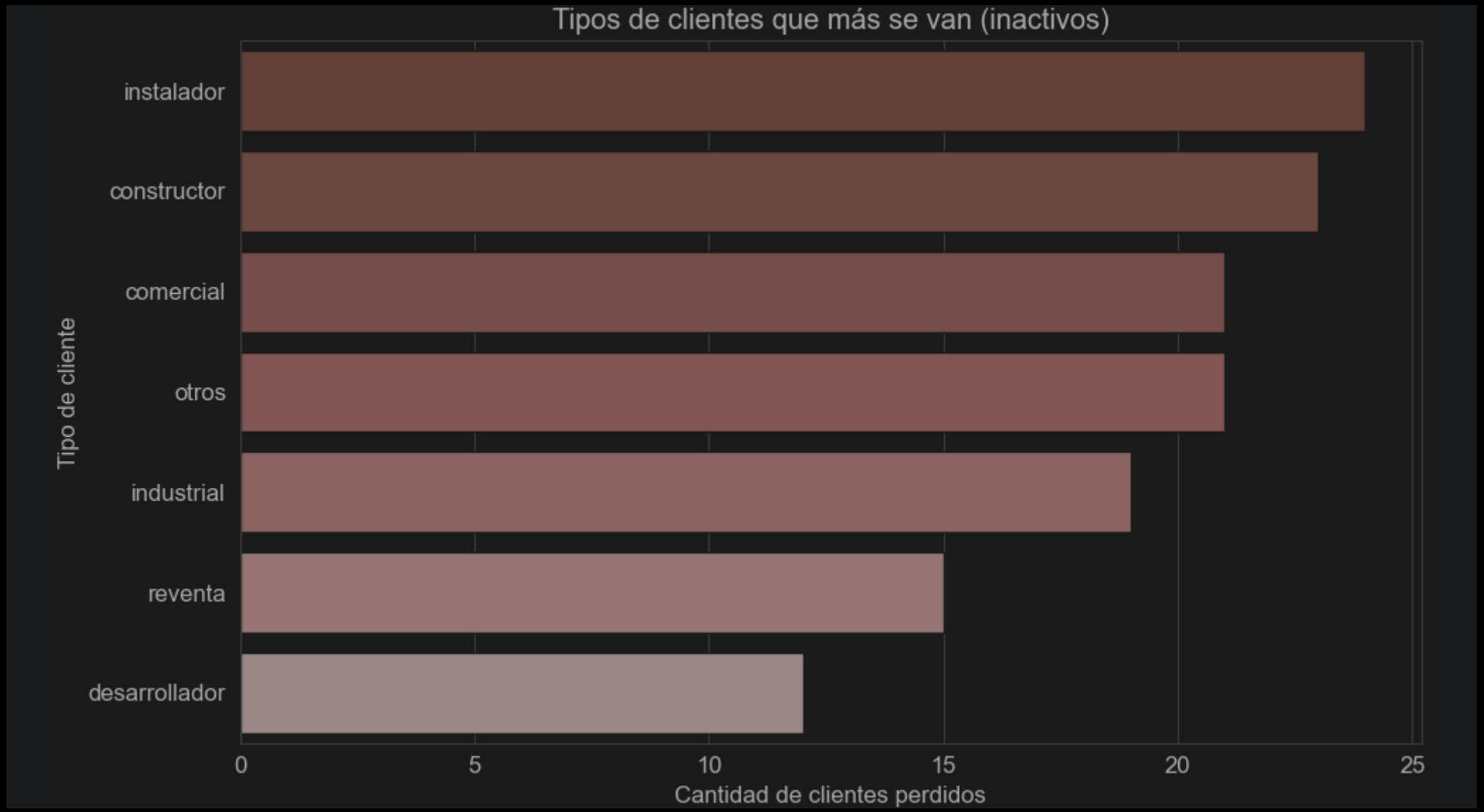
```
Clientes activos: 80
Clientes inactivos: 135
```

```
Clientes activos: 80
Clientes inactivos: 135
```



```python
# Unimos el estado del cliente con su tipo_cliente y contamos cuántos inactivos por tipo.
# Gráfica de barras para ver claramente cuál tipo pierde más clientes.

client_types = df[['numero_cliente', 'tipo_cliente']].drop_duplicates()
lost_clients = last_purchase[last_purchase['estado'] == 'Inactivo']
lost_with_type = lost_clients.merge(client_types, on='numero_cliente')

lost_by_type = (lost_with_type.groupby('tipo_cliente')
                .size()
                .reset_index(name='cantidad_perdidos')
                .sort_values('cantidad_perdidos', ascending=False))

plt.figure(figsize=(10, 6))                    # ← más legible en presentaciones
sns.barplot(
    data=lost_by_type,
    x='cantidad_perdidos',
    y='tipo_cliente',
    hue='tipo_cliente',        # ← SOLUCIÓN
    palette='Reds_d',
    legend=False               # ← SOLUCIÓN
)
plt.title('Tipos de clientes que más se van (inactivos)')
plt.xlabel('Cantidad de clientes perdidos')
plt.ylabel('Tipo de cliente')
plt.tight_layout()
plt.show()

display(lost_by_type)
```

✓ [28] 178ms


Tipos de clientes que más se van (inactivos)
instalador
constructor
comercial
otros
industrial
reventa
desarrollador
Tipo de cliente
0
5
10
15
20
25
Cantidad de clientes perdidos

7 rows 7 rows × 2 cols

	tipo_cliente	cantidad_perdidos
4	instalador	24
1	constructor	23
0	comercial	21
5	otros	21
3	industrial	19
6	reventa	15
2	desarrollador	12

```python
# Para priorizar, tomamos los clientes inactivos y calculamos cuánto gastaron en total.
# Los 7 que más gastaron son los que más nos interesa recuperar.
total_spent = (df.groupby('numero_cliente')['monto_factura']
               .sum()
               .reset_index()
               .rename(columns={'monto_factura': 'gasto_total'}))

priority_clients = (lost_clients.merge(total_spent, on='numero_cliente')
                    .sort_values('gasto_total', ascending=False)
                    .head(7))




print("Estos son los 7 clientes prioridad (incluye nombre y estado para contactarlos):")
priority_full = priority_clients.merge(
    df[['numero_cliente', 'nombre_cliente', 'state_cliente']].drop_duplicates(),
    on='numero_cliente'
)
display(priority_full[['numero_cliente', 'nombre_cliente', 'state_cliente', 'gasto_total']])
```

✓ [69] 47ms

Estos son los 7 clientes prioridad (incluye nombre y estado para contactarlos):

7 rows 7 rows × 4 cols

	numero_cliente	nombre_cliente	state_cliente	gasto_total
0	39870	Patricia González Sánchez	Michoacán	1060058.92
1	66483	Electrica Jiménez S.A. de C.V.	Jalisco	1016198.31
2	55550	Juan Martínez Pérez	Puebla	973912.03
3	64380	Juan Ramírez Gutiérrez	Aguascalientes	802326.09
4	84748	Construcciones Hernández S.A. de C.V.	Estado de México	790364.77
5	75650	Juan Cruz Rojas	Sinaloa	770061.28
6	33988	Carlos Rodríguez García	Sinaloa	703618.14

Aspecto	Ventaja	Desventaja
Facilidad de uso	Pocas líneas de código	Requiere conocimiento de Python
Velocidad	Muy rápido en datasets medianos	Puede ser lento en bases gigantes
Visualizaciones	Integración excelente con Seaborn	No es tan visual como herramientas BI
Reproducibilidad	Código guardado y compartible	Depende de la calidad inicial de datos

¿preguntas?

¡Gracias!

@ALMAAHUMADAMARTINEZ

Instagram

Facebook

Discord