

## Informe Notebook Proyecto: Cervecería casera en el mercado chileno



### Contexto del caso:

En el rubro de la creación de cervezas cada vez es más conocido el tema de las cervezas casera, estas durante estos últimos años han aumentado en popularidad debido a lo innovador de sus sabores, por eso cerveceras grandes de renombre como Kross quieren tener una parte de ese negocio haciendo ellos sus propios sabores innovadores, para ello la empresa Kross nos pidió a nosotros como grupo DJJ que analicemos un csv con muestras de cervezas caseras estadounidenses para que veamos cuáles fueron las mejores valoradas para integrarlas en el mercado chileno.

## Fase 1: Comprender Negocio

Nuestro objetivo como grupo es analizar los datos de las cervezas estadounidenses para poder ver cuáles fueron las mejores calificadas, para esto ocuparemos diferentes métodos y gráficos para sacar en claro todo los datos necesarios.

además de responder las siguientes preguntas

¿Cuáles son los tipos de cerveza mejor evaluados?

¿Cuáles son las cervezas con mejor sabor valorado?

¿Cuál de las cervezas tiene más reviews dentro de los datos?

## Importar librerías y el CSV

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

!pip install openpyxl
import pandas as pd
#
# Leer el archivo csv
#
file_path =
'C:/Users/massr/Desktop/reviewsCerveceriaKedro/cerveceriar/data/01_raw
/beer_reviews.csv'
df = pd.read_csv(file_path, sep=",")

Requirement already satisfied: openpyxl in c:\users\massr\desktop\
reviewscerveceriakedro\venv\lib\site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in c:\users\massr\desktop\
reviewscerveceriakedro\venv\lib\site-packages (from openpyxl) (2.0.0)

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

## Fase 2: Comprender los datos del CSV

Ahora que tenemos el csv cargado veremos la cantidad de datos y columnas además de analizar los datos que más se nos hagan útiles para el proyecto

```
df
```

```
{"type": "dataframe", "variable_name": "df"}
```

ver número de filas y columnas del csv

```
df.shape
```

```
(1586614, 13)
```

ver columnas del csv

```
df.columns
```

```
Index(['brewery_id', 'brewery_name', 'review_time', 'review_overall',
       'review_aroma', 'review_appearance', 'review_profilename',
       'beer_style',
       'review_palate', 'review_taste', 'beer_name', 'beer_abv',
       'beer_beerid'],
      dtype='object')
```

Resumen de los datos del csv

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1586614 entries, 0 to 1586613
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   brewery_id      1586614 non-null   int64  
 1   brewery_name    1586599 non-null   object  
 2   review_time     1586614 non-null   int64  
 3   review_overall  1586614 non-null   float64 
 4   review_aroma    1586614 non-null   float64 
 5   review_appearance  1586614 non-null   float64 
 6   review_profilename  1586266 non-null   object  
 7   beer_style      1586614 non-null   object  
 8   review_palate   1586614 non-null   float64 
 9   review_taste    1586614 non-null   float64 
 10  beer_name       1586614 non-null   object  
 11  beer_abv        1518829 non-null   float64 
 12  beer_beerid    1586614 non-null   int64  
dtypes: float64(6), int64(3), object(4)
memory usage: 157.4+ MB
```

Resumen estadistico de las columnas

```
df.describe()
```

```
{"summary": {"\n    \"name\": \"df\", \n    \"rows\": 8, \n    \"fields\": [\n        {\n            \"column\": \"brewery_id\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 559027.2865717531, \n                \"min\": 1.0, \n                \"max\": 1586614.0, \n                \"num_unique_values\": 8, \n                \"samples\": [\n                    3130.0992018222455, \n                    429.0, \n                    1586614.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"review_time\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 549815951.7857368, \n                \"min\": 1586614.0, \n                \"max\": 1326285348.0, \n                \"num_unique_values\": 8, \n                \"samples\": [\n                    1224089280.0122108, \n                    1239202881.5, \n                    1586614.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"review_overall\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 560951.6715230996, \n                \"min\": 0.0, \n                \"max\": 1586614.0, \n                \"num_unique_values\": 8, \n                \"samples\": [\n                    3.8155808533140387, \n                    4.0, \n                    1586614.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"review_aroma\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 560951.6514682331, \n                \"min\": 0.69761672878963, \n                \"max\": 1586614.0, \n                \"num_unique_values\": 7, \n                \"samples\": [\n                    1586614.0, \n                    3.735636077836197, \n                    4.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"review_appearance\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 560951.7007400723, \n                \"min\": 0.0, \n                \"max\": 1586614.0, \n                \"num_unique_values\": 7, \n                \"samples\": [\n                    1586614.0, \n                    3.8416416973504584, \n                    4.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"review_palate\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 560951.6518386202, \n                \"min\": 0.6822183633739491, \n                \"max\": 1586614.0, \n                \"num_unique_values\": 7, \n                \"samples\": [\n                    1586614.0, \n                    3.7437013665579655, \n                    4.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"review_taste\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 560951.6215891932, \n                \"min\": 0.7319696098919117, \n                \"max\": 1586614.0, \n                \"num_unique_values\": 8, \n                \"samples\": [\n                    3.792860456292457, \n                    4.0, \n                    1586614.0\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            } \n        }, \n        {\n            \"column\": \"beer_abv\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 536982.7349568895, \n                \"min\": 0.01, \n                \"max\": 1518829.0, \n                \"num_unique_values\": 8, \n                \"samples\": [\n                    7.042386753215804, \n                    6.5,\n                    1586614.0\n                ]\n            }\n        }\n    ]\n}
```

```
n      1518829.0\n      ],\n      \"semantic_type\": \"\",\n      \"description\": \"\n      },\n      {\n      \"column\":\n      \"beer_beerid\",\n      \"properties\": {\n      \"dtype\":\n      \"number\",\n      \"std\": 552618.143701588,\n      \"min\":\n      3.0,\n      \"max\": 1586614.0,\n      \"num_unique_values\": 8,\n      \"samples\": [\n      21712.79427888573,\n      13906.0,\n      1586614.0\n      ],\n      \"semantic_type\": \"\",\n      \"description\": \"\"\n      }\n    }\n  ]\n},\n\"type\":\"dataframe\"}
```

#Ver los datos que nos resulten más interesantes

Nombre de la cerveceria

```
df.brewery_name\n\n0          Vecchio Birraio\n1          Vecchio Birraio\n2          Vecchio Birraio\n3          Vecchio Birraio\n4  Caldera Brewing Company\n...\n1586609  The Defiant Brewing Company\n1586610  The Defiant Brewing Company\n1586611  The Defiant Brewing Company\n1586612  The Defiant Brewing Company\n1586613  The Defiant Brewing Company\nName: brewery_name, Length: 1586614, dtype: object
```

nombre de las cervezas

```
df.beer_name\n\n0          Sausa Weizen\n1          Red Moon\n2  Black Horse Black Beer\n3          Sausa Pils\n4          Cauldron DIPA\n...\n1586609  The Horseman's Ale\n1586610  The Horseman's Ale\n1586611  The Horseman's Ale\n1586612  The Horseman's Ale\n1586613  The Horseman's Ale\nName: beer_name, Length: 1586614, dtype: object
```

regusto en el paladar

```
df.review_palate
```

```
0      1.5
1      3.0
2      3.0
3      2.5
4      4.0
      ...
1586609   4.0
1586610   2.0
1586611   3.5
1586612   4.5
1586613   4.5
Name: review_palate, Length: 1586614, dtype: float64
```

reseña general de la cerveza

```
df.review_overall
0      1.5
1      3.0
2      3.0
3      3.0
4      4.0
      ...
1586609   5.0
1586610   4.0
1586611   4.5
1586612   4.0
1586613   5.0
Name: review_overall, Length: 1586614, dtype: float64
```

gusto de la cerveza

```
df.review_taste
0      1.5
1      3.0
2      3.0
3      3.0
4      4.5
      ...
1586609   4.0
1586610   4.0
1586611   4.0
1586612   4.5
1586613   4.5
Name: review_taste, Length: 1586614, dtype: float64
```

apariencia de la cerveza

```
df.review_appearance

0      2.5
1      3.0
2      3.0
3      3.5
4      4.0
      ...
1586609   3.5
1586610   2.5
1586611   3.0
1586612   4.5
1586613   4.5
Name: review_appearance, Length: 1586614, dtype: float64
```

grados de alcohol en la cerveza

```
df.beer_abv

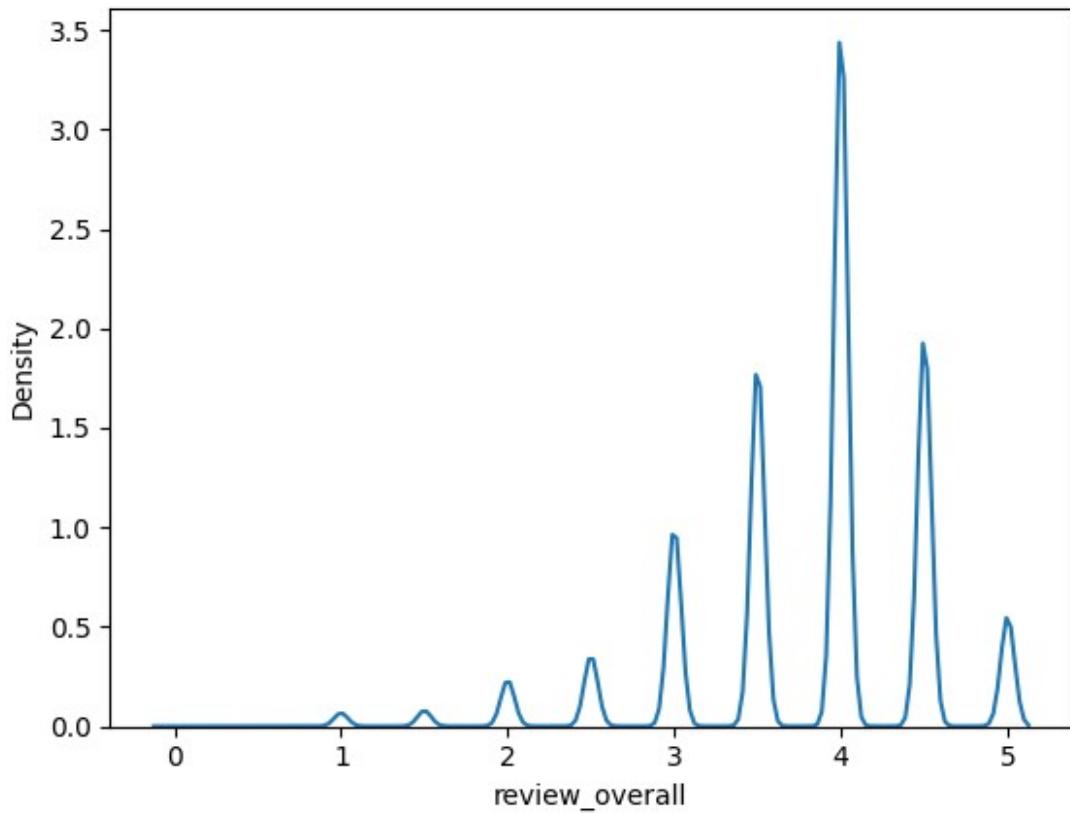
0      5.0
1      6.2
2      6.5
3      5.0
4      7.7
      ...
1586609   5.2
1586610   5.2
1586611   5.2
1586612   5.2
1586613   5.2
Name: beer_abv, Length: 1586614, dtype: float64
```

#Empezaremos con la graficación y análisis de los datos

En esta parte del proyecto nos encargamos de ver bien los datos que consideramos más factibles a tomar en este proyecto, en relación a las preguntas que planteamos en la parte 1.

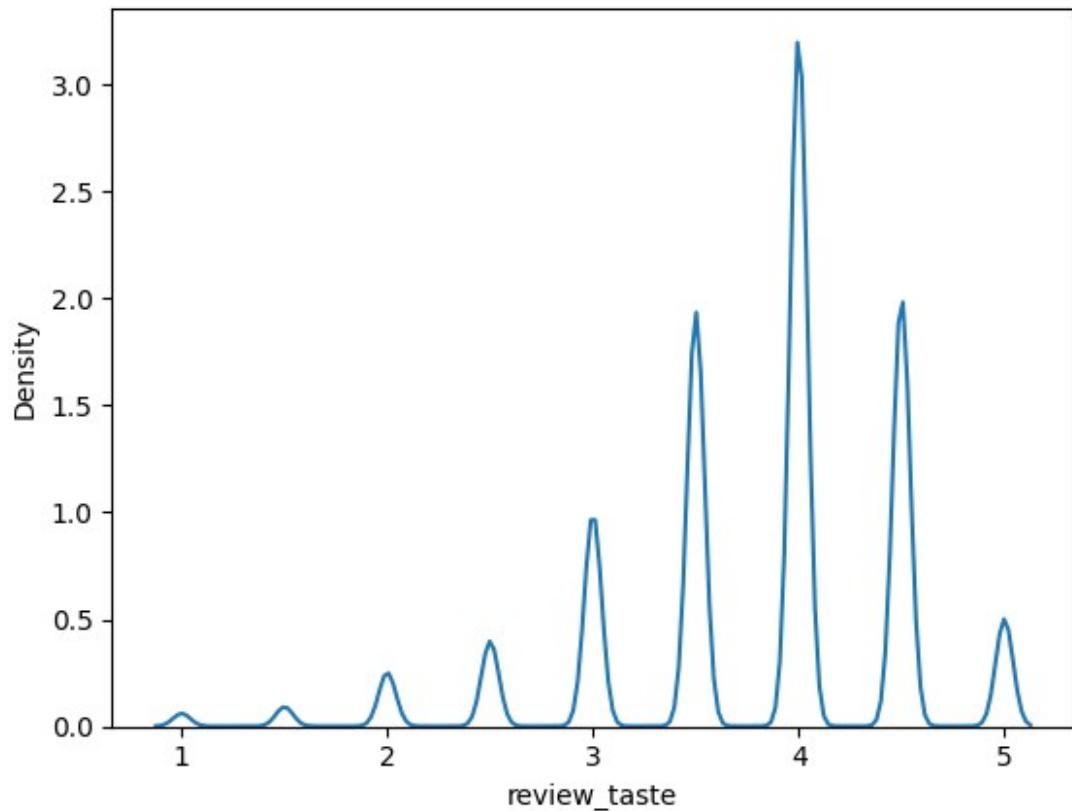
```
from seaborn import kdeplot
kdeplot(df.review_overall)

<Axes: xlabel='review_overall', ylabel='Density'>
```



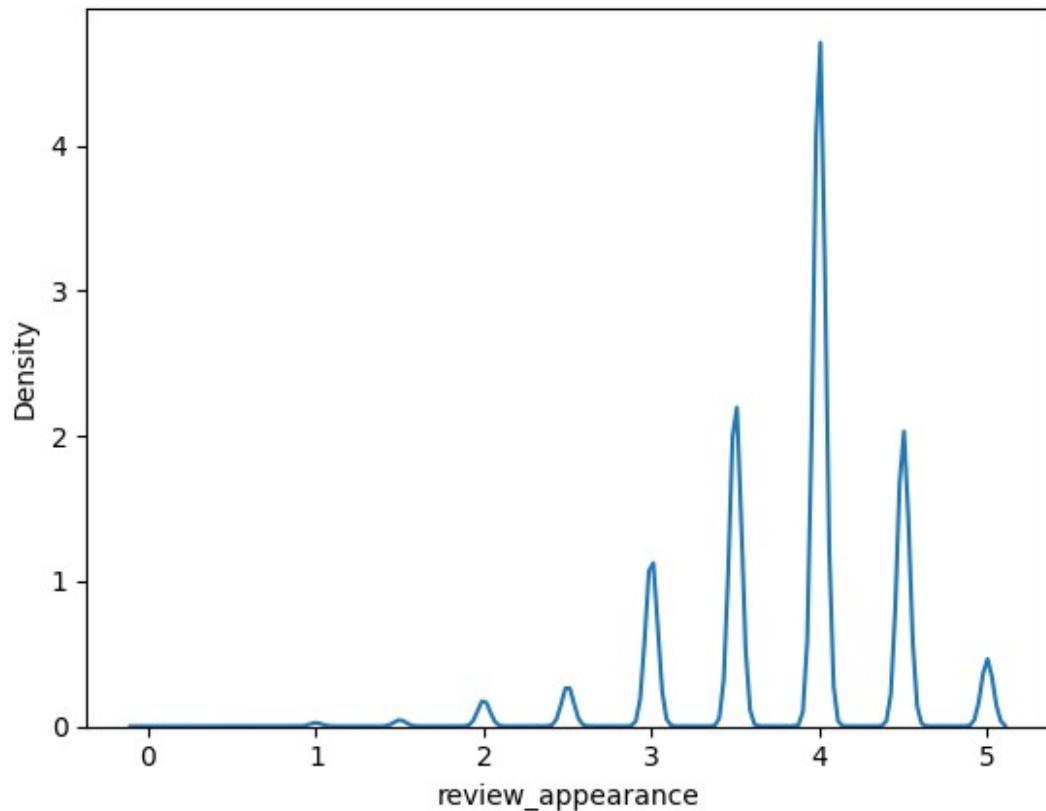
```
from seaborn import kdeplot
kdeplot(df.review_taste)

<Axes: xlabel='review_taste', ylabel='Density'>
```



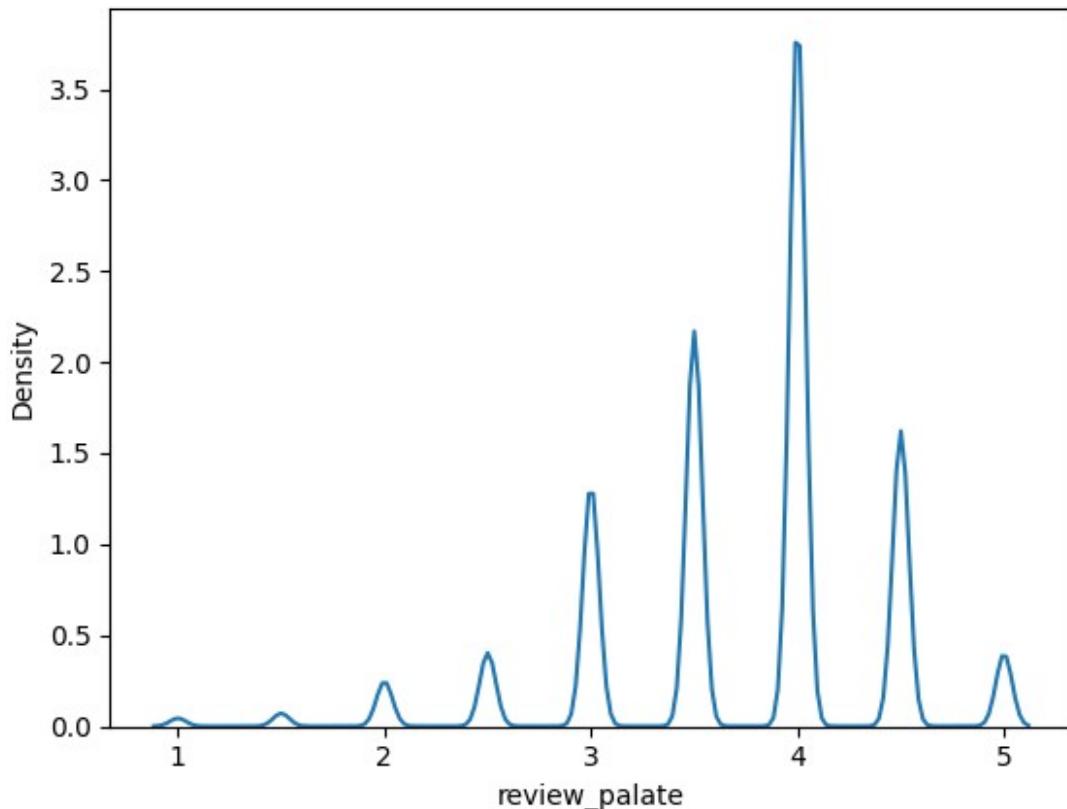
```
from seaborn import kdeplot
kdeplot(df.review_appearance)

<Axes: xlabel='review_appearance', ylabel='Density'>
```



```
from seaborn import kdeplot
kdeplot(df.review_palate)

<Axes: xlabel='review_palate', ylabel='Density'>
```



## Identificando datos

Después de cargar el CSV decidimos que los datos que vamos a tomar para analizar serán review\_overall que no ayudará a identificar a las cervezas que tienen una mejor reseña entre los catadores. El review\_taste sera utilizado para identificar cuál de las cervezas tuvieron una mejor valoración en cuanto al sabor, el review\_appearance y review\_palate serán utilizado para ver cuales de las cervezas tiene una mejor apariencia y sabor en el paladar ya que como grupo que este es un factor también a tomar en cuenta. Además de todo esto se tendrá en cuenta a los datos de beer\_abv para clasificar a las cervezas.

En los siguientes gráficos intentamos ver a más detalle la distribución de los anteriores datos que graficamos.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

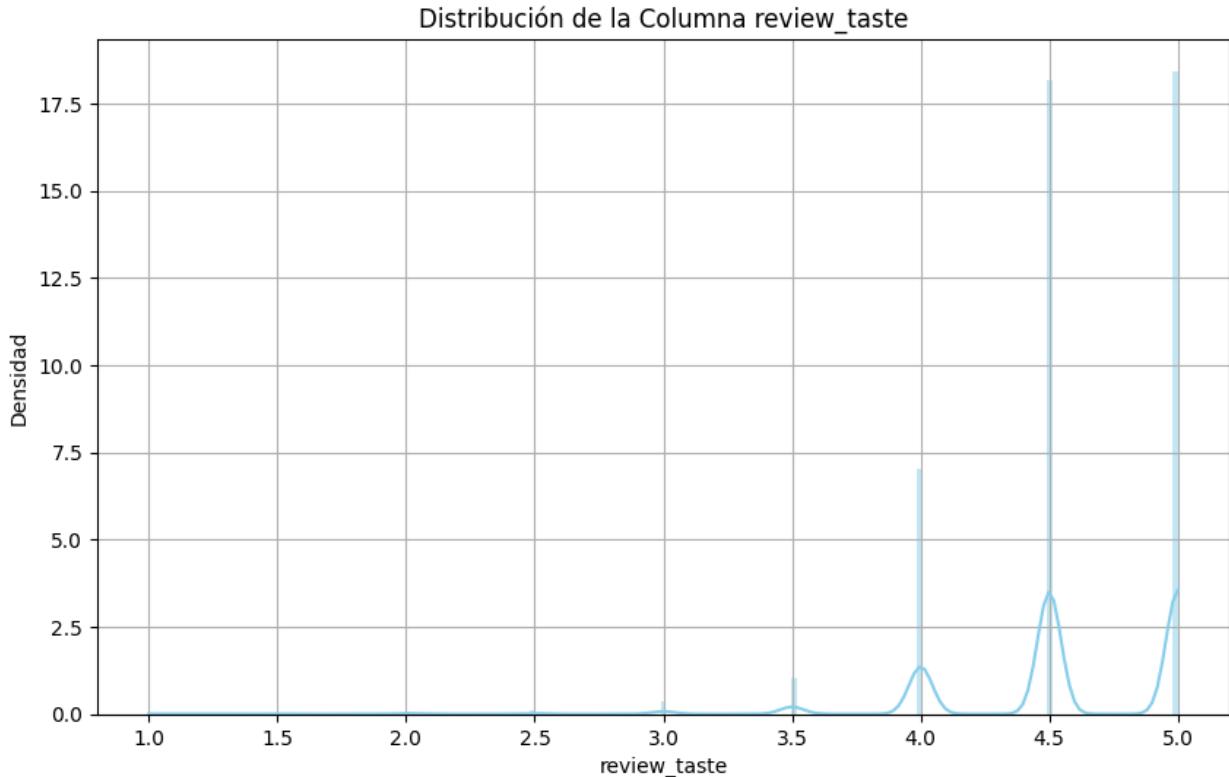
# Extraer la columna de interés
data = df['review_taste']

# Crear el histograma con una distribución normal superpuesta
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, stat="density", linewidth=0,
```

```

color='skyblue')
plt.title('Distribución de la Columna review_taste')
plt.xlabel('review_taste')
plt.ylabel('Densidad')
plt.grid(True)
plt.show()

```



```

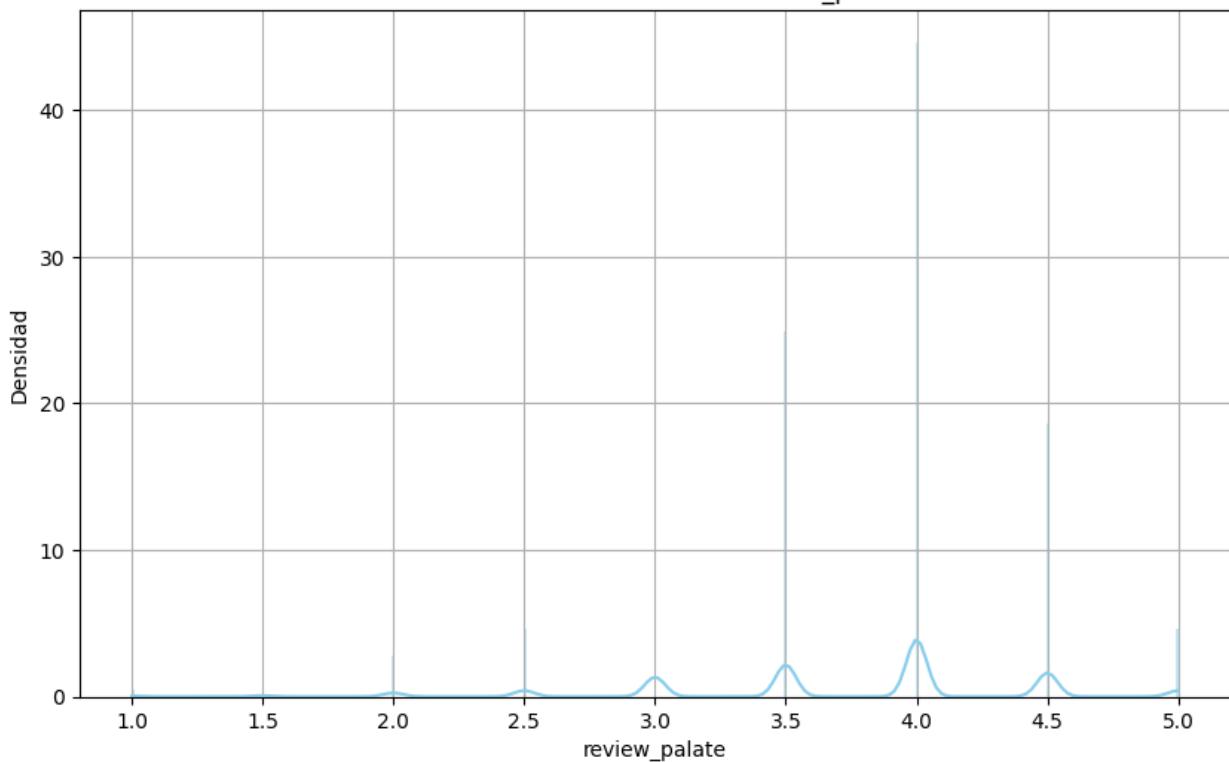
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Extraer la columna de interés
data = df['review_palate']

# Crear el histograma con una distribución normal superpuesta
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, stat="density", linewidth=0,
color='skyblue')
plt.title('Distribución de la Columna review_palate')
plt.xlabel('review_palate')
plt.ylabel('Densidad')
plt.grid(True)
plt.show()

```

Distribución de la Columna review\_palate

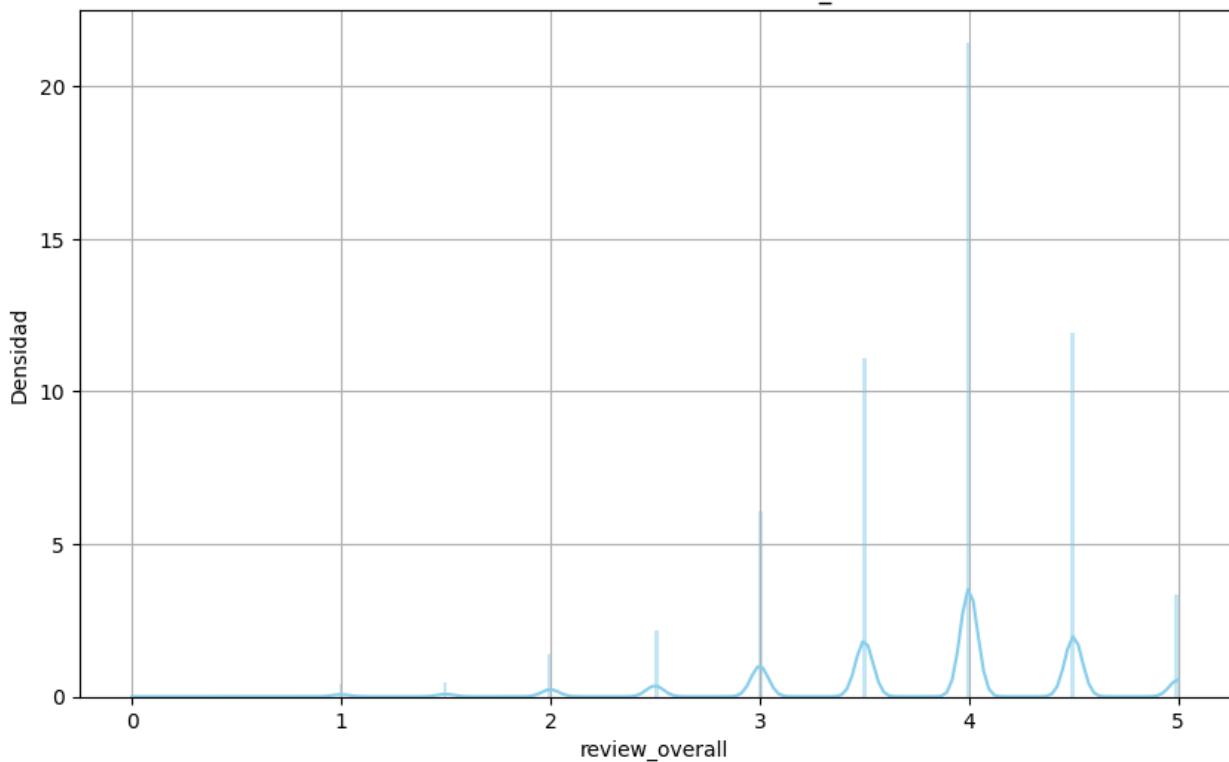


```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Extraer la columna de interés
data = df['review_overall']

# Crear el histograma con una distribución normal superpuesta
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, stat="density", linewidth=0,
color='skyblue')
plt.title('Distribución de la Columna review_overall')
plt.xlabel('review_overall')
plt.ylabel('Densidad')
plt.grid(True)
plt.show()
```

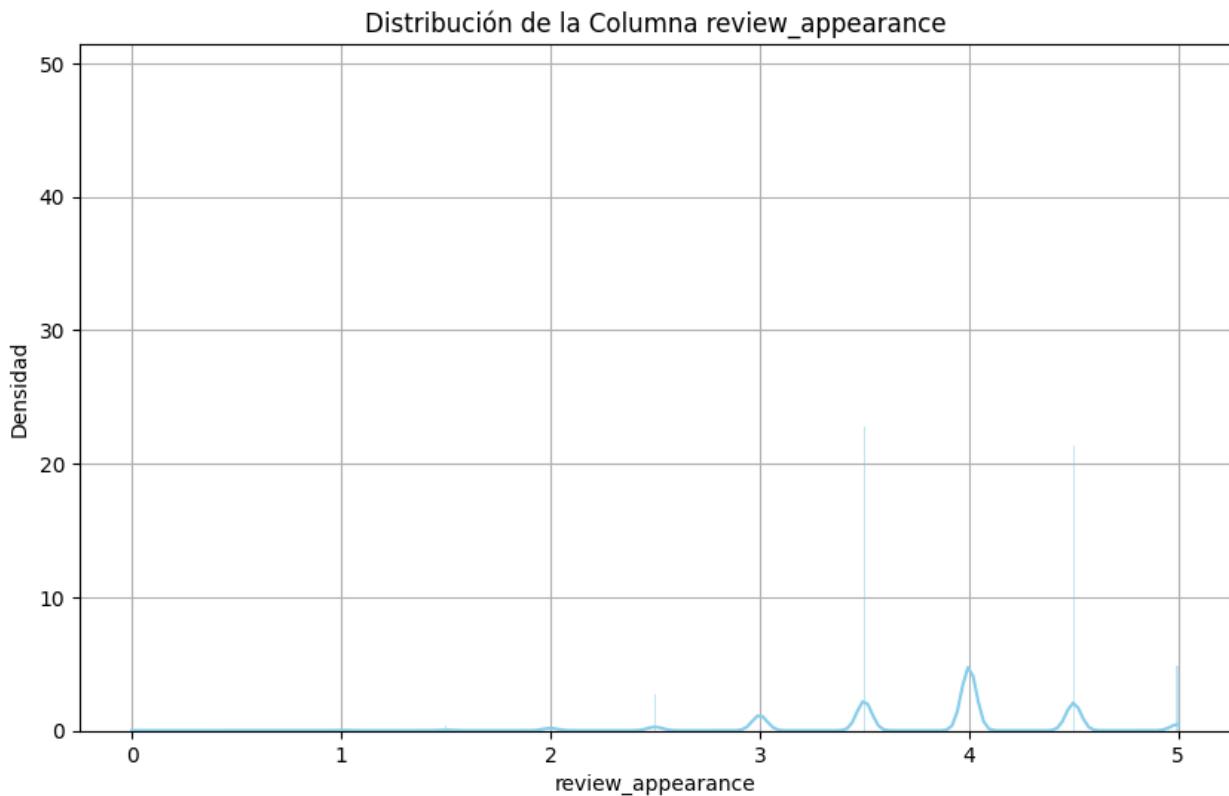
Distribución de la Columna review\_overall



```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Extraer la columna de interés
data = df['review_appearance']

# Crear el histograma con una distribución normal superpuesta
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, stat="density", linewidth=0,
color='skyblue')
plt.title('Distribución de la Columna review_appearance')
plt.xlabel('review_appearance')
plt.ylabel('Densidad')
plt.grid(True)
plt.show()
```



## Datos de tendencia central

en esta parte con la información que graficamos anteriormente no era suficiente así que queríamos tener a detalle de la media, moda y mediana de los datos ya analizados

```
#Codigo que analiza la media, mediana y moda de los datos
import pandas as pd
from scipy import stats

# codigo que ayuda a calcular la media de las rewievs
media = df['review_overall'].mean()
print(f'Media: {media}')

# Codigo que ayuda a calcular la mediana de las rewievs
mediana = df['review_overall'].median()
print(f'Mediana: {mediana}')

# Codigo que se usa para medir la moda de las rewievs
moda = df['review_overall'].mode()
print(f'Moda: {moda.values}')

# Estadísticas descriptivas adicionales
estadisticas = df['review_overall'].describe()
```

```
print('\nEstadísticas descriptivas: ')
print(estadisticas)

Media: 3.8781144211239
Mediana: 4.0
Moda: [4.]

Estadísticas descriptivas:
count    88620.000000
mean      3.878114
std       0.695637
min       0.000000
25%      3.500000
50%      4.000000
75%      4.500000
max       5.000000
Name: review_overall, dtype: float64

#Código que analiza la media, mediana y moda de los datos
import pandas as pd
from scipy import stats

# código que ayuda a calcular la media del sabor
media = df['review_taste'].mean()
print(f'Media: {media}')

# Código que ayuda a calcular la mediana del sabor
mediana = df['review_taste'].median()
print(f'Mediana: {mediana}')

# Código que se usa para medir la moda del sabor
moda = df['review_taste'].mode()
print(f'Moda: {moda.values}')

# Estadísticas descriptivas adicionales
estadisticas = df['review_taste'].describe()
print('\nEstadísticas descriptivas: ')
print(estadisticas)

Media: 3.8865380275332884
Mediana: 4.0
Moda: [4.]

Estadísticas descriptivas:
count    88620.000000
mean      3.886538
std       0.706509
min       1.000000
25%      3.500000
50%      4.000000
```

```
75%           4.500000
max           5.000000
Name: review_taste, dtype: float64

#Codigo que analiza la media, mediana y moda de los datos
import pandas as pd
from scipy import stats

# codigo que ayuda a calcular la media de la apariencia
media = df['review_appearance'].mean()
print(f'Media: {media}')

# Codigo que ayuda a calcular la mediana de la apariencia
mediana = df['review_appearance'].median()
print(f'Mediana: {mediana}')

# Codigo que se usa para medir la moda de la apariencia
moda = df['review_appearance'].mode()
print(f'Moda: {moda.values}')

# Estadísticas descriptivas adicionales
estadisticas = df['review_appearance'].describe()
print('\nEstadísticas descriptivas:')
print(estadisticas)

Media: 3.9009986459038593
Mediana: 4.0
Moda: [4.]

Estadísticas descriptivas:
count    88620.000000
mean      3.900999
std       0.592547
min       0.000000
25%      3.500000
50%      4.000000
75%      4.500000
max       5.000000
Name: review_appearance, dtype: float64

#Codigo que analiza la media, mediana y moda de los datos
import pandas as pd
from scipy import stats

# codigo que ayuda a calcular la media de la apariencia
media = df['review_palate'].mean()
print(f'Media: {media}')

# Codigo que ayuda a calcular la mediana de la apariencia
```

```

mediana = df['review_palate'].median()
print(f'Mediana: {mediana}')

# Código que se usa para medir la moda de la apariencia
moda = df['review_palate'].mode()
print(f'Moda: {moda.values}')

# Estadísticas descriptivas adicionales
estadisticas = df['review_palate'].describe()
print('\nEstadísticas descriptivas:')
print(estadisticas)

Media: 3.8274260889189797
Mediana: 4.0
Moda: [4.0]

Estadísticas descriptivas:
count    88620.000000
mean      3.827426
std       0.658091
min       1.000000
25%      3.500000
50%      4.000000
75%      4.000000
max      5.000000
Name: review_palate, dtype: float64

```

## Gráfico de dispersión

En esta parte del proyecto tomamos los datos que analizamos anteriormente y los graficamos de tal forma que podamos ver cuales son los datos que destacan del resto.

```

# Importar las bibliotecas necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

varianza_x = df['review_overall'].var()
desviacion_x = df['review_overall'].std()

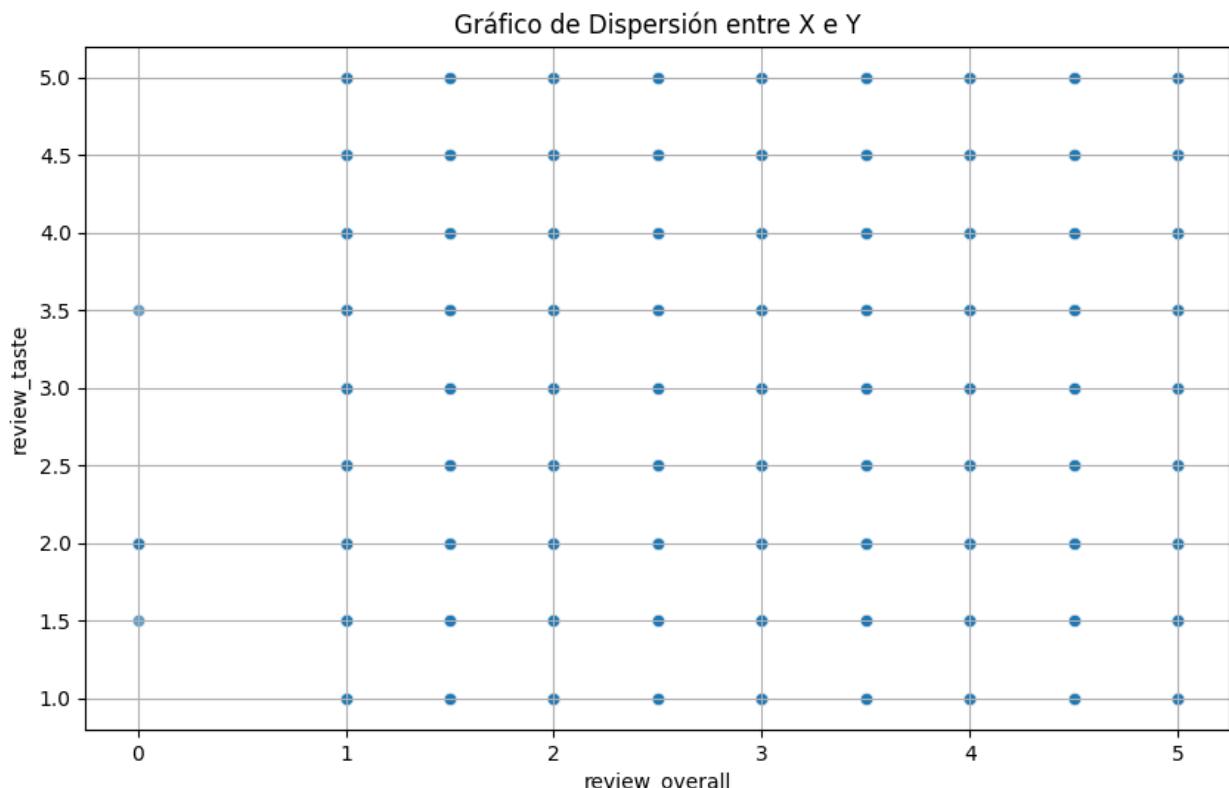
varianza_y = df['review_taste'].var()
desviacion_y = df['review_taste'].std()

print(f'Varianza de X: {varianza_x}')
print(f'Desviación estándar de X: {desviacion_x}')
print(f'Varianza de Y: {varianza_y}')
print(f'Desviación estándar de Y: {desviacion_y}')

```

```
# Crear un gráfico de dispersión
plt.figure(figsize=(10, 6))
sns.scatterplot(x='review_overall', y='review_taste', data=df,
alpha=0.7)
plt.title('Gráfico de Dispersión entre X e Y')
plt.xlabel('review_overall')
plt.ylabel('review_taste')
plt.grid(True)
plt.show()
```

Varianza de X: 0.519295876714356  
Desviación estándar de X: 0.7206218680517238  
Varianza de Y: 0.5357795098053175  
Desviación estándar de Y: 0.7319696098919117



```
# Importar las bibliotecas necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

varianza_x = df['review_time'].var()
desviacion_x = df['review_time'].std()
```

```

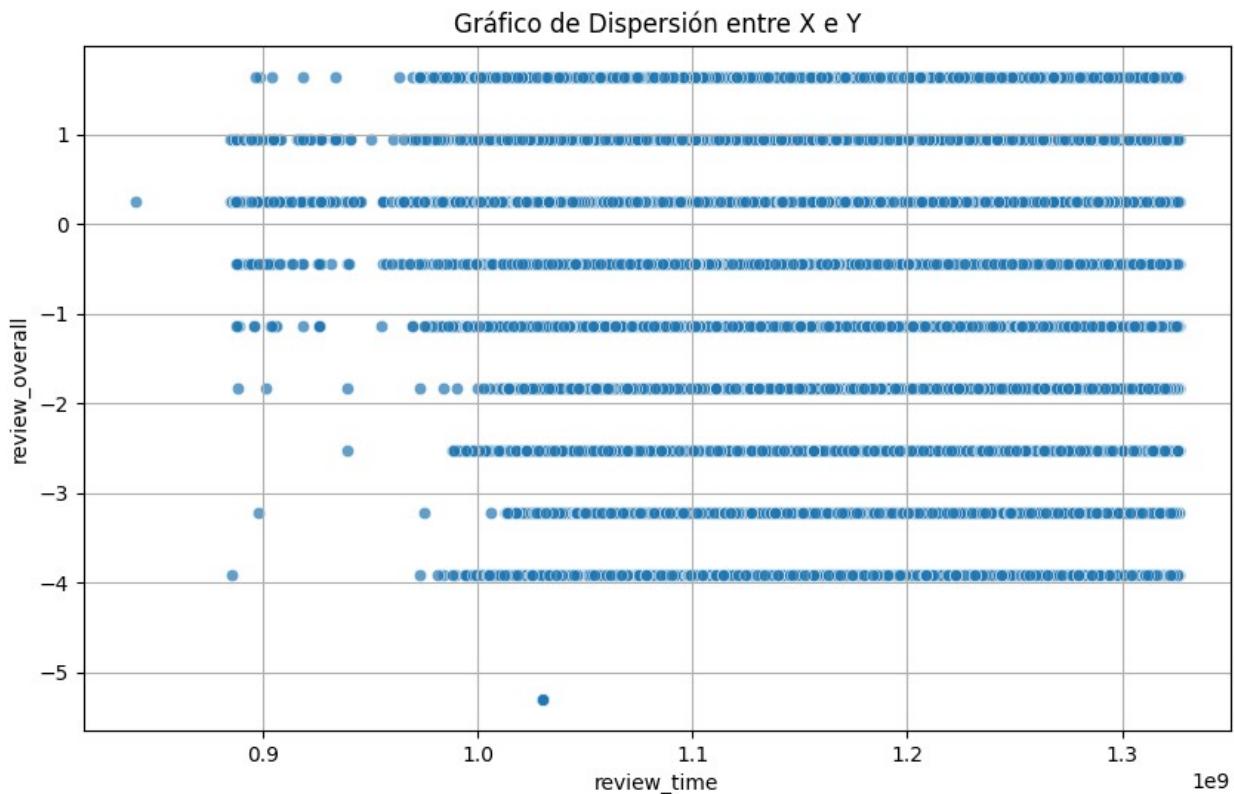
varianza_x = df['review_overall'].var()
desviacion_x = df['review_overall'].std()

print(f'Varianza de X: {varianza_x}')
print(f'Desviación estándar de X: {desviacion_x}')
print(f'Varianza de Y: {varianza_y}')
print(f'Desviación estándar de Y: {desviacion_y}')

# Crear un gráfico de dispersión
plt.figure(figsize=(10, 6))
sns.scatterplot(x='review_time', y='review_overall', data=df,
alpha=0.7)
plt.title('Gráfico de Dispersion entre X e Y')
plt.xlabel('review_time')
plt.ylabel('review_overall')
plt.grid(True)
plt.show()

```

Varianza de X: 5859025964738176.0  
 Desviación estándar de X: 76544274.53923759  
 Varianza de Y: 1.0000006302734188  
 Desviación estándar de Y: 1.0000003151366597



```

# Importar las bibliotecas necesarias
import pandas as pd

```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

varianza_x = df['review_time'].var()
desviacion_x = df['review_time'].std()

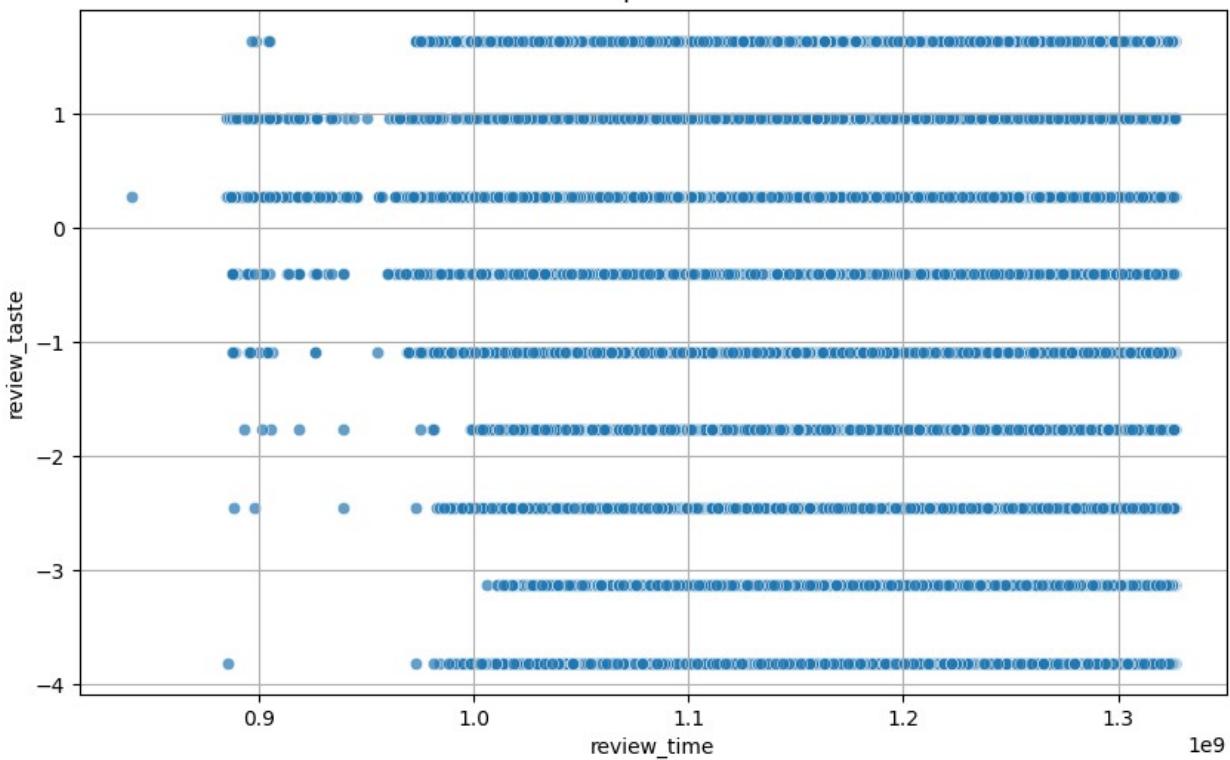
varianza_y = df['review_taste'].var()
desviacion_y = df['review_taste'].std()

print(f'Varianza de X: {varianza_x}')
print(f'Desviación estándar de X: {desviacion_x}')
print(f'Varianza de Y: {varianza_y}')
print(f'Desviación estándar de Y: {desviacion_y}')

# Crear un gráfico de dispersión
plt.figure(figsize=(10, 6))
sns.scatterplot(x='review_time', y='review_taste', data=df, alpha=0.7)
plt.title('Gráfico de Dispersión entre X e Y')
plt.xlabel('review_time')
plt.ylabel('review_taste')
plt.grid(True)
plt.show()

Varianza de X: 5859025964738176.0
Desviación estándar de X: 76544274.53923759
Varianza de Y: 1.000000630273419
Desviación estándar de Y: 1.0000003151366599
```

Gráfico de Dispersion entre X e Y



```
# Importar las bibliotecas necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

varianza_x = df['review_time'].var()
desviacion_x = df['review_time'].std()

varianza_y = df['review_appearance'].var()
desviacion_y = df['review_appearance'].std()

print(f'Varianza de X: {varianza_x}')
print(f'Desviación estándar de X: {desviacion_x}')
print(f'Varianza de Y: {varianza_y}')
print(f'Desviación estándar de Y: {desviacion_y}')

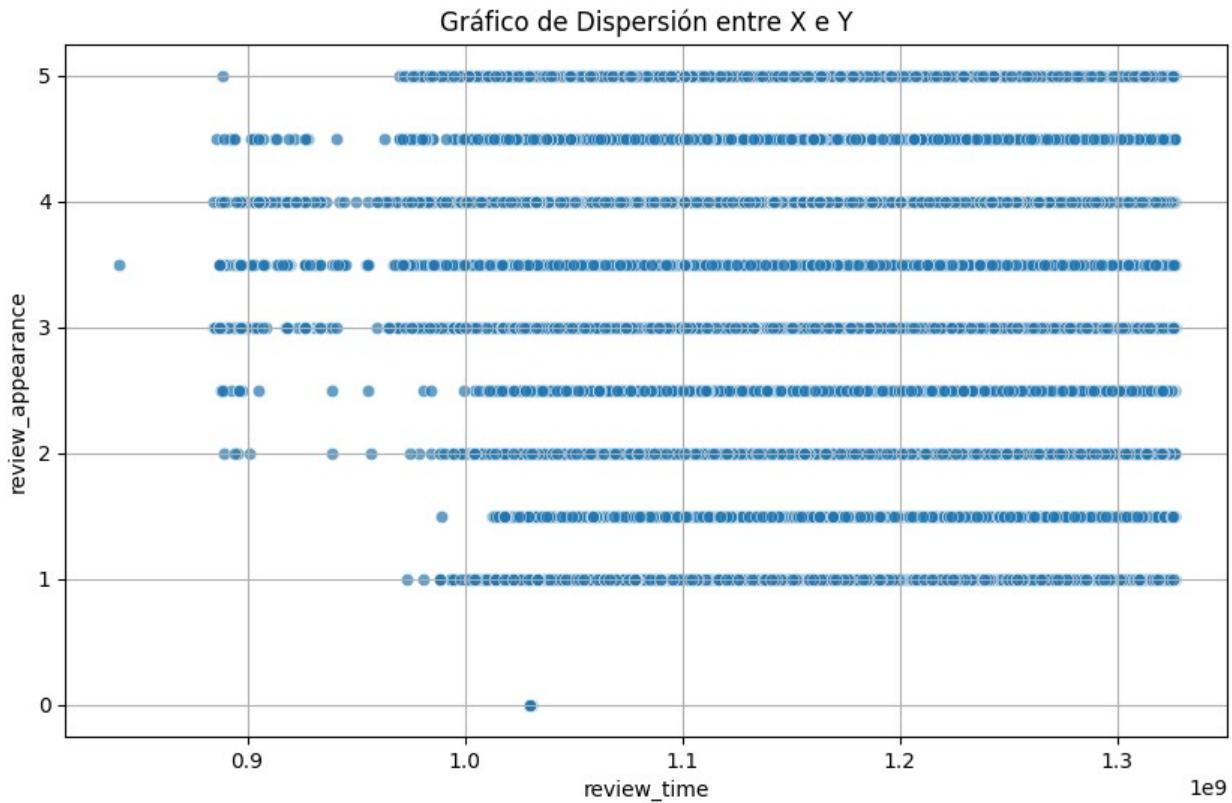
# Crear un gráfico de dispersión
plt.figure(figsize=(10, 6))
sns.scatterplot(x='review_time', y='review_appearance', data=df,
alpha=0.7)
plt.title('Gráfico de Dispersion entre X e Y')
plt.xlabel('review_time')
plt.ylabel('review_appearance')
```

```

plt.grid(True)
plt.show()

Varianza de X: 5859025964738176.0
Desviación estándar de X: 76544274.53923759
Varianza de Y: 0.3795702998810948
Desviación estándar de Y: 0.6160927688920678

```



```

# Importar las bibliotecas necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

varianza_x = df['review_time'].var()
desviacion_x = df['review_time'].std()

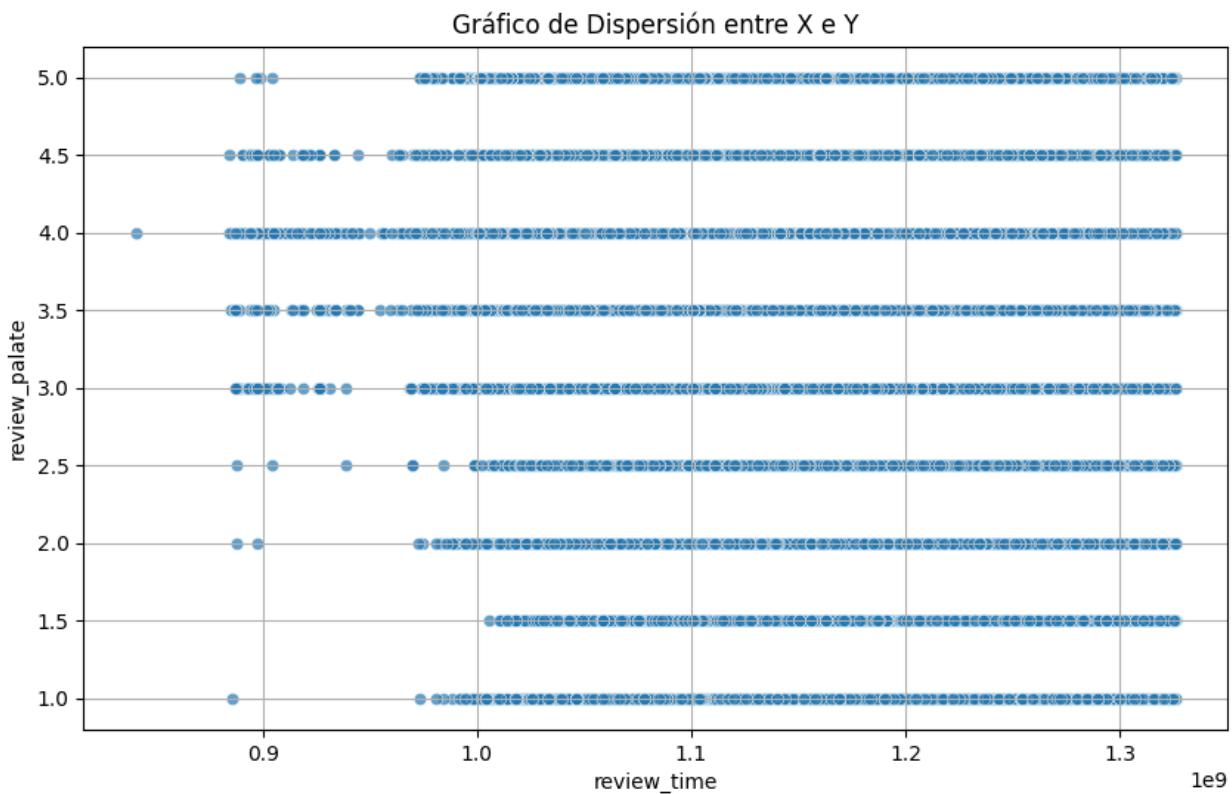
varianza_y = df['review_palate'].var()
desviacion_y = df['review_palate'].std()

print(f'Varianza de X: {varianza_x}')
print(f'Desviación estándar de X: {desviacion_x}')
print(f'Varianza de Y: {varianza_y}')
print(f'Desviación estándar de Y: {desviacion_y}')

```

```
# Crear un gráfico de dispersión
plt.figure(figsize=(10, 6))
sns.scatterplot(x='review_time', y='review_palate', data=df,
alpha=0.7)
plt.title('Gráfico de Dispersión entre X e Y')
plt.xlabel('review_time')
plt.ylabel('review_palate')
plt.grid(True)
plt.show()
```

Varianza de X: 5859025964738176.0  
Desviación estándar de X: 76544274.53923759  
Varianza de Y: 0.4654218953246297  
Desviación estándar de Y: 0.6822183633739491



## Fase 3: Prepare Data

#Deletion of null data

In this stage of the project we will dedicate ourselves as a group to preparing the data in a better way to be able to have only the best.

En esta etapa del proyecto nos dedicaremos como grupo a preparar los datos de mejor forma para poder tener solo los mejores.

We eliminate data that is null from the CSV

Eliminamos los dato que sean nulos del CSV

The elimination of null data will help us better identify the information we need from the CSV. This is done in case there is null or missing data that affects our analysis.

La eliminación de los datos nulos nos ayudarán a identificar de mejor manera la información que necesitamos del csv, este se hace por si acaso llegara a haber un dato nulo o faltante que afecte a nuestro análisis

```
df.shape  
(176905, 13)  
  
df = df.dropna()  
-----  
----  
NameError                               Traceback (most recent call  
last)  
<ipython-input-6-1ac174be4d67> in <cell line: 1>()  
----> 1 df = df.dropna()  
  
NameError: name 'df' is not defined  
  
df.shape  
(86034, 13)
```

After this data removal, we can realize that at least 68,136 of the table were null. This will help us avoid mistakes in our detailed analysis of the beers.

Después de esta eliminación de datos, podemos darnos cuenta de que al menos 68,136 de la tabla eran nulos. Esto nos ayudará a no tener fallas en nuestro análisis detallado de las cervezas.

## normalización y estandarización de los datos

Standardization and normalization are necessary for the preprocessing of beer data, this process can significantly improve the performance and stability of our future models. Additionally, the data that we pass through these processes will be treated fairly and appropriately, facilitating more effective analysis and more accurate results.

La estandarización y normalización son necesarios para el preprocesamiento de los datos de las cervezas, este proceso puede mejorar significativamente el rendimiento y la estabilidad de nuestros futuros modelos. Además, los datos que pasemos por estos procesos serán tratados de manera justa y adecuada, facilitando un análisis más efectivo y resultados más precisos.

```
('review_overall','review_aroma', 'review_appearance', 'review_palate', 'review_taste',
'beer_abv')
```

```
#Normalizacion de los datos cerveciles
from sklearn.preprocessing import MinMaxScaler

# Crear el escalador
scaler = MinMaxScaler()

# Normalizar solo las columnas 'A' y 'B'
df[['review_overall', 'beer_abv', 'review_aroma',
'review_appearance' , 'review_palate' , 'review_taste' ]] =
scaler.fit_transform(df[['review_overall', 'beer_abv', 'review_aroma',
'review_appearance' , 'review_palate' , 'review_taste' ]])

# Mostrar el DataFrame con columnas normalizadas
print(df)
```

```
      brewery_id          brewery_name review_time \
0           10325        Vecchio Birraio 1.234818e+09
1           10325        Vecchio Birraio 1.235915e+09
2           10325        Vecchio Birraio 1.235917e+09
3           10325        Vecchio Birraio 1.234725e+09
4           1075   Caldera Brewing Company 1.293735e+09
...           ...
176899       3052 Hook Norton Brewery Co. Ltd. 1.207019e+09
176900       3052 Hook Norton Brewery Co. Ltd. 1.206228e+09
176901       3052 Hook Norton Brewery Co. Ltd. 1.177193e+09
176902       3052 Hook Norton Brewery Co. Ltd. 1.170833e+09
176903       3052 Hook Norton Brewery Co. Ltd. 1.146714e+09
```

```
      review_overall  review_aroma  review_appearance
review_profilename \
0                  0.3        0.250                 0.5
stcules
1                  0.6        0.375                 0.6
stcules
2                  0.6        0.375                 0.6
stcules
3                  0.6        0.500                 0.7
stcules
4                  0.8        0.875                 0.8
johnmichaelsen
...
...
176899             0.8        0.750                 0.7
Gueuzedude
176900             1.0        0.625                 0.9
oberon
176901             0.9        0.625                 0.6
```

```

Stephen63
176902          0.9        0.750        0.8
bark
176903          0.8        0.500        0.9
zavenx

              beer_style  review_palate  review_taste \
0                  Hefeweizen      0.125      0.125
1          English Strong Ale      0.500      0.500
2    Foreign / Export Stout      0.500      0.500
3      German Pilsener      0.375      0.500
4 American Double / Imperial IPA      0.750      0.875
...
176899          English Bitter      0.750      0.750
176900          English Bitter      0.625      0.625
176901          English Bitter      0.625      0.625
176902          English Bitter      0.750      0.750
176903      English Brown Ale      0.625      0.750

              beer_name  beer_abv  beer_beerid
0      Sausa Weizen  0.085863     47986.0
1          Red Moon  0.106678     48213.0
2  Black Horse Black Beer  0.111882     48215.0
3      Sausa Pils  0.085863     47969.0
4      Cauldron DIPA  0.132697     64883.0
...
176899          Hooky Bitter  0.061578     35153.0
176900          Hooky Bitter  0.061578     35153.0
176901          Hooky Bitter  0.061578     35153.0
176902          Hooky Bitter  0.061578     35153.0
176903      Copper Ale  0.082394     7265.0

```

[171395 rows x 13 columns]

```

#Estandarizacion de los datos cerveciles
from sklearn.preprocessing import StandardScaler

# Crear el escalador
scaler = StandardScaler()

# Estandarizar solo las columnas 'A' y 'B'
df[['review_overall', 'beer_abv', 'review_aroma',
'review_appearance', 'review_palate', 'review_taste']] =
scaler.fit_transform(df[['review_overall', 'beer_abv', 'review_aroma',
'review_appearance', 'review_palate', 'review_taste']])

# Mostrar el DataFrame con columnas estandarizadas
print(df)

```

|                | brewery_id | brewery_name                   | review_time   | \                 |     |
|----------------|------------|--------------------------------|---------------|-------------------|-----|
| 0              | 10325      | Vecchio Birraio                | 1.234818e+09  |                   |     |
| 1              | 10325      | Vecchio Birraio                | 1.235915e+09  |                   |     |
| 2              | 10325      | Vecchio Birraio                | 1.235917e+09  |                   |     |
| 3              | 10325      | Vecchio Birraio                | 1.234725e+09  |                   |     |
| 4              | 1075       | Caldera Brewing Company        | 1.293735e+09  |                   |     |
| ...            | ...        | ...                            | ...           | ...               |     |
| 176899         | 3052       | Hook Norton Brewery Co. Ltd.   | 1.207019e+09  |                   |     |
| 176900         | 3052       | Hook Norton Brewery Co. Ltd.   | 1.206228e+09  |                   |     |
| 176901         | 3052       | Hook Norton Brewery Co. Ltd.   | 1.177193e+09  |                   |     |
| 176902         | 3052       | Hook Norton Brewery Co. Ltd.   | 1.170833e+09  |                   |     |
| 176903         | 3052       | Hook Norton Brewery Co. Ltd.   | 1.146714e+09  |                   |     |
|                |            | review_overall                 | review_aroma  | review_appearance |     |
|                |            | review_profilename             | \             |                   |     |
| 0              |            | -3.438073                      | -2.695559     | -2.354764         |     |
| stcules        |            |                                |               |                   |     |
| 1              |            | -1.264599                      | -1.952210     | -1.508684         |     |
| stcules        |            |                                |               |                   |     |
| 2              |            | -1.264599                      | -1.952210     | -1.508684         |     |
| stcules        |            |                                |               |                   |     |
| 3              |            | -1.264599                      | -1.208862     | -0.662603         |     |
| stcules        |            |                                |               |                   |     |
| 4              |            | 0.184383                       | 1.021184      | 0.183477          |     |
| johnmichaelsen |            |                                |               |                   |     |
| ...            | ...        | ...                            | ...           | ...               |     |
| ...            | ...        | ...                            | ...           | ...               |     |
| 176899         |            | 0.184383                       | 0.277836      | -0.662603         |     |
| Gueuzedude     |            |                                |               |                   |     |
| 176900         |            | 1.633365                       | -0.465513     | 1.029558          |     |
| oberon         |            |                                |               |                   |     |
| 176901         |            | 0.908874                       | -0.465513     | -1.508684         |     |
| Stephen63      |            |                                |               |                   |     |
| 176902         |            | 0.908874                       | 0.277836      | 0.183477          |     |
| bark           |            |                                |               |                   |     |
| 176903         |            | 0.184383                       | -1.208862     | 1.029558          |     |
| zavenx         |            |                                |               |                   |     |
|                |            | beer_style                     | review_palate | review_taste      | \   |
| 0              |            | Hefeweizen                     | -3.539804     | -3.388700         |     |
| 1              |            | English Strong Ale             | -1.243704     | -1.244187         |     |
| 2              |            | Foreign / Export Stout         | -1.243704     | -1.244187         |     |
| 3              |            | German Pilsener                | -2.009070     | -1.244187         |     |
| 4              |            | American Double / Imperial IPA | 0.287030      | 0.900325          |     |
| ...            |            | ...                            | ...           | ...               | ... |
| 176899         |            | English Bitter                 | 0.287030      | 0.185488          |     |
| 176900         |            | English Bitter                 | -0.478337     | -0.529350         |     |
| 176901         |            | English Bitter                 | -0.478337     | -0.529350         |     |
| 176902         |            | English Bitter                 | 0.287030      | 0.185488          |     |
| 176903         |            | English Brown Ale              | -0.478337     | 0.185488          |     |

```

          beer_name  beer_abv  beer_beerid
0           Sausa Weizen -1.013139      47986.0
1             Red Moon -0.480154      48213.0
2    Black Horse Black Beer -0.346908      48215.0
3           Sausa Pils -1.013139      47969.0
4        Cauldron DIPA  0.186077      64883.0
...
176899         Hooky Bitter -1.634955      35153.0
176900         Hooky Bitter -1.634955      35153.0
176901         Hooky Bitter -1.634955      35153.0
176902         Hooky Bitter -1.634955      35153.0
176903       Copper Ale -1.101970      7265.0

```

[171395 rows x 13 columns]

After better optimizing the data that we as a group considered main, we will begin to graph and size the information extracted from it,

Despues de optimizar de mejor manera los datos que como grupo concideramos principales, comenzaremos a graficar y dimensionar la informacion extraida de estos mismo,

## Outliers data processing

For this section of the dataset preparation we take the data to eliminate those that are Outliers. We use an IQR-based treatment for outliers.

## Tratamiento de datos Outliers

Para esta sección de la preparación del dataset tomamos los datos para eliminar los que sean Outliers. Utilizamos un tratamiento basado en el método IQR para los valores atípicos.

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Muestra las primeras filas del dataframe para verificar su contenido
print("Datos originales:")
print(df.head())

data = df['review_appearance']

# Calcular los cuartiles
Q1 = data.quantile(0.25)

```

```

Q3 = data.quantile(0.75)

# Calcular el rango intercuartílico (IQR)
IQR = Q3 - Q1

# Definir los límites para los valores válidos
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtrar los datos para eliminar outliers
filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]

# Crear un nuevo DataFrame con los datos sin los outliers
filtered_df = df[(df['review_appearance'] >= lower_bound) &
                  (df['review_appearance'] <= upper_bound)]

# Crear gráficos de dispersión con los datos
plt.figure(figsize=(12, 6))

# Gráfico de dispersión de datos originales
plt.subplot(1, 2, 1)
sns.scatterplot(x=np.arange(len(data)), y=data, color='blue',
                 label='Originales')
plt.axhline(y=lower_bound, color='r', linestyle='--', label='Límite inferior')
plt.axhline(y=upper_bound, color='r', linestyle='--', label='Límite superior')
plt.title('Datos Originales')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

# Gráfico de dispersión de datos filtrados
plt.subplot(1, 2, 2)
sns.scatterplot(x=np.arange(len(filtered_data)), y=filtered_data,
                 color='green', label='Filtrados')
plt.title('Datos Despues de Filtrar Outliers')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

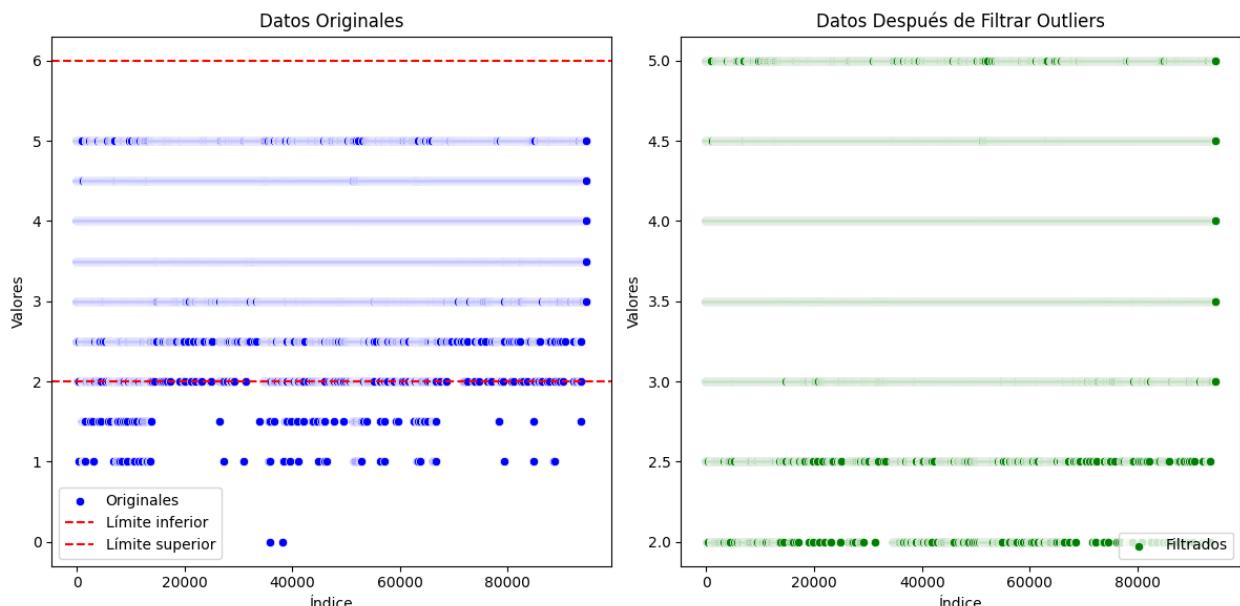
plt.tight_layout()
plt.show()

```

Datos originales:

|   | brewery_id | brewery_name    | review_time | review_overall | \ |
|---|------------|-----------------|-------------|----------------|---|
| 0 | 10325      | Vecchio Birraio | 1234817823  | 1.5            |   |
| 1 | 10325      | Vecchio Birraio | 1235915097  | 3.0            |   |
| 2 | 10325      | Vecchio Birraio | 1235916604  | 3.0            |   |
| 3 | 10325      | Vecchio Birraio | 1234725145  | 3.0            |   |

|   |                                |                         |                    |     |
|---|--------------------------------|-------------------------|--------------------|-----|
| 4 | 1075                           | Caldera Brewing Company | 1293735206         | 4.0 |
|   | review_aroma                   | review_appearance       | review_profilename | \   |
| 0 | 2.0                            | 2.5                     | stcules            |     |
| 1 | 2.5                            | 3.0                     | stcules            |     |
| 2 | 2.5                            | 3.0                     | stcules            |     |
| 3 | 3.0                            | 3.5                     | stcules            |     |
| 4 | 4.5                            | 4.0                     | johnmichaelsen     |     |
|   | beer_style                     | review_palate           | review_taste       | \   |
| 0 | Hefeweizen                     | 1.5                     | 1.5                |     |
| 1 | English Strong Ale             | 3.0                     | 3.0                |     |
| 2 | Foreign / Export Stout         | 3.0                     | 3.0                |     |
| 3 | German Pilsener                | 2.5                     | 3.0                |     |
| 4 | American Double / Imperial IPA | 4.0                     | 4.5                |     |
|   | beer_name                      | beer_abv                | beer_beerid        |     |
| 0 | Sausa Weizen                   | 5.0                     | 47986.0            |     |
| 1 | Red Moon                       | 6.2                     | 48213.0            |     |
| 2 | Black Horse Black Beer         | 6.5                     | 48215.0            |     |
| 3 | Sausa Pils                     | 5.0                     | 47969.0            |     |
| 4 | Cauldron DIPA                  | 7.7                     | 64883.0            |     |



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Muestra las primeras filas del dataframe para verificar su contenido
print("Datos originales:")

```

```

print(df.head())

data = df['review_overall']

# Calcular los cuartiles
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)

# Calcular el rango intercuartílico (IQR)
IQR = Q3 - Q1

# Definir los límites para los valores válidos
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtrar los datos para eliminar outliers
filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]

# Crear un nuevo DataFrame con los datos sin outliers
filtered_df = df[(df['review_overall'] >= lower_bound) &
                  (df['review_overall'] <= upper_bound)]

# Crear gráficos de dispersión
plt.figure(figsize=(12, 6))

# Gráfico de dispersión de datos originales
plt.subplot(1, 2, 1)
sns.scatterplot(x=np.arange(len(data)), y=data, color='blue',
label='Originales')
plt.axhline(y=lower_bound, color='r', linestyle='--', label='Límite
inferior')
plt.axhline(y=upper_bound, color='r', linestyle='--', label='Límite
superior')
plt.title('Datos Originales')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

# Gráfico de dispersión de datos filtrados
plt.subplot(1, 2, 2)
sns.scatterplot(x=np.arange(len(filtered_data)), y=filtered_data,
color='green', label='Filtrados')
plt.title('Datos Despues de Filtrar Outliers')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

plt.tight_layout()
plt.show()

```

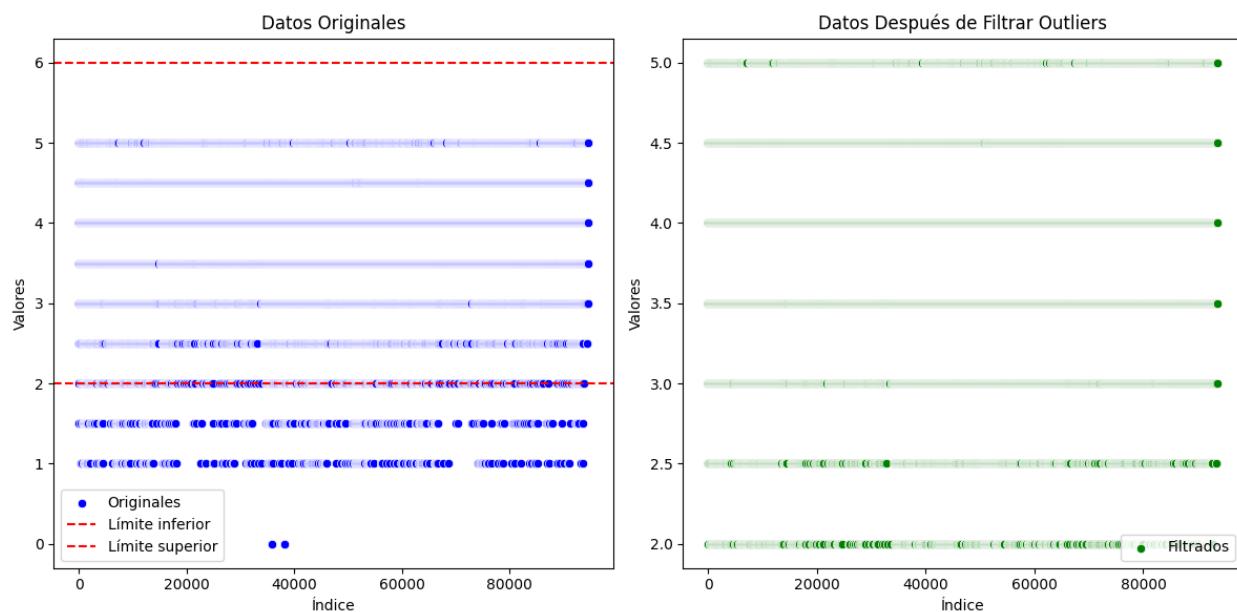
Datos originales:

```
brewery_id      brewery_name review_time review_overall \
0      10325      Vecchio Birraio 1234817823          1.5
1      10325      Vecchio Birraio 1235915097          3.0
2      10325      Vecchio Birraio 1235916604          3.0
3      10325      Vecchio Birraio 1234725145          3.0
4      1075  Caldera Brewing Company 1293735206          4.0
```

```
review_aroma  review_appearance review_profilename \
0            2.0                  2.5           stcules
1            2.5                  3.0           stcules
2            2.5                  3.0           stcules
3            3.0                  3.5           stcules
4            4.5                  4.0  johnmichaelsen
```

```
beer_style  review_palate  review_taste \
0      Hefeweizen          1.5          1.5
1      English Strong Ale      3.0          3.0
2      Foreign / Export Stout      3.0          3.0
3      German Pilsener          2.5          3.0
4  American Double / Imperial IPA      4.0          4.5
```

```
beer_name  beer_abv  beer_beerid
0      Sausa Weizen      5.0      47986.0
1      Red Moon          6.2      48213.0
2  Black Horse Black Beer      6.5      48215.0
3      Sausa Pils          5.0      47969.0
4      Cauldron DIPA        7.7      64883.0
```



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Muestra las primeras filas del dataframe para verificar su contenido
print("Datos originales:")
print(df.head())


data = df['review_taste']

# Calcular los cuartiles
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)

# Calcular el rango intercuartílico (IQR)
IQR = Q3 - Q1

# Definir los límites para los valores válidos
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtrar los datos para eliminar outliers
filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]

# Crear un nuevo DataFrame con los datos sin outliers
filtered_df = df[(df['review_taste'] >= lower_bound) &
                  (df['review_taste'] <= upper_bound)]


# Crear gráficos de dispersión
plt.figure(figsize=(12, 6))

# Gráfico de dispersión de datos originales
plt.subplot(1, 2, 1)
sns.scatterplot(x=np.arange(len(data)), y=data, color='blue',
label='Originales')
plt.axhline(y=lower_bound, color='r', linestyle='--', label='Límite
inferior')
plt.axhline(y=upper_bound, color='r', linestyle='--', label='Límite
superior')
plt.title('Datos Originales')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

# Gráfico de dispersión de datos filtrados
plt.subplot(1, 2, 2)
sns.scatterplot(x=np.arange(len(filtered_data)), y=filtered_data,
color='green', label='Filtrados')
plt.title('Datos Despues de Filtrar Outliers')

```

```

plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

plt.tight_layout()
plt.show()

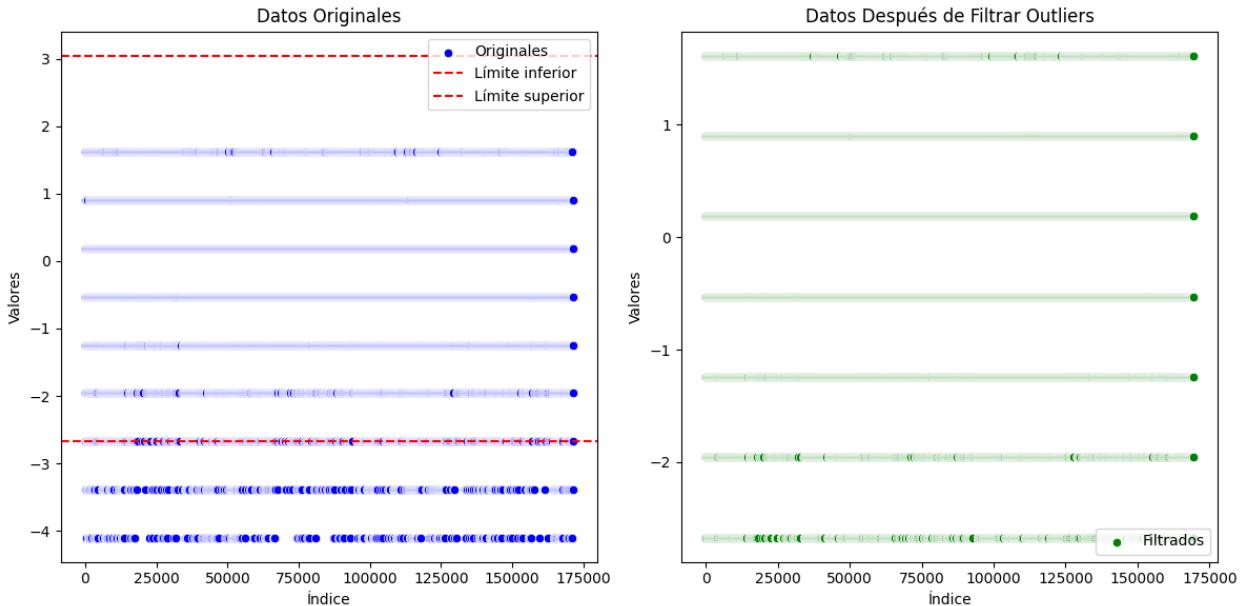
Datos originales:
      brewery_id      brewery_name  review_time
review_overall \
0          10325        Vecchio Birraio  1.234818e+09     -3.438073
1          10325        Vecchio Birraio  1.235915e+09     -1.264599
2          10325        Vecchio Birraio  1.235917e+09     -1.264599
3          10325        Vecchio Birraio  1.234725e+09     -1.264599
4          1075  Caldera Brewing Company  1.293735e+09      0.184383

      review_aroma  review_appearance review_profilename \
0      -2.695559           -2.354764             stcules
1      -1.952210           -1.508684             stcules
2      -1.952210           -1.508684             stcules
3      -1.208862           -0.662603             stcules
4       1.021184            0.183477   johnmichaelsen

      beer_style  review_palate  review_taste \
0      Hefeweizen      -3.539804     -3.388700
1      English Strong Ale      -1.243704     -1.244187
2      Foreign / Export Stout      -1.243704     -1.244187
3      German Pilsener      -2.009070     -1.244187
4  American Double / Imperial IPA      0.287030      0.900325

      beer_name  beer_abv  beer_beerid
0      Sausa Weizen -1.013139      47986.0
1      Red Moon    -0.480154      48213.0
2  Black Horse Black Beer -0.346908      48215.0
3      Sausa Pils    -1.013139      47969.0
4      Cauldron DIPA    0.186077      64883.0

```



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Muestra las primeras filas del dataframe para verificar su contenido
print("Datos originales:")
print(df.head())

data = df['review_palate']

# Calcular los cuartiles
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)

# Calcular el rango intercuartílico (IQR)
IQR = Q3 - Q1

# Definir los límites para los valores válidos
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtrar los datos para eliminar outliers
filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]

# Crear un nuevo DataFrame con los datos sin outliers
filtered_df = df[(df['review_palate'] >= lower_bound) &
                  (df['review_palate'] <= upper_bound)]

# Crear gráficos de dispersión

```

```

plt.figure(figsize=(12, 6))

# Gráfico de dispersión de datos originales
plt.subplot(1, 2, 1)
sns.scatterplot(x=np.arange(len(data)), y=data, color='blue',
label='Originales')
plt.axhline(y=lower_bound, color='r', linestyle='--', label='Límite
inferior')
plt.axhline(y=upper_bound, color='r', linestyle='--', label='Límite
superior')
plt.title('Datos Originales')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

# Gráfico de dispersión de datos filtrados
plt.subplot(1, 2, 2)
sns.scatterplot(x=np.arange(len(filtered_data)), y=filtered_data,
color='green', label='Filtrados')
plt.title('Datos Después de Filtrar Outliers')
plt.xlabel('Índice')
plt.ylabel('Valores')
plt.legend()

plt.tight_layout()
plt.show()

```

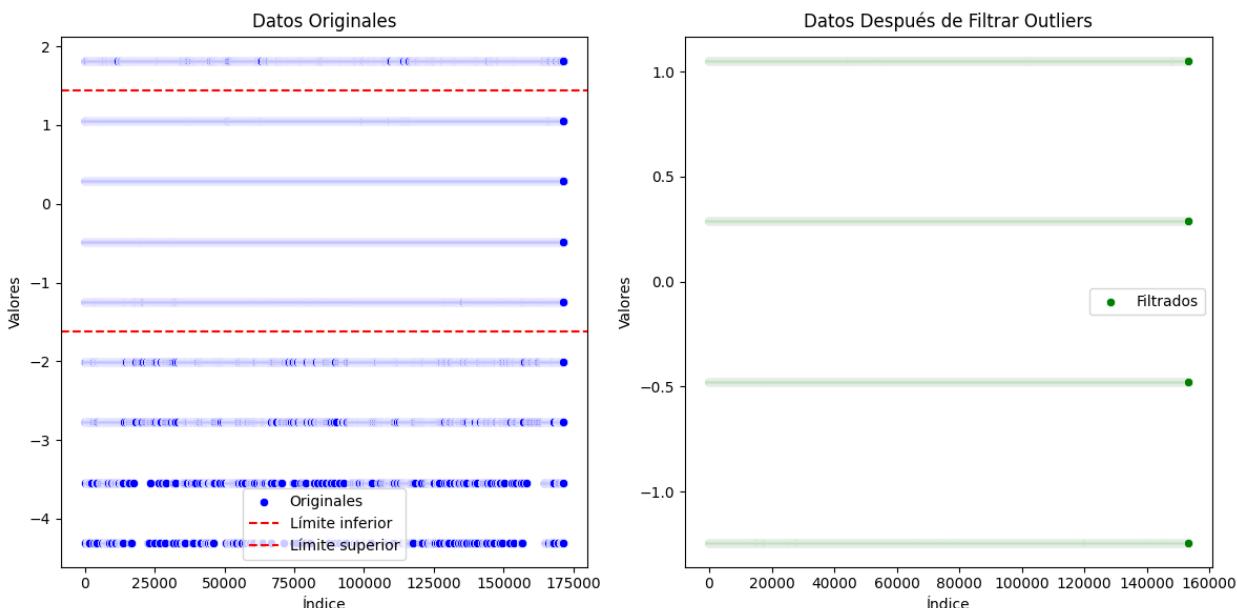
Datos originales:

|                    | brewery_id | brewery_name            | review_time  |           |
|--------------------|------------|-------------------------|--------------|-----------|
| review_overall \ 0 | 10325      | Vecchio Birraio         | 1.234818e+09 | -3.438073 |
| 1                  | 10325      | Vecchio Birraio         | 1.235915e+09 | -1.264599 |
| 2                  | 10325      | Vecchio Birraio         | 1.235917e+09 | -1.264599 |
| 3                  | 10325      | Vecchio Birraio         | 1.234725e+09 | -1.264599 |
| 4                  | 1075       | Caldera Brewing Company | 1.293735e+09 | 0.184383  |

|   | review_aroma | review_appearance | review_profilename | \ |
|---|--------------|-------------------|--------------------|---|
| 0 | -2.695559    | -2.354764         | stcules            |   |
| 1 | -1.952210    | -1.508684         | stcules            |   |
| 2 | -1.952210    | -1.508684         | stcules            |   |
| 3 | -1.208862    | -0.662603         | stcules            |   |
| 4 | 1.021184     | 0.183477          | johnmichaelsen     |   |

|   | beer_style | review_palate | review_taste | \ |
|---|------------|---------------|--------------|---|
| 0 | Hefeweizen | -3.539804     | -3.388700    |   |

|   |                                |           |             |
|---|--------------------------------|-----------|-------------|
| 1 | English Strong Ale             | -1.243704 | -1.244187   |
| 2 | Foreign / Export Stout         | -1.243704 | -1.244187   |
| 3 | German Pilsener                | -2.009070 | -1.244187   |
| 4 | American Double / Imperial IPA | 0.287030  | 0.900325    |
|   |                                |           |             |
| 0 | beer_name                      | beer_abv  | beer_beerid |
| 0 | Sausa Weizen                   | -1.013139 | 47986.0     |
| 1 | Red Moon                       | -0.480154 | 48213.0     |
| 2 | Black Horse Black Beer         | -0.346908 | 48215.0     |
| 3 | Sausa Pils                     | -1.013139 | 47969.0     |
| 4 | Cauldron DIPA                  | 0.186077  | 64883.0     |



## Mapas de calor

We graph using heat maps to see which data is so correlated and which is not

Graficamos mediante mapas de calor para ver que datos es tan correlacionados y cuales no

```
# Importar las bibliotecas necesarias
import seaborn as sns
import matplotlib.pyplot as plt

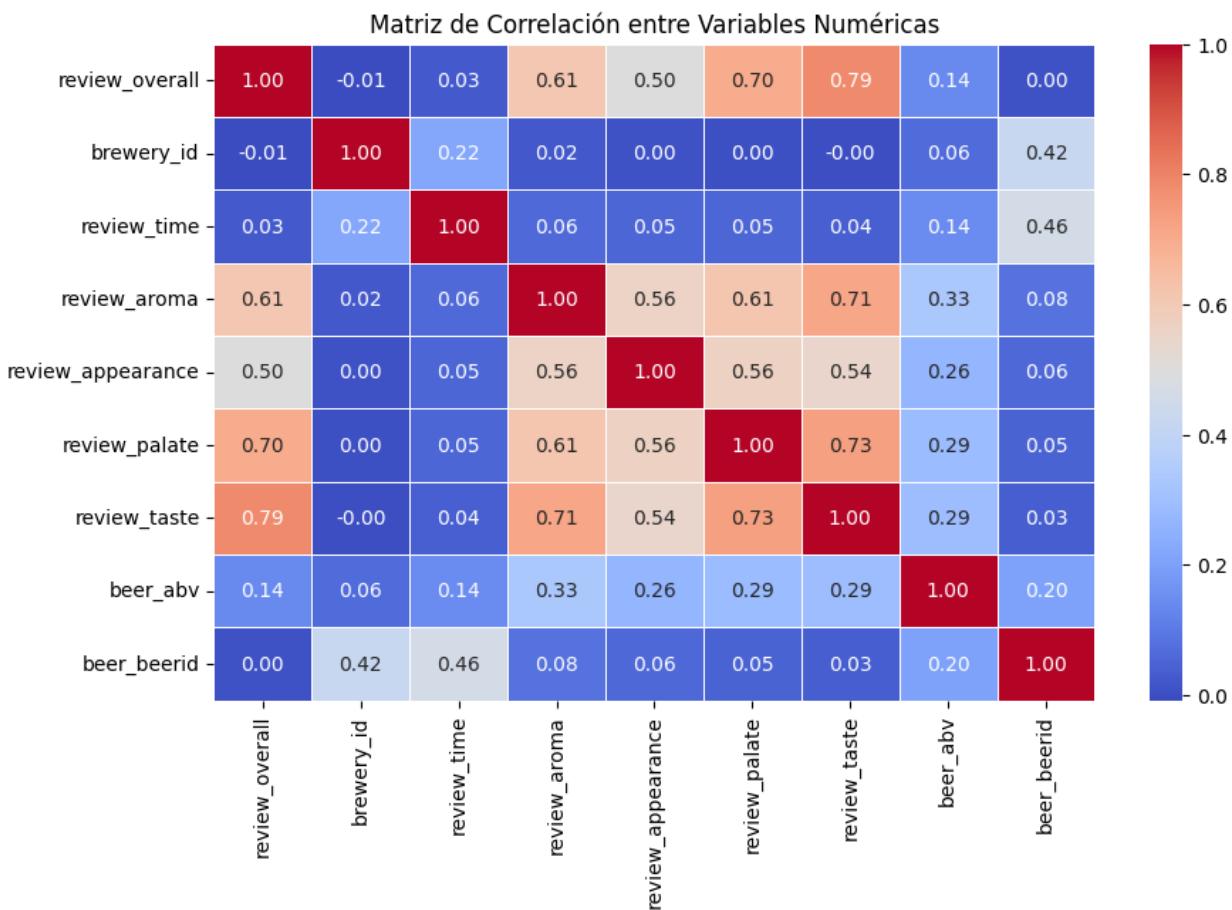
# Seleccionar las columnas num ricas para la correlaci n
numeric_columns = ['review_overall', 'brewery_id',
'review_time', 'review_aroma', 'review_appearance', 'review_palate',
'review_taste', 'beer_abv', 'beer_beerid']

# Calcular la matriz de correlaci n
correlation_matrix = df[numeric_columns].corr()
```

```

# Crear la matriz de calor
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
            linewidths=0.5, fmt=".2f")
plt.title('Matriz de Correlación entre Variables Numéricas')
plt.show()

```



```

# Importar las bibliotecas necesarias
import seaborn as sns
import matplotlib.pyplot as plt

# Seleccionar las columnas numéricas para la correlación
numeric_columns = ['review_overall', 'review_aroma',
                    'review_appearance',
                    'review_palate', 'review_taste', 'beer_abv']

# Calcular la matriz de correlación
correlation_matrix = df[numeric_columns].corr()

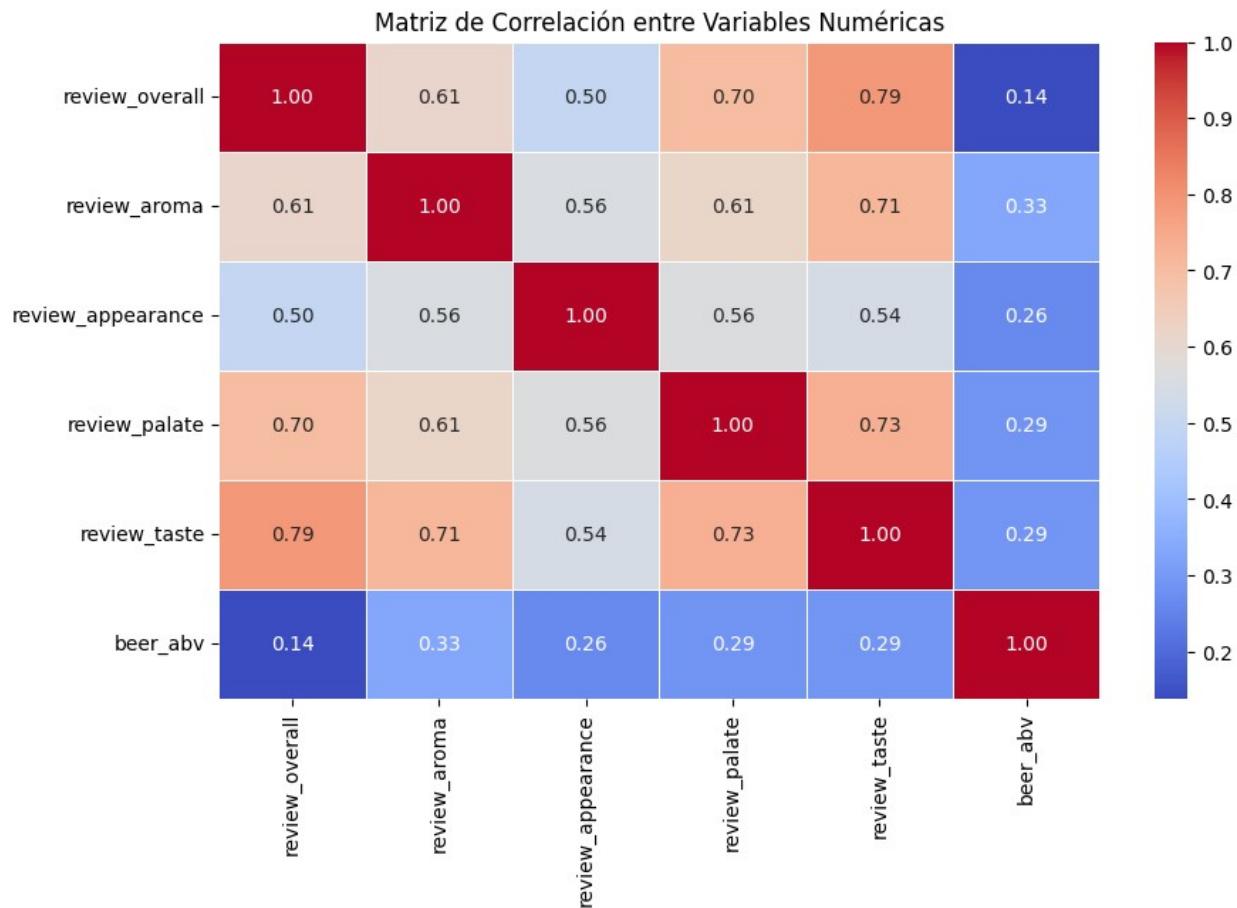
# Crear la matriz de calor

```

```

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
            linewidths=0.5, fmt=".2f")
plt.title('Matriz de Correlación entre Variables Numéricas')
plt.show()

```



We removed review\_profilename since for us as a team it is not necessary for the future of the project.

Eliminamos review\_profilename ya que para nosotros como equipo no es necesario para el futuro del proyecto.

```

# Eliminar filas con valores NaN
df = df.dropna()

# Verificar las columnas disponibles
print("Columnas disponibles:", df.columns)

# Eliminar la columna "review_profilename"
df.drop(["review_profilename"], axis=1, inplace=True)

```

```
# Mostrar las primeras filas del DataFrame procesado
display(df.head())

Columnas disponibles: Index(['brewery_id', 'brewery_name',
   'review_time', 'review_overall',
   'review_aroma', 'review_appearance', 'review_profilename',
   'beer_style',
   'review_palate', 'review_taste', 'beer_name', 'beer_abv',
   'beer_beerid'],
  dtype='object')

      brewery_id      brewery_name  review_time  review_overall \
0            10325    Vecchio Birraio  1234817823           1.5
1            10325    Vecchio Birraio  1235915097           3.0
2            10325    Vecchio Birraio  1235916604           3.0
3            10325    Vecchio Birraio  1234725145           3.0
4            1075  Caldera Brewing Company  1293735206           4.0

      review_aroma  review_appearance          beer_style \
0             2.0                 2.5        Hefeweizen
1             2.5                 3.0  English Strong Ale
2             2.5                 3.0  Foreign / Export Stout
3             3.0                 3.5       German Pilsener
4             4.5                 4.0  American Double / Imperial IPA

      review_palate  review_taste          beer_name  beer_abv
beer_beerid
0                  1.5            1.5        Sausa Weizen      5.0
47986
1                  3.0            3.0        Red Moon        6.2
48213
2                  3.0            3.0  Black Horse Black Beer      6.5
48215
3                  2.5            3.0        Sausa Pils      5.0
47969
4                  4.0            4.5      Cauldron DIPA      7.7
64883

df.columns

Index(['brewery_id', 'brewery_name', 'review_time', 'review_overall',
   'review_aroma', 'review_appearance', 'beer_style',
   'review_palate',
   'review_taste', 'beer_name', 'beer_abv', 'beer_beerid'],
  dtype='object')
```

#Data leak In this part we take review\_overall as the main data to filter the beer data and thus answer the question of which of the beers is the best.

#Filtración de datos En esta parte tomamos como dato principal el de review\_overall para filtrar los datos de las cerveza y así responder a la pregunta de cuál de las cervezas es la mejor.

In this data filtering we only take into account data that is greater than or equal to 4.5, leaving us with a total of 403,000 data, which is quite a lot.

En este filtrado de datos solo tomamos en cuenta los datos que sean mayor o igual a 4.5 dejándonos con un total de 403.000 datos lo cual es bastante.

## Filtracion de cervezas por graduacion alcoholica

```
#Filtrando datos con el beer_abv para cervezas suaves
Cervezas_Suave = df[df['beer_abv'] >=3.5]

Cervezas_Suave

{"summary": {"name": "Cervezas_Suave", "rows": 816, "fields": [{"column": "brewery_id", "properties": {"dtype": "number", "std": 4666, "min": 395, "max": 16866, "num_unique_values": 5, "samples": [6513, 848, 16866], "semantic_type": "\\", "description": "\n"}, "properties": {"column": "brewery_name", "dtype": "category", "num_unique_values": 5, "samples": ["Schorschbr\u00e4u", "Bristol Brewing Company", "The Bruery"], "semantic_type": "\\", "description": "\n"}, "properties": {"column": "review_time", "dtype": "number", "std": 34575564.84949963, "min": 1169872678.0, "max": 1326138083.0, "num_unique_values": 816, "samples": [1264383426.0, 1256755372.0, 1248741821.0], "semantic_type": "\\", "description": "\n"}, "properties": {"column": "review_overall", "dtype": "number", "std": 1.121580149420226, "min": -4.1625638118932695, "max": 1.6333651003337686, "num_unique_values": 9, "samples": [-0.5401082417513705, -3.43807269786489, 1.6333651003337686], "semantic_type": "\\", "description": "\n"}, "properties": {"column": "review_aroma", "dtype": "number", "std": 0.8475518522943241, "min": -2.695559069095713, "max": 1.7645331885600606, "num_unique_values": 7, "samples": [-0.46551294026782625, 1.7645331885600606], "semantic_type": "\\", "description": "\n"}, "properties": {"column": "review_appearance", "dtype": "number", "std": 0.9444845614186712, "min": -4.893005613813422, "max": 1.875638494487823, "num_unique_values": 7, "samples": [-0.46551294026782625, 1.7645331885600606], "semantic_type": "\\", "description": "\n"}}, "description": "\n"}}
```

```

    "num_unique_values": 9, "samples": [
        -0.6626030461251434,
        1.0295579809501676
    ], "semantic_type": "\\", "column": "\"beer_style\"", "properties": {
        "num_unique_values": 6, "samples": [
            "\"American Double / Imperial Stout\"",
            "\"Doppelbock\"",
            "\"Old Ale\""
        ], "description": "\\", "semantic_type": "\\", "column": "\"review_palate\"", "properties": {
            "dtype": "number", "std": 1.0164906802441886, "min": -4.305170537561391, "max": 1.8177632813571962, "num_unique_values": 9, "samples": [
                -3.539803810196568,
                0.28702982662754933,
                -2.774437082831745
            ], "description": "\\", "semantic_type": "\\", "column": "\"review_taste\"", "properties": {
                "dtype": "number", "std": 0.9509543880444422, "min": -3.3886997234031533, "max": 1.6151628240260614, "num_unique_values": 8, "samples": [
                    -0.5293496963007449,
                    1.6151628240260614,
                    0.18548781047485716
                ], "description": "\\", "semantic_type": "\\", "column": "\"beer_name\"", "properties": {
                "dtype": "category", "num_unique_values": 21, "samples": [
                    "\"Jeffersons Reserve Big Fella Bourbon Barrel Stout\"",
                    "\"Papier (Rye Whiskey Barrel)\"",
                    "\"Cherry Chocolate Rain\""
                ], "semantic_type": "\\", "description": "\\", "column": "\"beer_abv\"", "properties": {
                    "dtype": "number", "std": 1.131724840088745, "min": 3.6504797449727615, "max": 22.393787229761863, "num_unique_values": 12, "samples": [
                        4.316711053673796,
                        4.98294236237483,
                        3.872556847873106
                    ], "semantic_type": "\\", "description": "\\", "column": "\"beer_beerid\"", "properties": {
                    "dtype": "number", "std": 7006.325784120946, "min": 32963.0, "max": 75505.0, "num_unique_values": 21, "samples": [
                        45774.0,
                        61328.0
                    ], "semantic_type": "\\", "description": "\\", "column": "\"beer_abv\""
                }
            }, "type": "dataframe", "variable_name": "Cervezas_Suave"
        }
    ]
}

#Filtrando datos con el beer_abv para cervezas estandar
Cervezas_Estandar = df[df['beer_abv'] <=5.5]

```

Cervezas\_Estandar



```

\"dnichols\", \n          \"CRichardsMN\", \n          \"cosmicevan\"\n
], \n      \"semantic_type\": \"\", \n      \"description\": \"\"\n}
}, \n    { \n        \"column\": \"beer_style\", \n
\"properties\": { \n            \"dtype\": \"category\", \n
\"num_unique_values\": 7, \n            \"samples\": [\n
\"Eisbock\", \n                \"American Strong Ale\", \n                \"Russian\nImperial Stout\"\n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\"\n        } \n    }, \n    { \n        \"column\": \n
\"review_palate\", \n        \"properties\": { \n            \"dtype\": \n
\"number\", \n            \"std\": 1.2221064476633683, \n            \"min\": -\n4.021736116321637, \n            \"max\": 1.8414910779221663, \n
\"num_unique_values\": 9, \n            \"samples\": [\n
1.8230259184802111, \n                -2.5559293177606865, \n
1.0901225191997357 \n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\"\n        } \n    }, \n    { \n        \"column\": \n
\"review_taste\", \n        \"properties\": { \n            \"dtype\": \n
\"number\", \n            \"std\": 1.159996229006083, \n            \"min\": -\n3.8155427475159627, \n            \"max\": 1.6491667246946529, \n
\"num_unique_values\": 9, \n            \"samples\": [\n
3.8155427475159627, \n                -0.40009932738432796, \n
1.766276695436982 \n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\"\n        } \n    }, \n    { \n        \"column\": \n
\"beer_name\", \n        \"properties\": { \n            \"dtype\": \n
\"category\", \n            \"num_unique_values\": 20, \n
\"samples\": [\n
\"Schorshbr\\u00e4u Schorschbock 31%\", \n
\"Short's Anniversary Ale 2006\", \n                \"Black Damnation VI -\nMessy\" \n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\"\n        } \n    }, \n    { \n        \"column\": \n
\"beer_abv\", \n        \"properties\": { \n            \"dtype\": \n
\"number\", \n            \"std\": 2.2953517596103254, \n            \"min\": -\n5.579105488136417, \n            \"max\": 21.811437237576676, \n
\"num_unique_values\": 17, \n            \"samples\": [\n
10.255050389964566, \n                21.811437237576676, \n
7.731934632624516 \n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\"\n        } \n    }, \n    { \n        \"column\": \n
\"beer_beerid\", \n        \"properties\": { \n            \"dtype\": \n
\"number\", \n            \"std\": 17236, \n            \"min\": 1640, \n
\"max\": 74021, \n            \"num_unique_values\": 20, \n
\"samples\": [\n
51466, \n                35420, \n                70633 \n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\"\n        } \n    }
}, \n    \"type\": \"dataframe\", \n    \"variable_name\": \"Cervezas_Fuertes\"}

```

## Filtracion de las cervezas por review general

```
# Filtrar las filas donde 'review_overall' es mayor o igual a 4.5
filtered_df = df[df['review_overall'] >= 4.5]
```

filtered\_df

```
{"type": "dataframe", "variable_name": "filtered_df"}
```

For the second filtering of the data we took into account only the data that is greater than or equal to 5.0, since we wanted to see which were the best beers that were classified in the data.

Para el segundo filtrado de los datos tomamos en cuenta sólo los datos que sean mayor o igual a 5.0, ya que queríamos ver cuales eran las mejores cervezas que estaban clasificadas en los datos.

```
# Filtrar las filas donde 'review_overall' es mayor o igual a 5
filtered_df_2 = df[df['review_overall'] >= 5.0]

filtered_df_2

{"summary": {"name": "filtered_df_2", "rows": 91320, "fields": [{"column": "brewery_id", "properties": {"dtype": "number", "min": 1, "max": 27870, "num_unique_values": 2371, "samples": [198, 2312], "semantic_type": "\\", "description": "\n"}, {"column": "brewery_name", "properties": {"dtype": "category", "num_unique_values": 2346, "samples": ["Einbecker Brauhaus AG", "Elk Grove Brewing Co Inc."], "semantic_type": "\\", "description": "\n"}, {"column": "review_time", "properties": {"dtype": "number", "std": 75681482, "min": 896659201, "max": 1326272438, "num_unique_values": 91292, "samples": [1244601357, 1274972794, 1187733548], "semantic_type": "\\", "description": "\n"}, {"column": "review_overall", "properties": {"dtype": "number", "std": 0.0, "min": 5.0, "max": 5.0, "num_unique_values": 1, "samples": [5.0], "semantic_type": "\\", "description": "\n"}, {"column": "review_aroma", "properties": {"dtype": "number", "min": 1.0, "max": 5.0, "num_unique_values": 9, "samples": [1.5], "semantic_type": "\\", "description": "\n"}, {"column": "review_appearance", "properties": {"dtype": "number", "min": 1.0, "max": 5.0, "num_unique_values": 9, "samples": [1.5], "semantic_type": "\\", "description": "\n"}]}, "properties": {"semantic_type": "\\", "description": "\n"}}, "semantic_type": "\\", "description": "\n"}]
```

```

    },\n      {\n        \"column\": \"review_profilename\",\\n\n      \"properties\": {\n        \"dtype\": \"category\",\\n\n        \"num_unique_values\": 16271,\\n        \"samples\": [\n          \"pmcadamis\"\n        ],\\n        \"semantic_type\": \"\",\\n\n        \"description\": \"\"\\n      },\\n      {\n        \"column\": \"beer_style\",\\n\n        \"properties\": {\n          \"dtype\": \"category\",\\n\n          \"num_unique_values\": 104,\\n\n          \"samples\": [\n            \"Keller Bier / Zwickel Bier\"\n          ],\\n\n          \"semantic_type\": \"\",\\n\n          \"description\": \"\"\\n        }\n      },\\n      {\n        \"column\": \"review_palate\",\\n\n        \"properties\": {\n          \"dtype\": \"number\",\\n\n          \"std\": 0.4893853371885971,\\n\n          \"min\": 1.0,\\n\n          \"max\": 5.0,\\n\n          \"num_unique_values\": 9,\\n\n          \"samples\": [\n            1.5\n          ],\\n\n          \"semantic_type\": \"\",\\n\n          \"description\": \"\"\\n        }\n      },\\n      {\n        \"column\": \"review_taste\",\\n\n        \"properties\": {\n          \"dtype\": \"number\",\\n\n          \"std\": 0.45007960792813984,\\n\n          \"min\": 1.0,\\n\n          \"max\": 5.0,\\n\n          \"num_unique_values\": 9,\\n\n          \"samples\": [\n            1.5\n          ],\\n\n          \"semantic_type\": \"\",\\n\n          \"description\": \"\"\\n        }\n      },\\n      {\n        \"column\": \"beer_name\",\\n\n        \"properties\": {\n          \"dtype\": \"category\",\\n\n          \"num_unique_values\": 11802,\\n\n          \"samples\": [\n            \"Mash Eisbock\"\n          ],\\n\n          \"semantic_type\": \"\",\\n\n          \"description\": \"\"\\n        }\n      },\\n      {\n        \"column\": \"beer_abv\",\\n\n        \"properties\": {\n          \"dtype\": \"number\",\\n\n          \"std\": 2.5373844377373405,\\n\n          \"min\": 0.01,\\n\n          \"max\": 41.0,\\n\n          \"num_unique_values\": 338,\\n\n          \"samples\": [\n            10.03\n          ],\\n\n          \"semantic_type\": \"\",\\n\n          \"description\": \"\"\\n        }\n      },\\n      {\n        \"column\": \"beer_beerid\",\\n\n        \"properties\": {\n          \"dtype\": \"number\",\\n\n          \"std\": 20355,\\n\n          \"min\": 4,\\n\n          \"max\": 77303,\\n\n          \"num_unique_values\": 12436,\\n\n          \"samples\": [\n            6517\n          ],\\n\n          \"semantic_type\": \"\",\\n\n          \"description\": \"\"\\n        }\n      }\n    },\\n    \"type\": \"dataframe\",\\n    \"variable_name\": \"filtered_df_2\"\n  }

```

#CSV of the First data breach. In this part we place a possible download of the data leak that we have done previously, because they may be used in the future.

#CSV de la primera filtración de datos. En esta parte colocamos una posible descarga de la filtración de datos que hemos hecho anteriormente, debido a que los mismos pueden ser ocupados en un futuro.

```

# Guardar el DataFrame filtrado en un nuevo archivo CSV
filtered_df.to_csv('filtered_review_overall.csv', index=False)

# Descargar el archivo CSV filtrado
from google.colab import files

```

```
files.download('filtered_review_overall.csv')  
<IPython.core.display.Javascript object>  
<IPython.core.display.Javascript object>
```

## Segunda filtacion de datos

```
# Guardar el DataFrame filtrado en un nuevo archivo CSV  
#filtered_df_2.to_csv('filtered_review_overall_2.csv', index=False)  
  
# Descargar el archivo CSV filtrado  
#files.download('filtered_review_overall_2.csv')  
  
<IPython.core.display.Javascript object>  
<IPython.core.display.Javascript object>
```

## Summary of the first part of the project

In this dataset we take the data from the Kross brewery to be able to do a market analysis. While observing the data we found that there are several factors that can affect the classification of a beer, among the variables that most influence are flavor, aroma, palate and appearance. These 4 variables are the ones that impact the grades the most.

We were able to identify which of these variables are most correlated thanks to the Correlation Matrix. Because of this, we identify the variables that most affect a beer's score.

Regarding the methodology used, it is CRISP-DM, due to the advantages it brings such as:

- Clear structure: Defines an orderly process to handle large volumes of data.
- Completeness: Addresses the entire project life cycle, from business analysis to implementation.

Due to the factors mentioned above, we can identify which beers will sell better in the Chilean market without suffering losses due to possible defective products.

## Resumen de la primera parte del proyecto

En este dataset tomamos los datos de la cervecería Kross para poder hacer un análisis de mercado. Mientras observamos los datos encontramos que hay varios factores que pueden afectar en la clasificación de una cerveza entre las variables que más influyen son el sabor, aroma, paladar y apariencia. Estas 4 variables son las que más impactan las calificaciones.

Pudimos identificar cuales de estas variables se correlacionan en mayor cantidad gracias a la Matriz de correlacion. Debido a esto, identificamos las variales que más afectan el puntaje de una cerveza.

En cuanto a la metodología que utilizada es CRISP-DM, debido a las ventajas que esta trae como :

- Estructura clara: Define un proceso ordenado para manejar grandes volúmenes de datos.
- Compleitud: Aborda todo el ciclo de vida del proyecto, desde el análisis del negocio hasta la implementación.

Debido a los factores mencionados anteriormente podemos identificar que cervezas se venderán mejor en el mercado chileno sin sufrir perdidas por posibles productos defectuosos.

## Fase 4: modelar los datos

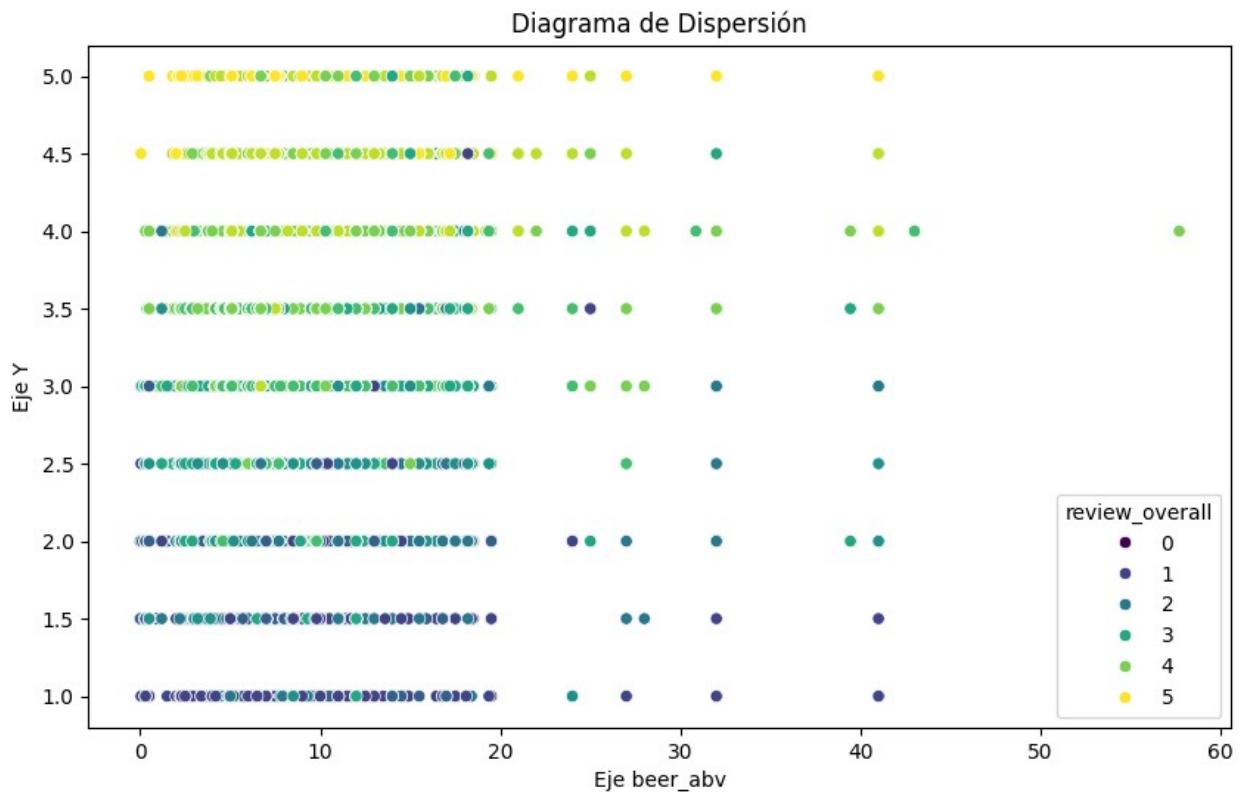
```
['review_overall', 'brewery_id', 'review_time', 'review_aroma',
'review_appearance', 'review_palate', 'review_taste',
'beer_abv', 'beer_beerid']

['review_overall',
'brewery_id',
'review_time',
'review_aroma',
'review_appearance',
'review_palate',
'review_taste',
'beer_abv',
'beer_beerid']
```

## Graficos de dispersión

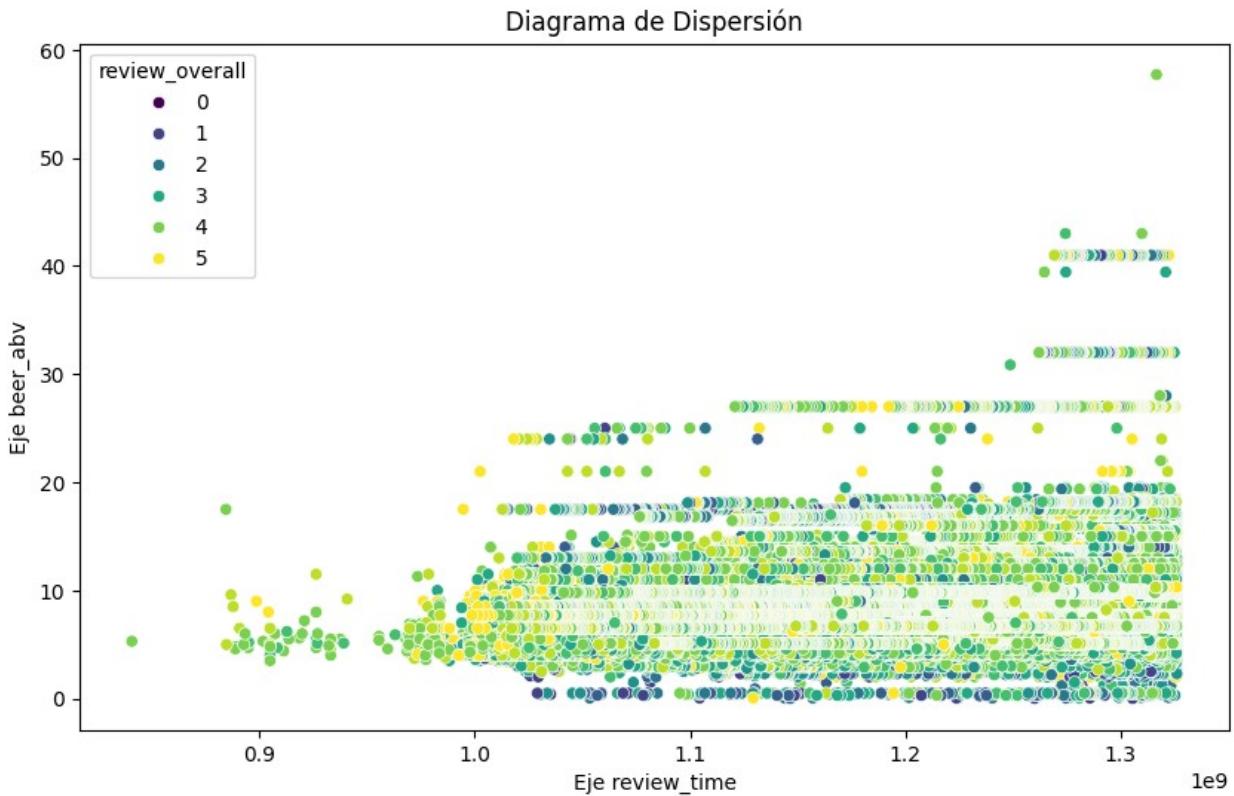
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Gráfico de dispersión
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='beer_abv', y='review_palate',
hue='review_overall', palette='viridis')
plt.title('Diagrama de Dispersión')
plt.xlabel('Eje beer_abv')
plt.ylabel('Eje Y')
plt.show()
```



#Gráfico de dispersión

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='review_time', y='beer_abv',
hue='review_overall', palette='viridis')
plt.title('Diagrama de Dispersion')
plt.xlabel('Eje review_time')
plt.ylabel('Eje beer_abv')
plt.show()
```



#Gráfico de dispersión

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='review_appearance', y='review_palate',
hue='review_overall', palette='viridis')
plt.title('Diagrama de Dispersión')
plt.xlabel('Eje review_time')
plt.ylabel('Eje beer_abv')
plt.show()
```

## Curva ROC

-Español Un AUC de 0,89 en una curva ROC señala un rendimiento excepcional del modelo en la clasificación binaria. Esto implica que el modelo es efectivo para diferenciar entre las clases, sobrepasando la habilidad aleatoria de discriminación. Por lo general, un AUC tan elevado indica una gran exactitud en las predicciones.

-Ingles An AUC of 0.89 on a ROC curve signals exceptional model performance in binary classification. This implies that the model is effective in differentiating between classes, surpassing random discrimination ability. Generally, such a high AUC indicates high prediction accuracy.

```
import pandas as pd
import numpy as np
```

```

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Convertir 'review_overall' a binario
threshold = 4
df['binary_review'] = (df['review_overall'] >= threshold).astype(int)

# Definir las características y la variable objetivo
X = df[['brewery_id', 'review_time', 'review_aroma',
'review_appearance', 'review_palate', 'review_taste', 'beer_abv']]
y = df['binary_review'] # Usar la nueva variable binaria como
objetivo

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

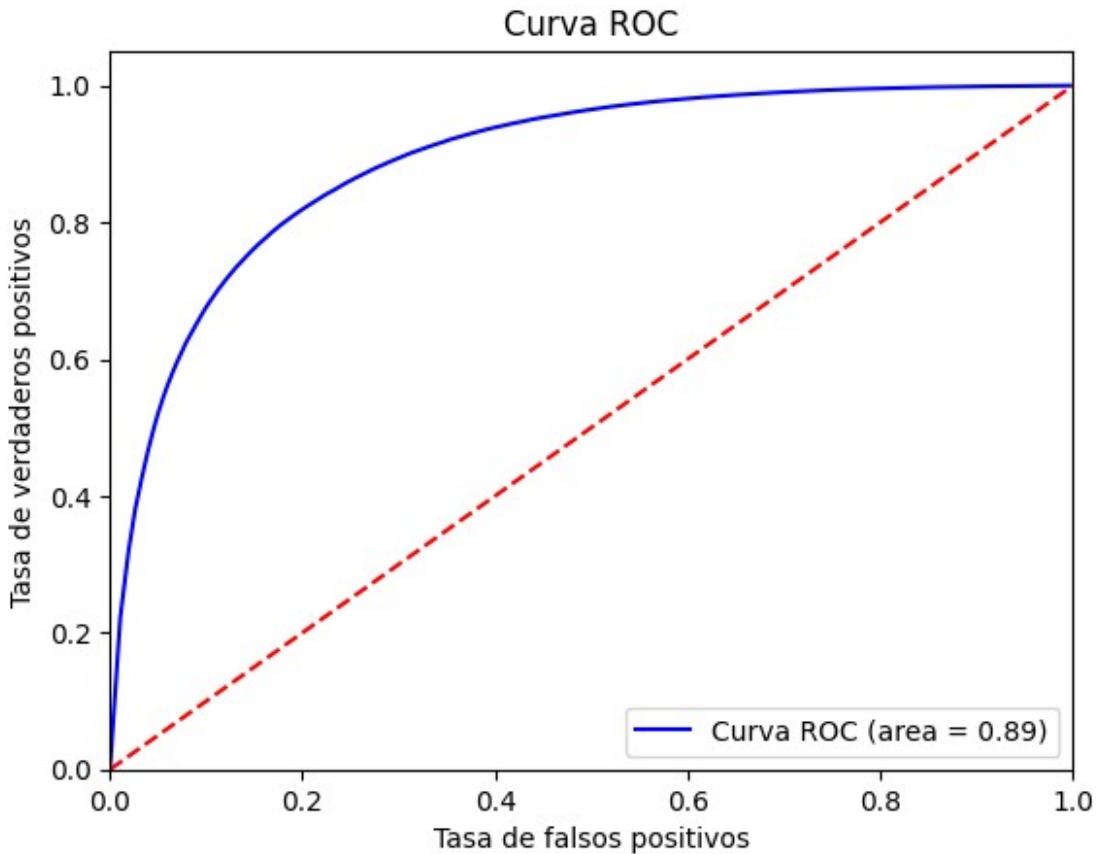
# Entrenar un clasificador
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predecir probabilidades
y_scores = model.predict_proba(X_test)[:, 1] # Probabilidades para la
clase positiva

# Calcular la curva ROC
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Graficar la curva ROC
plt.figure()
plt.plot(fpr, tpr, color='blue', label='Curva ROC (area = ' +
{:.2f}).format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de falsos positivos')
plt.ylabel('Tasa de verdaderos positivos')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.show()

```



```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.preprocessing import KBinsDiscretizer

n_bins = 3 # Puedes ajustar el número de categorías según necesites
discretizer = KBinsDiscretizer(n_bins=n_bins, encode='ordinal',
strategy='uniform')
y =
discretizer.fit_transform(df[['review_overall']]).astype(int).ravel()

# Seleccionar características
X = df[['review_aroma', 'review_palate', 'review_taste', 'beer_abv']]

# Divide los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

model = RandomForestClassifier()
model.fit(X_train, y_train)

```

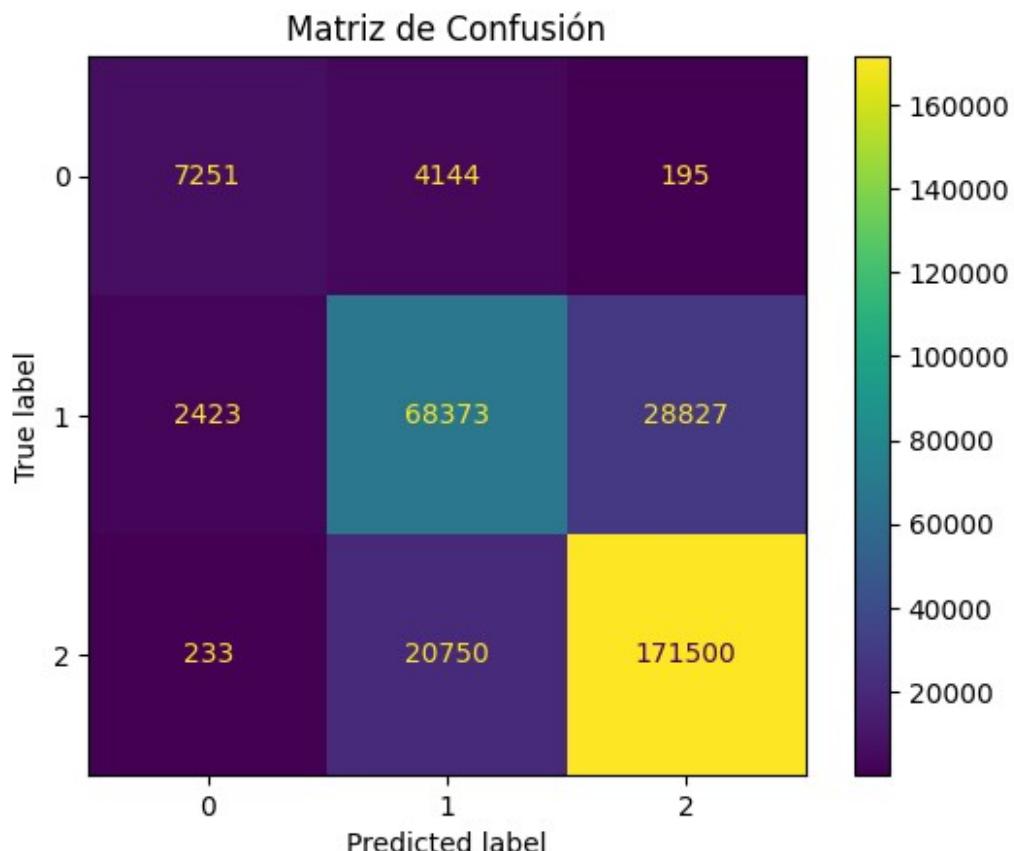
```

y_pred = model.predict(X_test)

# Generar la matriz de confusión
cm = confusion_matrix(y_test, y_pred)

# Mostrar la matriz de confusión
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Matriz de Confusión")
plt.show()

```



## Arbol de desicion

```

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn import tree

X = df[['beer_abv', 'review_palate']]
y = df['review_overall']

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=250)

from sklearn.tree import DecisionTreeRegressor

# Usar DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)

# Realiza predicciones
y_pred = regressor.predict(X_test)

# Evaluación (por ejemplo, usando el error cuadrático medio)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print('MSE:', mse)

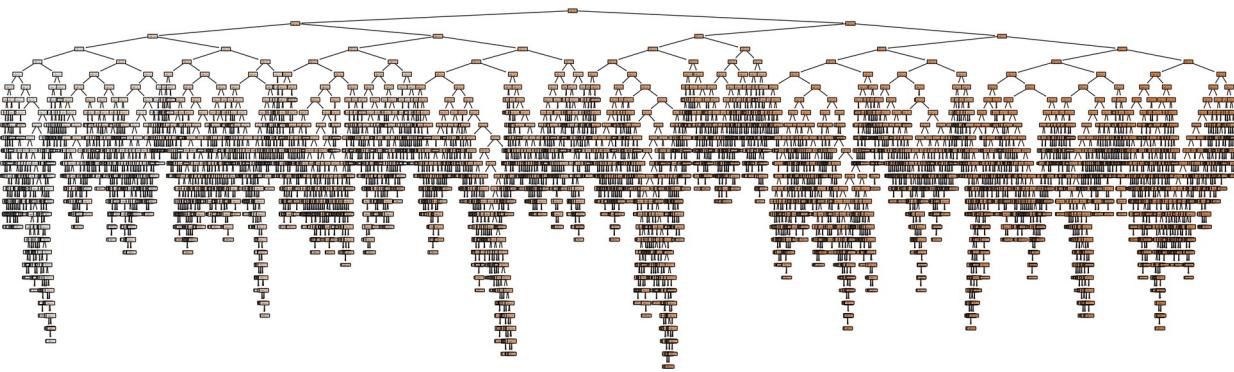
MSE: 0.25676838494066123

from sklearn.tree import DecisionTreeRegressor
#Si deseas predecir un valor continuo
# Cambiar a DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train) # Donde y_train es continuo

DecisionTreeRegressor()

plt.figure(figsize=(30, 9))
tree.plot_tree(regressor, filled=True)
plt.show()

```



## Arbol de desiciones 2: Review Overall

```

# Crear categorías a partir de la calificación general
bins = [0, 2, 4, 5] # Definir los cortes para cada categoría
labels = ['Baja', 'Media', 'Alta']

```

```

df['review_overall_cat'] = pd.cut(df['review_overall'], bins=bins,
labels=labels, include_lowest=True)

# Eliminar filas con valores nulos en la variable objetivo
df.dropna(subset=['review_overall_cat'], inplace=True)

# Definir las características (X) y la variable objetivo (y)
X = df.drop(['review_overall_cat', 'review_overall', 'beer_name',
'brewery_name'], axis=1)
y = df['review_overall_cat']

# One-Hot Encoding para variables categóricas restantes si es
necesario
X = pd.get_dummies(X, drop_first=True)

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Crear y entrenar el modelo de árbol de clasificación
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Hacer predicciones y evaluar el modelo
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Reporte de clasificación:\n", classification_report(y_test,
y_pred))

Accuracy: 0.7543766951988446
Reporte de clasificación:
      precision    recall  f1-score   support
      Alta       0.64     0.59     0.61      8512
      Baja       0.67     0.59     0.63      889
      Media      0.81     0.83     0.82     18988
      accuracy                           0.75     28389
      macro avg       0.70     0.67     0.69     28389
      weighted avg      0.75     0.75     0.75     28389

```

This classification tree provides a useful analysis that could be applied in a market analysis context for the brewery to evaluate the quality of its beers based on various sensory characteristics and general attributes. From the accuracy of 75.4% and the analysis of the three beer quality categories ("High", "Medium", "Low"), we can draw the following key conclusions:

## Product Segmentation by Quality:

This model can help the brewery segment its products into different quality categories (High, Medium, Low), based on beer characteristics such as aroma, flavor, appearance, and alcohol level. This would allow the company to position its products appropriately in the market, aimed at different consumer profiles.

By knowing the quality of beers and the characteristics that determine said quality, the marketing team could create campaigns specifically aimed at consumers who prefer "High" or "Medium" quality beers, adapting advertising strategies according to the preferences of the target audience. .

Conclusion: This classification model has the potential to be a valuable tool in strategic decision making within the beer market. By segmenting products by quality and analyzing the key attributes that impact that classification, the brewery can identify opportunities for improvement, refine its product offering, and develop more effective marketing strategies.

## ESPAÑOL:

Este árbol de clasificación proporciona un análisis útil que podría aplicarse en un contexto de análisis de mercado para la cervecería para que evalúe la calidad de sus cervezas en función de diversas características sensoriales y atributos generales. A partir de la precisión del 75.4% y el análisis de las tres categorías de calidad de la cerveza ("Alta", "Media", "Baja"), podemos extraer las siguientes conclusiones clave:

## Segmentación de Productos por Calidad:

Este modelo puede ayudar a la cervecería a segmentar sus productos en diferentes categorías de calidad (Alta, Media, Baja), basándose en las características de la cerveza como aroma, sabor, apariencia, y el nivel de alcohol. Esto permitiría a la empresa posicionar sus productos adecuadamente en el mercado, dirigidos a diferentes perfiles de consumidores.

Al conocer la calidad de las cervezas y las características que determinan dicha calidad, el equipo de marketing podría crear campañas dirigidas específicamente a los consumidores que prefieren cervezas de "Alta" o "Media" calidad, adaptando las estrategias publicitarias según las preferencias del público objetivo.

Conclusion: Este modelo de clasificación tiene el potencial de ser una herramienta valiosa en la toma de decisiones estratégicas dentro del mercado de la cerveza. Al segmentar productos por calidad y analizar los atributos clave que impactan esa clasificación, la cervecería puede identificar oportunidades de mejora, refinar su oferta de productos y desarrollar estrategias de marketing más efectivas.

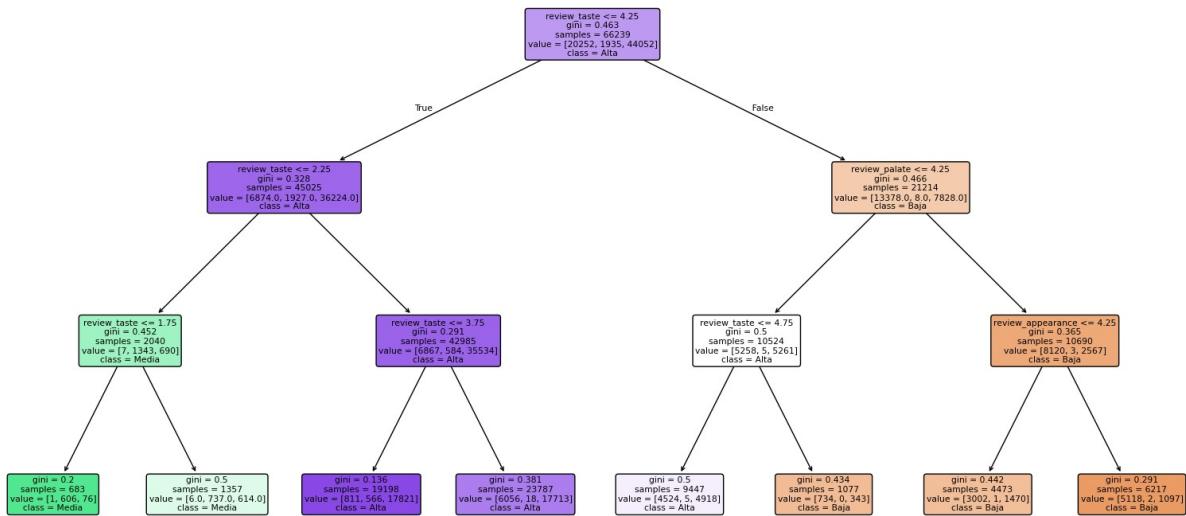
```
from sklearn.ensemble import RandomForestClassifier  
  
rf_model = RandomForestClassifier(random_state=42)  
rf_model.fit(X_train, y_train)  
  
RandomForestClassifier(random_state=42)  
  
import matplotlib.pyplot as plt  
from sklearn.tree import plot_tree
```

```

clf = DecisionTreeClassifier(max_depth=3) # Limita la profundidad
clf.fit(X_train, y_train)

# Visualizar el árbol de decisión
plt.figure(figsize=(20,10)) # Tamaño del gráfico
plot_tree(clf, filled=True, feature_names=X.columns,
          class_names=['Baja', 'Media', 'Alta'], rounded=True)
plt.show()

```



This decision tree can help make decisions based on selected attributes (such as `review_taste` and `review_palate`). For example, if a product has a taste rating less than or equal to 2.25, it is classified as "High." This can be helpful in understanding how product ratings are performing and where improvements can be made.

Este árbol de decisión puede ayudar a tomar decisiones basadas en los atributos seleccionados (como `review_taste` y `review_palate`). Por ejemplo, si un producto tiene una calificación de gusto menor o igual a 2.25, se clasifica como "Alta". Esto puede ser útil para entender cómo se comportan las calificaciones de los productos y dónde se pueden realizar mejoras.

```

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=None, min_samples_leaf=2,
                             min_samples_split=10, random_state=42)
clf.fit(X_train, y_train)

DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=10,
                      random_state=42)

import pandas as pd
from sklearn.model_selection import train_test_split

```

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Cargar los datos (asegúrate de cambiar la ruta si es necesario)
file_path =
'C:/Users/massr/Desktop/reviewsCerveceriaKedro/cerveceriar/data/01_raw
/beer_reviews.csv'
df = pd.read_csv(file_path, sep=",")

# Crear categorías a partir de la calificación general
bins = [0, 2, 4, 5] # Definir los cortes para cada categoría
labels = ['Baja', 'Media', 'Alta']
data['review_overall_cat'] = pd.cut(data['review_overall'], bins=bins,
labels=labels, include_lowest=True)

# Eliminar filas con valores nulos en la variable objetivo
data.dropna(subset=['review_overall_cat'], inplace=True)

# Definir las características (X) y la variable objetivo (y)
X = data.drop(['review_overall_cat', 'review_overall',
'review_profilename', 'beer_name', 'beer_style', 'brewery_name'],
axis=1)
y = data['review_overall_cat']

# One-Hot Encoding para variables categóricas restantes si es
necesario
X = pd.get_dummies(X, drop_first=True)

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Crear y entrenar el modelo de árbol de clasificación
clf = DecisionTreeClassifier(max_depth=4, min_samples_leaf=2,
random_state=42) # Ajusta la profundidad y el mínimo de muestras
clf.fit(X_train, y_train)

# Hacer predicciones y evaluar el modelo
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Reporte de clasificación:\n", classification_report(y_test,
y_pred))

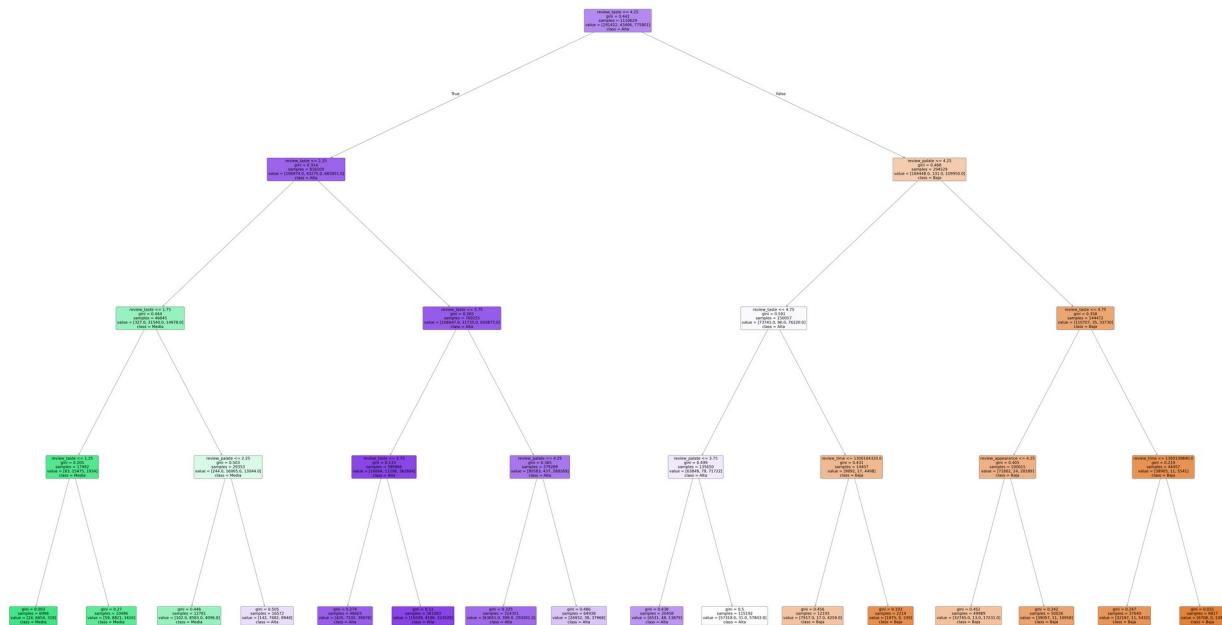
# Visualizar el árbol de decisión
plt.figure(figsize=(150,90)) # Tamaño del gráfico
plot_tree(clf, filled=True, feature_names=X.columns,
class_names=['Baja', 'Media', 'Alta'], rounded=True)
plt.show()

```

Accuracy: 0.790232885490089

Reporte de clasificación:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Alta         | 0.76      | 0.41   | 0.54     | 124283  |
| Baja         | 0.80      | 0.55   | 0.66     | 18755   |
| Media        | 0.79      | 0.94   | 0.86     | 332947  |
| accuracy     |           |        | 0.79     | 475985  |
| macro avg    | 0.79      | 0.64   | 0.68     | 475985  |
| weighted avg | 0.79      | 0.79   | 0.77     | 475985  |



Precision:

High: 0.70 Low: 0.76 Average: 0.82 Precision indicates the proportion of true positives among all positive predictions. The "Medium" category has the best accuracy, which is positive.

Recall:

High: 0.53 Low: 0.62 Average: 0.90 Recall measures the proportion of true positives out of the total number of real positive cases. The "Medium" category has a very high recall, while "High" is lower, which means that beers that are really "Medium" are being classified more "correctly", but some that are "High" are being missed. .

F1-Score:

High: 0.61 Low: 0.69 Average: 0.86 The F1-Score is the harmonic mean of precision and recall. The "Medium" category also stands out here, showing a good balance between precision and recall.

Accuracy:

The overall accuracy is 0.80, which means 80% of the predictions are correct. This is a good result and shows that the model is working quite well.

ESPAÑOL:

Precisión:

Alta: 0.70 Baja: 0.76 Media: 0.82 La precisión indica la proporción de verdaderos positivos entre todos los predicciones positivas. La categoría "Media" tiene la mejor precisión, lo cual es positivo.

Recall:

Alta: 0.53 Baja: 0.62 Media: 0.90 El recall mide la proporción de verdaderos positivos sobre el total de casos reales positivos. La categoría "Media" tiene un recall muy alto, mientras que "Alta" es más baja, lo que significa que se está clasificando más "correctamente" las cervezas que son realmente "Media", pero se están perdiendo algunas que son "Alta".

F1-Score:

Alta: 0.61 Baja: 0.69 Media: 0.86 El F1-Score es la media armónica de precisión y recall. La categoría "Media" también destaca aquí, mostrando un buen equilibrio entre precisión y recall.

Exactitud (Accuracy):

La precisión general es de 0.80, lo que significa que el 80% de las predicciones son correctas. Este es un buen resultado y muestra que el modelo está funcionando bastante bien.

## preguntas iniciales:

```
df
{"type": "dataframe", "variable_name": "df"}

# Agrupar por el tipo de cerveza (beer_beerid) y calcular el promedio
# de las evaluaciones
mejor_evaluadas = df.groupby('beer_style')
['review_overall'].mean().sort_values(ascending=False).head(10)

print("Tipos de cerveza mejor evaluados:")
print(mejor_evaluadas)

Tipos de cerveza mejor evaluados:
beer_style
American Wild Ale          4.093262
```

```

Gueuze                                4.086287
Quadrupel (Quad)                      4.071630
Lambic - Unblended                     4.048923
American Double / Imperial Stout      4.029820
Russian Imperial Stout                 4.023084
Weizenbock                            4.007969
American Double / Imperial IPA       3.998017
Flanders Red Ale                      3.992722
Rye Beer                               3.981737
Name: review_overall, dtype: float64

# Agrupar por el tipo de cerveza (beer_beerid) y calcular el promedio de las valoraciones de sabor
mejor_sabor = df.groupby('brewery_name')[['review_taste']].mean().sort_values(ascending=False).head(10)

print("Cervezas con mejor sabor valorado:")
print(mejor_sabor)

Cervezas con mejor sabor valorado:
brewery_name
Elizabeth Street Brewery              5.00
Binghams Brewery                     5.00
Ludwig Roth Bierbrauerei GmbH        5.00
Hakone Beer                           5.00
The Cellar Grill                      5.00
Thai Me Up                            5.00
Rascal Creek Brewing Co.             5.00
Sherlock's Home                       4.90
Barum Brewery Limited                4.75
Bad Bear Brewing Company              4.75
Name: review_taste, dtype: float64

mas_reviews = df['brewery_name'].value_counts().head(10)

print("Cervezas con más reviews:")
print(mas_reviews)

Cervezas con más reviews:
brewery_name
Boston Beer Company (Samuel Adams)   39444
Dogfish Head Brewery                  33839
Stone Brewing Co.                     33066
Sierra Nevada Brewing Co.            28751
Bell's Brewery, Inc.                  25191
Rogue Ales                            24083
Founders Brewing Company              20004
Victory Brewing Company               19479
Lagunitas Brewing Company              16837

```

Avery Brewing Company  
Name: count, dtype: int64

16107

## Modelo de regresion lineal

**normaliza los datos para el entrenamiento**

```
!pip install imblearn
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Identify non-numeric columns
non_numeric_cols = df.select_dtypes(exclude=['number']).columns

# Drop or encode non-numeric columns
# Here, we're dropping for simplicity. You might want to encode them instead.
df = df.drop(columns=non_numeric_cols)

# Separar las características (X) de la variable objetivo (y)
X = df.drop('review_overall', axis=1) # Suponiendo que 'review_overall' es la variable objetivo
y = df['review_overall']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Seleccionar las características (X) de la variable objetivo (y)
X = df.drop('review_overall', axis=1)
y = df['review_overall']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes)
Collecting imbalanced-learn (from imblearn)
  Downloading imbalanced_learn-0.12.4-py3-none-any.whl.metadata (8.3 kB)
Requirement already satisfied: numpy>=1.17.3 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.5.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3.5.0)
Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Downloading imbalanced_learn-0.12.4-py3-none-any.whl (258 kB)
258.3/258.3 kB 4.4 MB/s eta
0:00:00
balanced-learn, imblearn
Successfully installed imbalanced-learn-0.12.4 imblearn-0.0
```

El código prepara los datos para ser usados en un modelo. Primero, elimina las columnas que no son números. Luego, separa los datos en dos partes: una con las características que se usarán para predecir, y otra con el resultado que se quiere predecir (review\_overall). Divide los datos en entrenamiento (80%) y prueba (20%), y normaliza los datos para que todas las características estén en la misma escala. Hay algunas partes repetidas que podrían simplificarse

Modelo =

```
# Importar librerías necesarias
import pandas as pd
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline

# Separar las características (X) de la variable objetivo (y)
X = df.drop('review_overall', axis=1)
y = df['review_overall']

# Definir el modelo de regresión Lasso con regularización
```

```

alpha = 0.1 # Parámetro de regularización, ajustable
model = make_pipeline(StandardScaler(), Lasso(alpha=alpha))

# Definir K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Hacer predicciones con cross-validation
y_pred = cross_val_predict(model, X, y, cv=kf)

# Evaluar el modelo usando MSE y R2
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print("Rendimiento cuadrático medio (MSE):", mse)
print("Rendimiento de R^2:", r2)

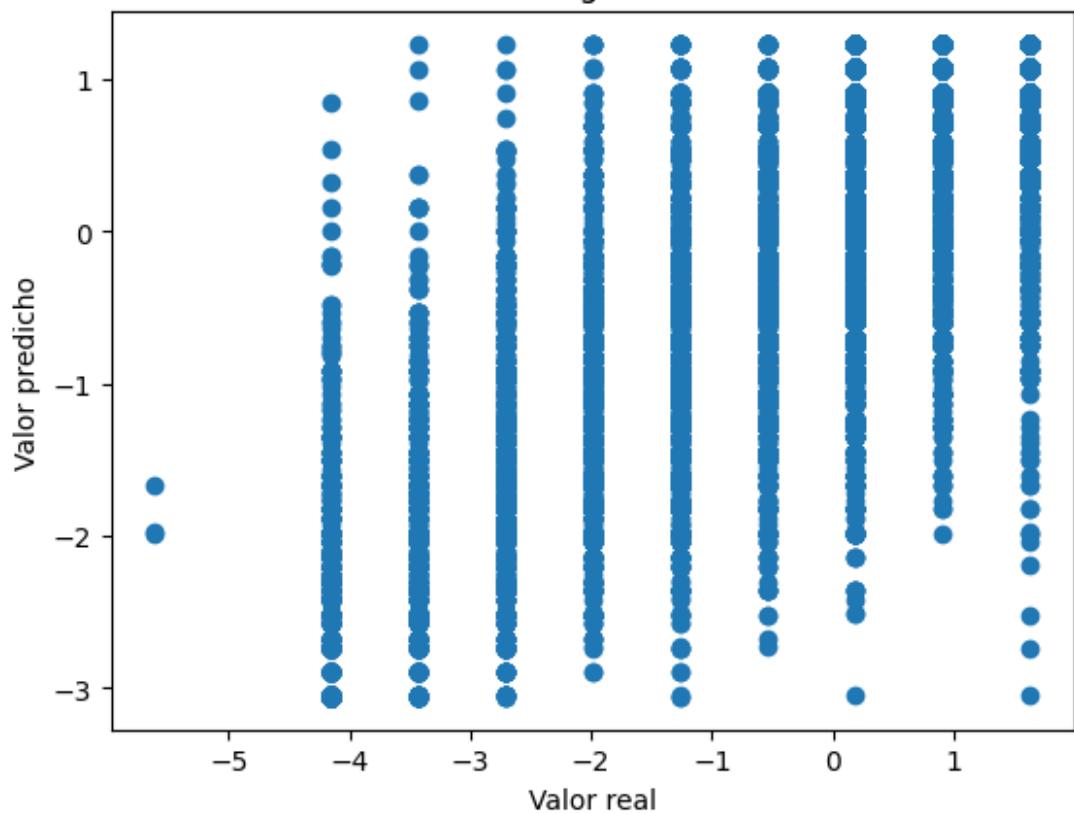
# Graficar el rendimiento del modelo
plt.scatter(y, y_pred)
plt.xlabel('Valor real')
plt.ylabel('Valor predicho')
plt.title('Rendimiento del modelo de regresión Lasso (Cross-Validation)')
plt.show()

plt.plot(y.values, label='Valor real')
plt.plot(y_pred, label='Valor predicho', linestyle='--')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.title('Comparación de valores reales y predichos (Cross-Validation)')
plt.legend()
plt.show()

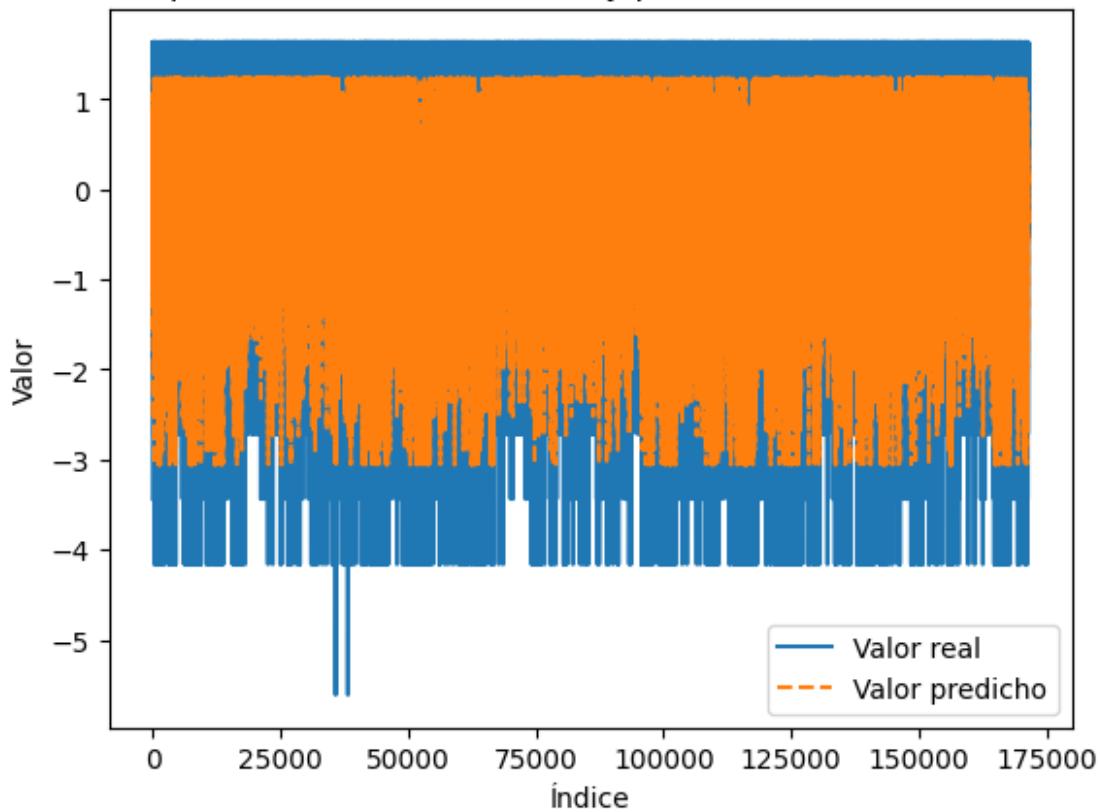
```

Rendimiento cuadrático medio (MSE): 0.3771648356446432  
Rendimiento de R<sup>2</sup>: 0.6228351643553567

### Rendimiento del modelo de regresión Lasso (Cross-Validation)



### Comparación de valores reales y predichos (Cross-Validation)



El código implementa un modelo de regresión Lasso para predecir las calificaciones de cervezas (review\_overall) a partir de otras características (X) y la variable objetivo (y), utilizando un modelo de regresión Lasso que incluye regularización para evitar el overfitting, penalizando coeficientes grandes. Se emplea un pipeline para normalizar los datos y aplicar el modelo de forma eficiente, y se implementa K-fold cross-validation para evaluar el modelo en diferentes subconjuntos de datos.

Los resultados muestran un error cuadrático medio (MSE) de 0.3772, lo que indica un error promedio relativamente bajo en las predicciones, aunque no perfecto. El coeficiente de determinación ( $R^2$ ) es de 0.6228, lo que significa que el modelo explica aproximadamente el 62.28% de la variabilidad en las calificaciones de las cervezas, dejando un 37.72% de variabilidad no explicada, sugiriendo que podrían ser relevantes otras características. En conclusión, el modelo tiene un rendimiento aceptable, pero hay oportunidades para mejorarlo.

## Modelo regresión polinomica

### Modelo 1

```
# Importar librerías necesarias para el modelo
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
```

```

from sklearn.linear_model import Ridge # Regularización para evitar
overfitting
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline

# Seleccionar características para el modelo
X = df[['review_aroma', 'review_palate', 'review_taste']]
y = df['review_overall']

# Crear un pipeline con normalización, características polinómicas y
Ridge (regularización)
degree = 2 # Grado del polinomio
poly_pipeline = make_pipeline(StandardScaler(),
PolynomialFeatures(degree=degree), Ridge(alpha=1.0)) # Ridge
regularización

# Definir K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Hacer predicciones con cross-validation
y_pred_poly = cross_val_predict(poly_pipeline, X, y, cv=kf)

# Evaluar el modelo usando MSE y R²
mse = mean_squared_error(y, y_pred_poly)
r2 = r2_score(y, y_pred_poly)

print("Rendimiento cuadrático medio (MSE):", mse)
print("Rendimiento de R^2:", r2)

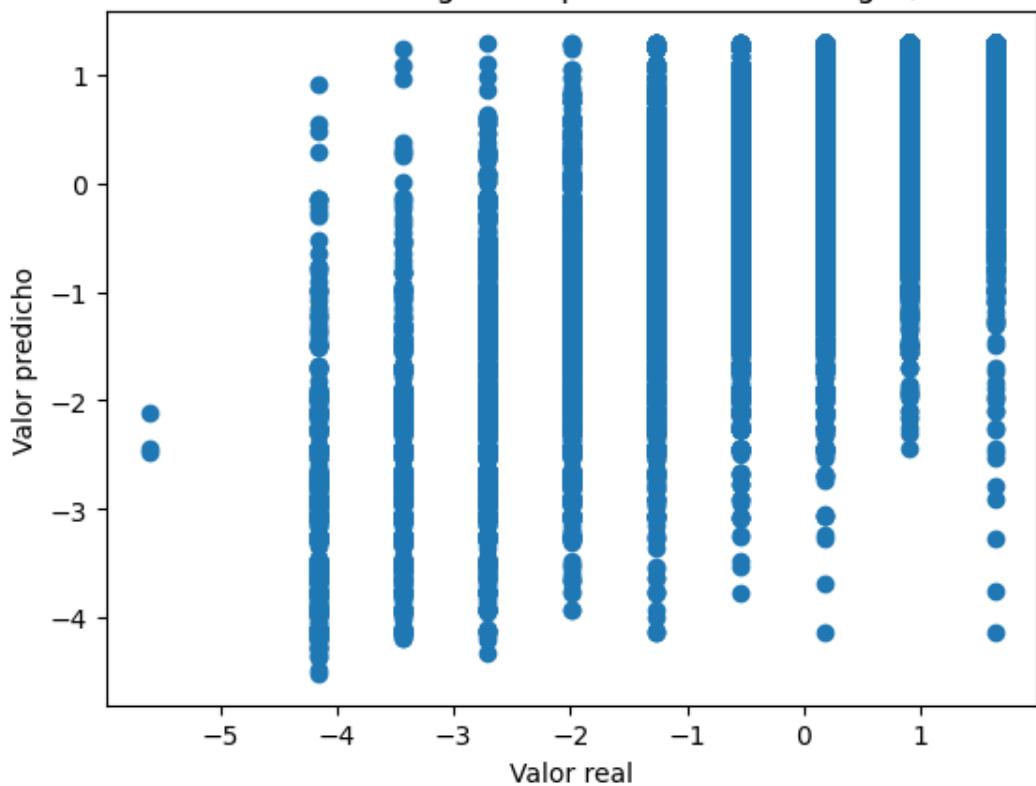
# Graficar el rendimiento del modelo
plt.scatter(y, y_pred_poly)
plt.xlabel('Valor real')
plt.ylabel('Valor predicho')
plt.title('Rendimiento del modelo de regresión polinómica con Ridge
(Cross-Validation)')
plt.show()

# Graficar la comparación de valores reales y predichos
plt.plot(y.values, label='Valor real')
plt.plot(y_pred_poly, label='Valor predicho', linestyle='--')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.title('Comparación de valores reales y predichos (Cross-
Validation)')
plt.legend()
plt.show()

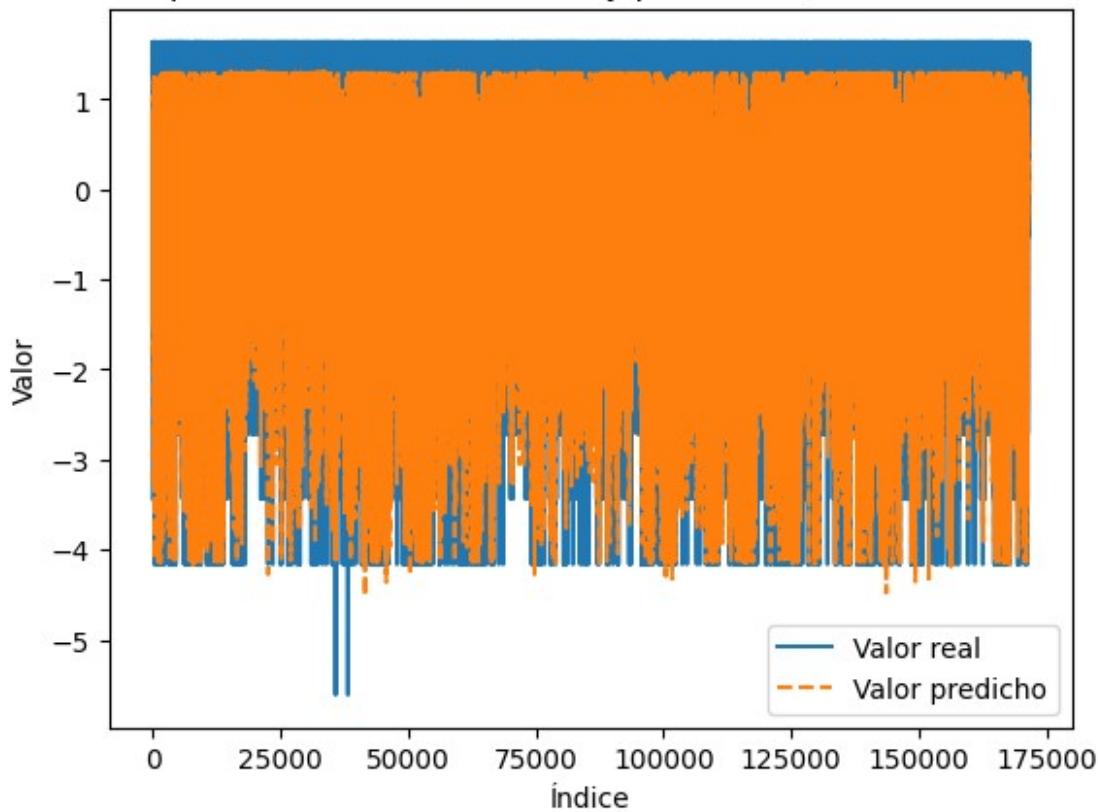
```

Rendimiento cuadrático medio (MSE): 0.35454966847515185  
Rendimiento de R<sup>2</sup>: 0.6454503315248481

Rendimiento del modelo de regresión polinómica con Ridge (Cross-Validation)



### Comparación de valores reales y predichos (Cross-Validation)



El código implementa un modelo de regresión utilizando Ridge para predecir las calificaciones generales de las cervezas (review\_overall) basándose en tres características: review\_aroma, review\_palate y review\_taste. Se crea un pipeline que normaliza los datos, genera características polinómicas de grado 2 y aplica la regresión Ridge, que ayuda a evitar el overfitting. Se utiliza K-fold cross-validation con 5 divisiones para evaluar el modelo en diferentes subconjuntos de datos.

Los resultados indican un rendimiento cuadrático medio (MSE) de 0.3545, lo que sugiere un error promedio bajo en las predicciones, y un coeficiente de determinación ( $R^2$ ) de 0.6455, lo que significa que el modelo explica aproximadamente el 64.55% de la variabilidad en las calificaciones de las cervezas. En resumen, el modelo tiene un buen ajuste, aunque hay margen de mejora al considerar más características o ajustar los hiperparámetros.

### Modelo 2

```
# Importar librerías necesarias
import pandas as pd
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
```

```

# Separar las características (X) de la variable objetivo (y)
X = df[['beer_abv', 'review_appearance']]
y = df['review_overall']

# Crear un pipeline con normalización, características polinómicas y
regresión lineal
degree = 2 # Grado del polinomio
poly_pipeline = make_pipeline(StandardScaler(),
PolynomialFeatures(degree=degree), LinearRegression())

# Definir K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Hacer predicciones con cross-validation
y_pred_poly = cross_val_predict(poly_pipeline, X, y, cv=kf)

# Evaluar el modelo usando MSE y R2
mse = mean_squared_error(y, y_pred_poly)
r2 = r2_score(y, y_pred_poly)

print("Rendimiento cuadrático medio (MSE):", mse)
print("Rendimiento de R^2:", r2)

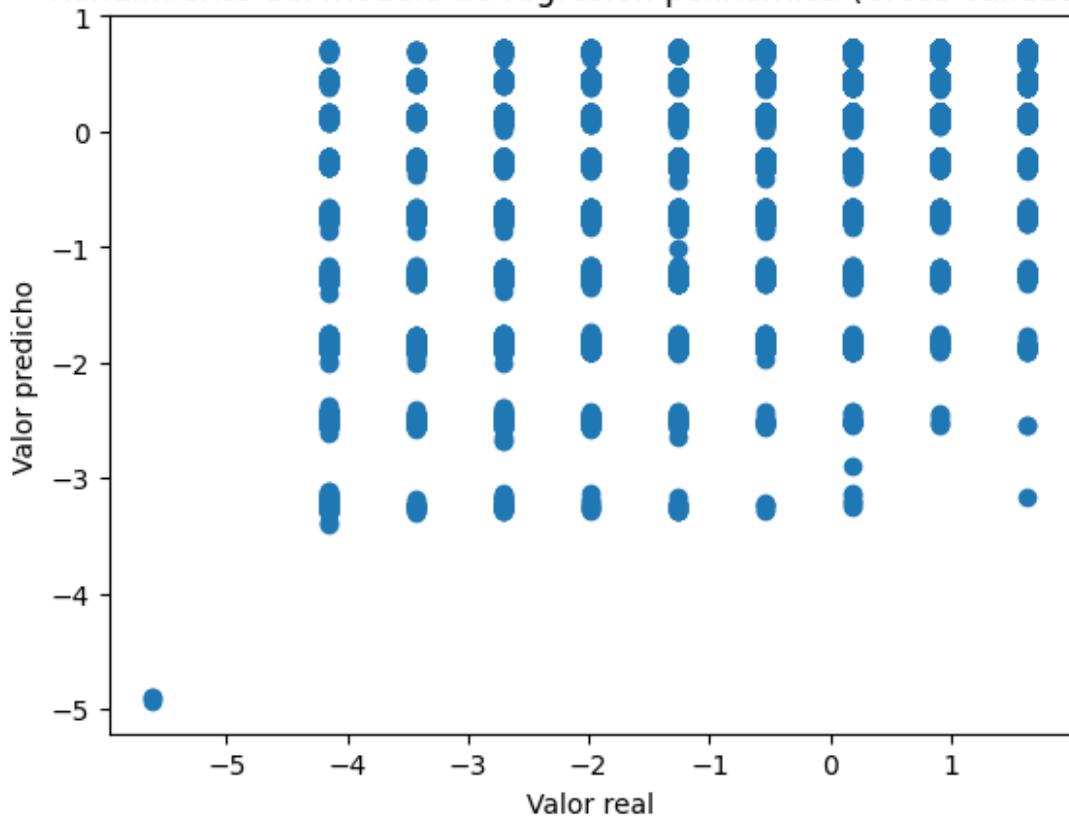
# Graficar el rendimiento del modelo
plt.scatter(y, y_pred_poly)
plt.xlabel('Valor real')
plt.ylabel('Valor predicho')
plt.title('Rendimiento del modelo de regresión polinómica (Cross-Validation)')
plt.show()

# Graficar la comparación de valores reales y predichos
plt.plot(y.values, label='Valor real')
plt.plot(y_pred_poly, label='Valor predicho', linestyle='--')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.title('Comparación de valores reales y predichos (Cross-Validation)')
plt.legend(loc="center")
plt.show()

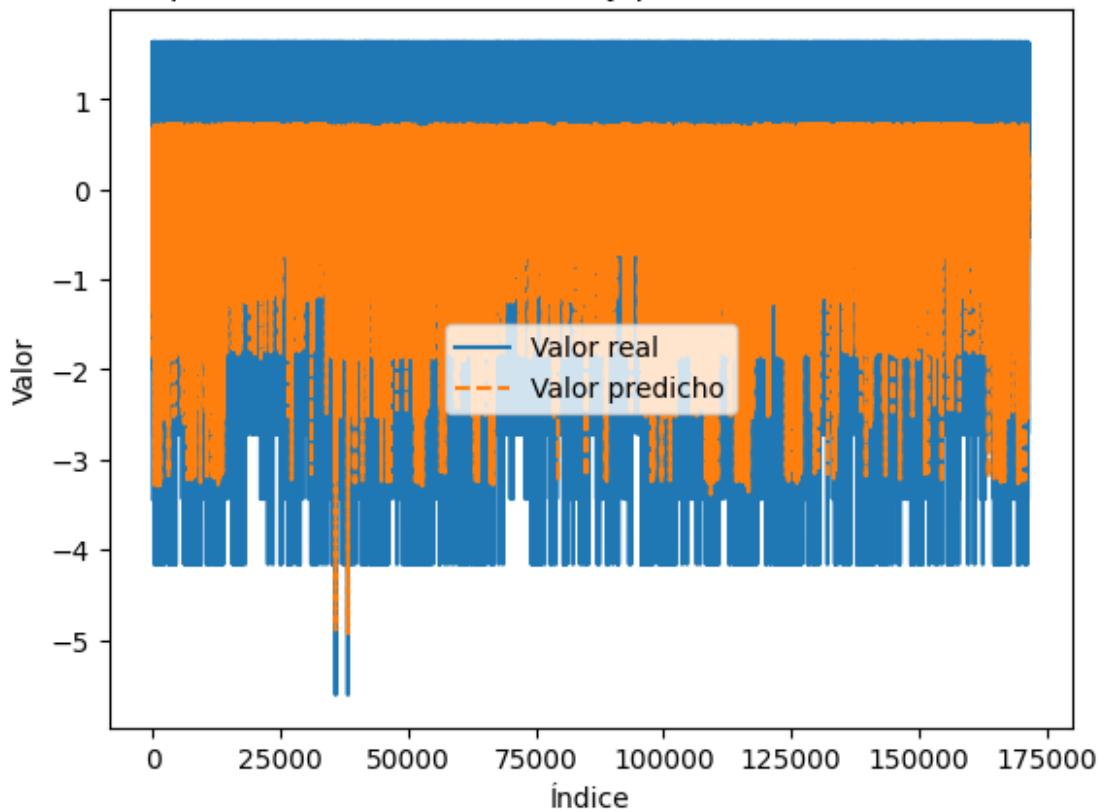
Rendimiento cuadrático medio (MSE): 0.7626536559180874
Rendimiento de R^2: 0.23734634408191246

```

### Rendimiento del modelo de regresión polinómica (Cross-Validation)



## Comparación de valores reales y predichos (Cross-Validation)



El código implementa un modelo de regresión polinómica para predecir las calificaciones generales de las cervezas (review\_overall) utilizando el contenido de alcohol por volumen (beer\_abv) y la apariencia de la cerveza (review\_appearance). Se crea un pipeline que normaliza los datos, genera características polinómicas de grado 2 y aplica regresión lineal. Se utiliza validación cruzada K-fold con 5 divisiones para evaluar el rendimiento del modelo.

Los resultados muestran un rendimiento cuadrático medio (MSE) de 0.7627, indicando un error promedio alto en las predicciones, y un coeficiente de determinación ( $R^2$ ) de 0.2373, lo que significa que el modelo solo explica el 23.73% de la variabilidad en las calificaciones. Estos valores sugieren que el modelo no está ajustando bien a los datos, lo que implica que se necesitan más características o un enfoque diferente para mejorar el rendimiento.

### Modelo 3

```
# Importar librerías necesarias
import pandas as pd
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Ridge # Usamos Ridge para
regularización
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
```

```

# Separar las características (X) de la variable objetivo (y)
X = df[['review_taste','beer_abv']]
y = df['review_appearance']

# Crear un pipeline con normalización, características polinómicas y
# Ridge (regularización)
degree = 2 # Grado del polinomio
poly_pipeline = make_pipeline(StandardScaler(),
PolynomialFeatures(degree=degree), Ridge(alpha=1.0)) # Ridge con
alpha=1.0

# Definir K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Hacer predicciones con cross-validation
y_pred_poly = cross_val_predict(poly_pipeline, X, y, cv=kf)

# Evaluar el modelo usando MSE y R²
mse = mean_squared_error(y, y_pred_poly)
r2 = r2_score(y, y_pred_poly)

print("Rendimiento cuadrático medio (MSE):", mse)
print("Rendimiento de R^2:", r2)

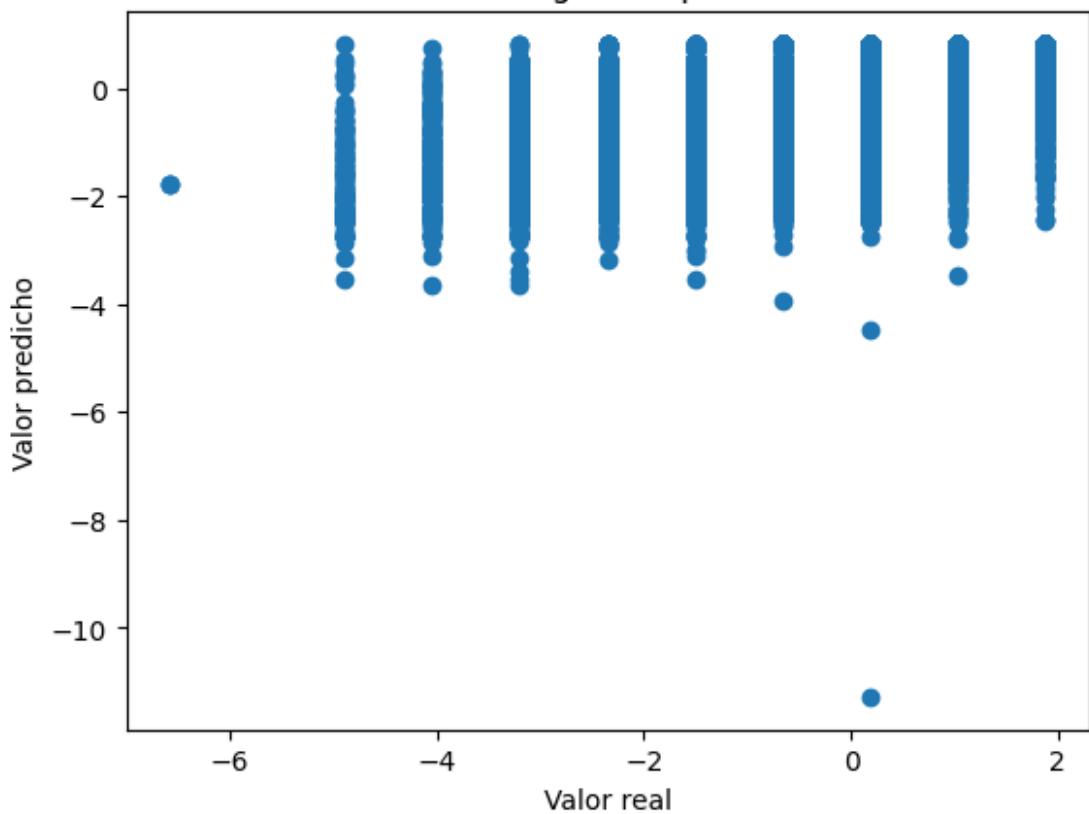
# Graficar el rendimiento del modelo
plt.scatter(y, y_pred_poly)
plt.xlabel('Valor real')
plt.ylabel('Valor predicho')
plt.title('Rendimiento del modelo de regresión polinómica (Cross-Validation)')
plt.show()

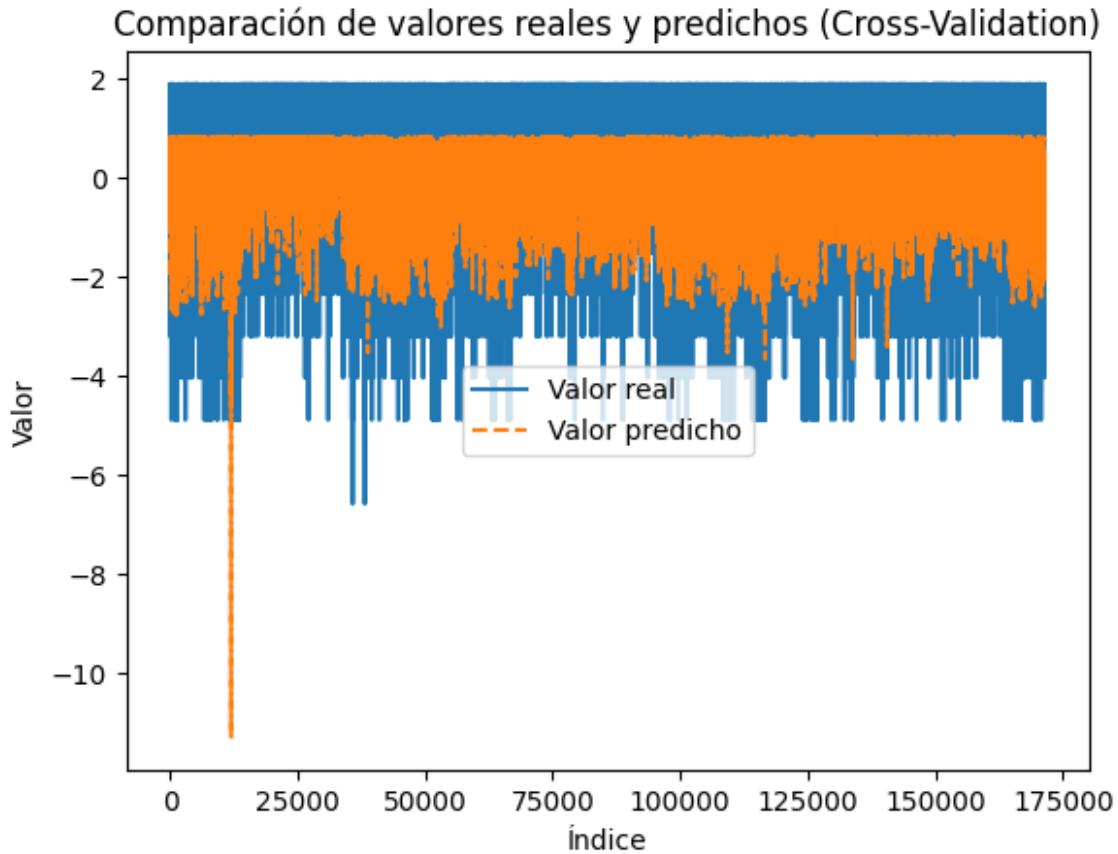
# Graficar la comparación de valores reales y predichos
plt.plot(y.values, label='Valor real')
plt.plot(y_pred_poly, label='Valor predicho', linestyle='--')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.title('Comparación de valores reales y predichos (Cross-Validation)')
plt.legend(loc="center")
plt.show()

Rendimiento cuadrático medio (MSE): 0.7054757130436135
Rendimiento de R^2: 0.29452428695638655

```

### Rendimiento del modelo de regresión polinómica (Cross-Validation)





El modelo de regresión polinómica con regularización Ridge para predecir la apariencia de cervezas (review\_appearance) utilizando dos características: el sabor de la cerveza (review\_taste) y su contenido de alcohol (beer\_abv). Primero, los datos se normalizan y se generan características polinómicas de grado 2 mediante un pipeline que combina estas transformaciones con el modelo Ridge, el cual ayuda a prevenir el sobreajuste (overfitting).

Se aplica K-fold cross-validation con 5 divisiones para obtener predicciones más robustas y evitar que los resultados se vean afectados por una sola división del conjunto de datos. Luego, se evalúa el rendimiento del modelo utilizando el error cuadrático medio (MSE) y el coeficiente de determinación ( $R^2$ ). En este caso, el MSE es 0.7055, lo que indica un error moderado en las predicciones, mientras que el  $R^2$  es 0.2945, lo que sugiere que el modelo solo explica aproximadamente el 29.45% de la variabilidad en la apariencia de la cerveza. Esto indica que hay margen para mejorar el modelo, posiblemente considerando más variables o ajustando el enfoque. Las gráficas muestran cómo se comparan los valores reales y los predichos, reflejando la efectividad del modelo en su predicción.

## Arbol de decisión 3

```
# Definir la calidad
def categorize_quality(value):
    if value <= 2:
        return 'Baja'
```

```

        elif 2 < value <= 4:
            return 'Media'
        else:
            return 'Alta'

# Crear una columna de calidad basada en review_appearance
df['beer_quality'] = df['review_appearance'].apply(categorize_quality)

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# Seleccionar las características que quieras usar (puedes añadir más columnas si lo prefieres)
X = df[['review_overall', 'review_aroma', 'review_palate',
'review_taste', 'beer_abv']].fillna(0)
y = df['beer_quality']

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Crear el modelo de árbol de decisión
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)

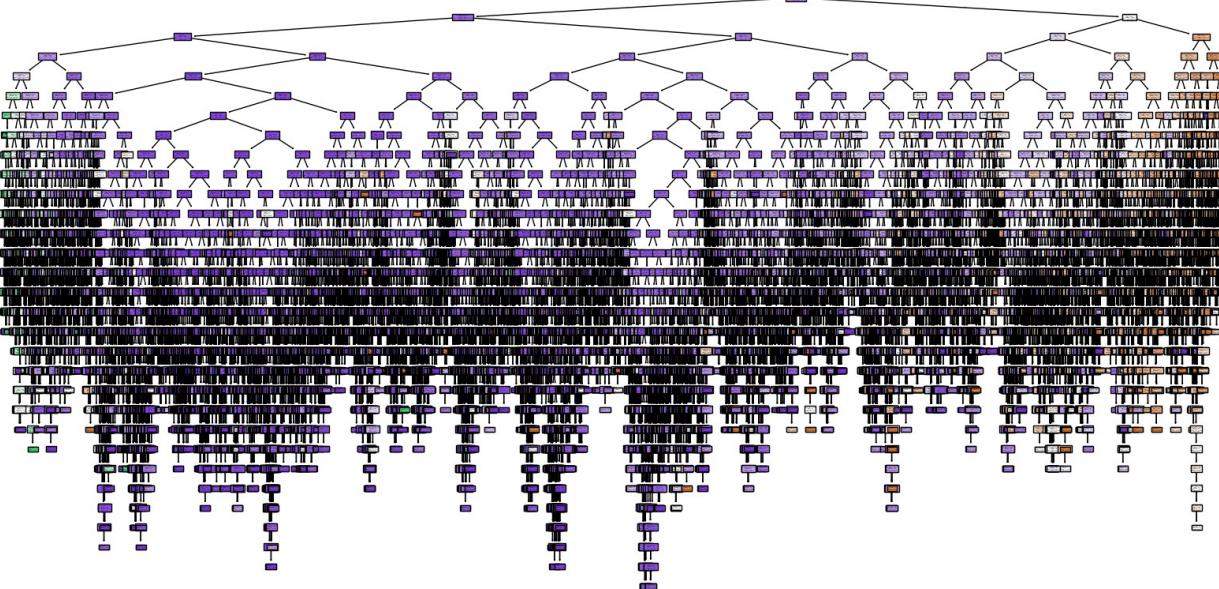
# Predecir y evaluar
y_pred = tree_clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

Accuracy: 0.7421044150821785
      precision    recall  f1-score   support
          Alta       0.52      0.37      0.44     8992
          Baja       0.29      0.27      0.28      652
        Media       0.80      0.87      0.83    27592
          accuracy           0.74     37236
      macro avg       0.54      0.51      0.52     37236
    weighted avg       0.72      0.74      0.73     37236

from sklearn import tree
import matplotlib.pyplot as plt

# Visualizar el árbol
plt.figure(figsize=(20,10))
tree.plot_tree(tree_clf, filled=True, feature_names=X.columns,
class_names=['Baja', 'Media', 'Alta'])
plt.show()

```



```

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Seleccionar las características
X = df[['review_overall', 'review_aroma', 'review_palate',
'review_taste', 'beer_abv']].fillna(0)
y = df['beer_quality']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Crear y ajustar el modelo de árbol de decisión con limitaciones para
evitar sobreajuste
tree_clf = DecisionTreeClassifier(
    random_state=42,
    max_depth=10, # Limitar la profundidad del árbol
    min_samples_split=10, # Número mínimo de muestras necesarias para
dividir un nodo
    min_samples_leaf=5, # Número mínimo de muestras necesarias en una
hoja
    class_weight='balanced' # Ajustar peso para balancear las clases
)

tree_clf.fit(X_train, y_train)

# Predicciones y evaluación
y_pred = tree_clf.predict(X_test)
print("Accuracy ajustado:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```

| Accuracy ajustado: 0.5818025566655924 |           |        |          |         |
|---------------------------------------|-----------|--------|----------|---------|
|                                       | precision | recall | f1-score | support |
| Alta                                  | 0.41      | 0.73   | 0.52     | 8992    |
| Baja                                  | 0.12      | 0.76   | 0.20     | 652     |
| Media                                 | 0.86      | 0.53   | 0.66     | 27592   |
| accuracy                              |           |        | 0.58     | 37236   |
| macro avg                             | 0.46      | 0.67   | 0.46     | 37236   |
| weighted avg                          | 0.74      | 0.58   | 0.62     | 37236   |

Better performance in minority classes

The "Low" class now has a recall of 76% (previously it was only 27%). This means that the model is doing a better job of identifying low-quality beers, although accuracy is still low (12%). This is typical in data sets. The "High" class also has a much better recall now (73% vs. 37% previously), meaning the model identifies many more high-quality beers.

Performance in the "Middle" class

The model is less accurate for the "Medium" class now (accuracy dropped from 82% to 86%), but it is still quite strong. The f1-score for the middle class (0.66) is still acceptable, but lowered compared to the previous setting.

español:

Mejor rendimiento en clases minoritarias

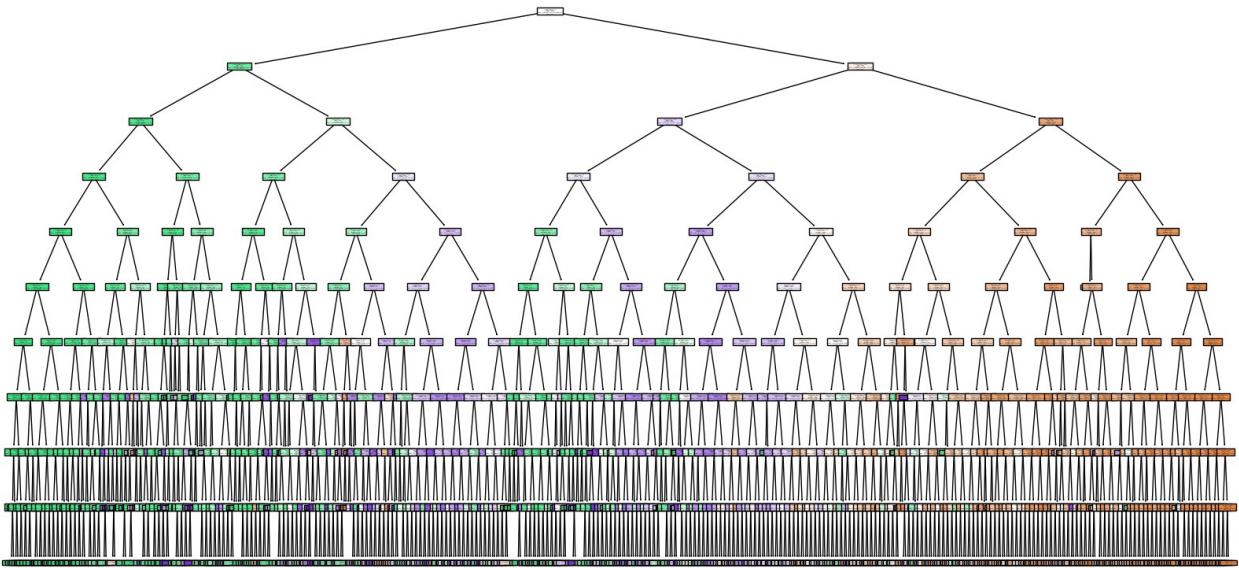
La clase "Baja" tiene ahora un recall del 76% (antes era solo del 27%). Esto significa que el modelo está haciendo un mejor trabajo identificando las cervezas de baja calidad, aunque la precisión sigue siendo baja (12%). Esto es típico en conjuntos de datos. La clase "Alta" también tiene un recall mucho mejor ahora (73% frente al 37% anterior), lo que significa que el modelo identifica muchas más cervezas de alta calidad.

Rendimiento en la clase "Media"

El modelo es menos preciso para la clase "Media" ahora (precision bajó del 82% al 86%), pero sigue siendo bastante fuerte. El f1-score para la clase media (0.66) aún es aceptable, pero bajó en comparación con el ajuste anterior.

```
from sklearn import tree
import matplotlib.pyplot as plt

# Visualizar el árbol ajustado
plt.figure(figsize=(20,10))
tree.plot_tree(tree_clf, filled=True, feature_names=X.columns,
class_names=['Baja', 'Media', 'Alta'])
plt.show()
```



```

import pandas as pd

# Cargar el archivo CSV
file_path =
'C:/Users/massr/Desktop/reviewsCerveceriaKedro/cerveceriar/data/01_raw
/beer_reviews.csv'
df = pd.read_csv(file_path, sep=",")

# Verificar las primeras filas para asegurarte de que se cargó
correctamente
df.head()

      brewery_id      brewery_name review_time review_overall \
0        10325    Vecchio Birraio  1234817823           1.5
1        10325    Vecchio Birraio  1235915097           3.0
2        10325    Vecchio Birraio  1235916604           3.0
3        10325    Vecchio Birraio  1234725145           3.0
4       1075 Caldera Brewing Company  1293735206           4.0

      review_aroma  review_appearance review_profilename \
0            2.0                  2.5          stcules
1            2.5                  3.0          stcules
2            2.5                  3.0          stcules
3            3.0                  3.5          stcules
4            4.5                  4.0  johnmichaelsen

      beer_style  review_palate  review_taste \
0      Hefeweizen         1.5          1.5
1  English Strong Ale         3.0          3.0
2  Foreign / Export Stout         3.0          3.0
3     German Pilsener         2.5          3.0

```

|   | 4 American Double / Imperial IPA | 4.0 | 4.5   |
|---|----------------------------------|-----|-------|
| 0 | Sausa Weizen                     | 5.0 | 47986 |
| 1 | Red Moon                         | 6.2 | 48213 |
| 2 | Black Horse Black Beer           | 6.5 | 48215 |
| 3 | Sausa Pils                       | 5.0 | 47969 |
| 4 | Cauldron DIPA                    | 7.7 | 64883 |

```

import pandas as pd # Importar pandas

# Cargar el archivo CSV (si no lo has hecho aún)
file_path =
'C:/Users/massr/Desktop/reviewsCerveceriaKedro/cerveceriar/data/01_raw
/beer_reviews.csv'
df = pd.read_csv(file_path, sep=",")

# Definir las categorías de calidad basadas en 'review_appearance'
df['beer_quality'] = pd.cut(df['review_appearance'],
                             bins=[0, 2, 3.5, 5], # Cambiar los
umbrales si es necesario
                             labels=['Baja', 'Media', 'Alta'])

# Verificar las primeras filas para asegurarte de que la columna se
creó correctamente
df[['review_appearance', 'beer_quality']].head()

review_appearance  beer_quality
0                2.5        Media
1                3.0        Media
2                3.0        Media
3                3.5        Media
4                4.0        Alta

import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# Cargar los datos
file_path =
'C:/Users/massr/Desktop/reviewsCerveceriaKedro/cerveceriar/data/01_raw
/beer_reviews.csv'
df = pd.read_csv(file_path, sep=",")

# Crear la columna 'beer_quality' basada en 'review_appearance' (que
será categórica)
df['beer_quality'] = pd.cut(df['review_appearance'],

```

```

        bins=[0, 2, 3.5, 5],
        labels=['Baja', 'Media', 'Alta'])

# Asegurarnos de que 'beer_quality' es categórica
df['beer_quality'] = df['beer_quality'].astype('category')

# Seleccionar las características (X) y la variable objetivo (y)
X = df[['review_overall', 'review_aroma', 'review_palate',
'review_taste', 'beer_abv']].fillna(0)
y = df['beer_quality']

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Normalizar los datos (X_train y X_test)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Verificar si hay valores nulos en y_train y y_test y rellenarlos si es necesario
y_train = y_train.fillna(y_train.mode()[0])
y_test = y_test.fillna(y_test.mode()[0])

# Aplicar SMOTE para balancear las clases en el conjunto de entrenamiento
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled,
y_train)

# Crear y ajustar el modelo de árbol de decisión con el conjunto balanceado
tree_clf = DecisionTreeClassifier(
    random_state=42,
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=5
)

# Entrenar el modelo
tree_clf.fit(X_train_smote, y_train_smote)

# Predicciones y evaluación
y_pred = tree_clf.predict(X_test_scaled)

# Asegurarnos de que y_pred y y_test sean tipo cadena
y_pred = pd.Series(y_pred).astype(str)
y_test = y_test.astype(str)

```

```

# Calcular la precisión y el reporte
print("Accuracy con SMOTE y datos normalizados:",
accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

Accuracy con SMOTE y datos normalizados: 0.674603191277036
      precision    recall   f1-score   support
      Alta       0.84     0.71     0.77    308410
      Baja       0.23     0.77     0.36    10514
      Media      0.52     0.59     0.56   157061
      accuracy          0.67    475985
      macro avg       0.53     0.69     0.56    475985
      weighted avg    0.72     0.67     0.69    475985

```

Precision:

For the High class, the accuracy is 0.84, which means that when the model predicts "High", it is correct 84% of the time. For the Low class, the accuracy is only 0.23, indicating that there is a high false positive rate (predicting "Low" when it is not actually "Low"). For the Medium class, the accuracy is 0.52, which also indicates moderate performance.

Recall:

For the High class, the recall is 0.69, indicating that the model identifies 69% of all true instances of "High". For the Low class, the recall is 0.77, which means that the model correctly identifies 77% of the true instances of "Low", despite its low precision. For the Medium class, the recall is 0.61.

F1-Score: This is a weighted average of precision and recall, which provides a better measure of model performance in imbalanced classes.

The F1-score for "High" is 0.76, which is pretty good. For "Low", it is 0.36, suggesting that the model is having difficulty correctly identifying this class. For "Medium", it is 0.56, which also indicates that there is room for improvement.

conclusion:

Class imbalance: The precision and recall for the "Low" class are significantly worse than the other classes. This could indicate that your model is having trouble learning from this less represented class, even though using SMOTE helps balance the dataset.

Español:

Precision:

Para la clase Alta, la precisión es 0.84, lo que significa que cuando el modelo predice "Alta", el 84% de las veces es correcto. Para la clase Baja, la precisión es solo 0.23, lo que indica que hay una alta tasa de falsos positivos (predice "Baja" cuando no es realmente "Baja"). Para la clase Media, la precisión es 0.52, lo que también indica un rendimiento moderado.

Recall:

Para la clase Alta, el recall es 0.69, lo que indica que el modelo identifica el 69% de todas las instancias verdaderas de "Alta". Para la clase Baja, el recall es 0.77, lo que significa que el modelo identifica correctamente el 77% de las instancias verdaderas de "Baja", a pesar de su baja precisión. Para la clase Media, el recall es 0.61.

1-Score: Este es un promedio ponderado de precisión y recall, que proporciona una mejor medida del rendimiento del modelo en clases desequilibradas.

La F1-score para "Alta" es 0.76, lo que es bastante bueno. Para "Baja", es 0.36, lo que sugiere que el modelo tiene dificultades para identificar correctamente esta clase. Para "Media", es 0.56, lo que también indica que hay margen de mejora.

conclusion:

Desbalance de clases: La precisión y recall para la clase "Baja" son significativamente peores que las otras clases. Esto podría indicar que tu modelo tiene problemas para aprender de esta clase menos representada, a pesar de que el uso de SMOTE ayuda a balancear el dataset.

## Conclusión

En esta parte del proyecto, empezamos a analizar de mejor manera los datos ya seleccionados anteriormente. Mediante diferentes métodos de análisis, descubrimos distintas informaciones que nos serán útiles para cuando ocupemos los datos de este CSV en las ventas reales con la cervecería Kross. El informe realizado analiza datos de cervezas caseras estadounidenses para ayudar a la cervecería Kross a identificar las mejores opciones para el mercado chileno. Las principales variables evaluadas fueron sabor, aroma, paladar y apariencia, que influyen directamente en las calificaciones de las cervezas. Estas características permiten clasificar las cervezas en categorías de alta, media y baja calidad, lo que ayudará a Kross a diferenciar sus productos en el mercado. Los modelos realizados en esta segunda unidad tienen una aplicación directa en el mercado, ayudando a las empresas a comprender mejor los patrones de comportamiento de sus productos y consumidores.

Árboles de Clasificación: Utilizando los árboles de clasificación, aprendimos cómo segmentar productos en diferentes categorías de calidad, como en el caso del análisis de cervezas (Alta, Media, Baja). Esta segmentación permite a la empresa identificar qué productos cumplen con los estándares de alta calidad y cuáles podrían requerir mejoras. A nivel de mercado, esto apoya una estrategia de diferenciación, donde los productos se pueden orientar a diferentes segmentos de consumidores con precios o enfoques de marketing diferenciados.

árbol de decisión fue el más destacado por su capacidad para segmentar cervezas en diferentes categorías de calidad. Este modelo mostró una precisión del 84% al clasificar cervezas de alta calidad y mejoró significativamente el reconocimiento de las cervezas de calidad baja, algo que otros modelos no lograron con la misma efectividad. Sus métricas de precisión, recall y F1-Score permitieron identificar patrones clave en los datos, haciendo de este modelo una herramienta útil para Kross al definir qué productos lanzar y cómo mejorarlos.

Otro modelo relevante fue la regresión Lasso, que logró explicar el 62% de la variabilidad en las calificaciones de las cervezas. Este modelo evitó el sobreajuste y mantuvo un buen equilibrio

entre simplicidad y precisión, siendo ideal para predecir las calificaciones generales de las cervezas. Su bajo error cuadrático medio (MSE) mostró que sus predicciones estaban cerca de los valores reales, lo que lo convierte en una herramienta confiable para comprender qué características afectan más la calidad percibida de las cervezas.

En resumen, estos modelos proporcionaron una base sólida para que Kross identifique las cervezas más prometedoras y aplique estrategias efectivas de marketing y desarrollo de productos en el mercado chileno.

## Modelos no supervisados

```
!pip install openpyxl
import pandas as pd

#
# Leer el archivo csv

file_path =
'C:/Users/massr/Desktop/reviewsCerveceriaKedro/cerveceriar/data/01_raw
/filtered_review_overall.csv'
df2 = pd.read_csv(file_path, sep=",")

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: openpyxl in c:\users\massr\desktop\
reviewservicerakedro\venv\lib\site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in c:\users\massr\desktop\
reviewservicerakedro\venv\lib\site-packages (from openpyxl) (2.0.0)

df2 = df2.dropna()

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

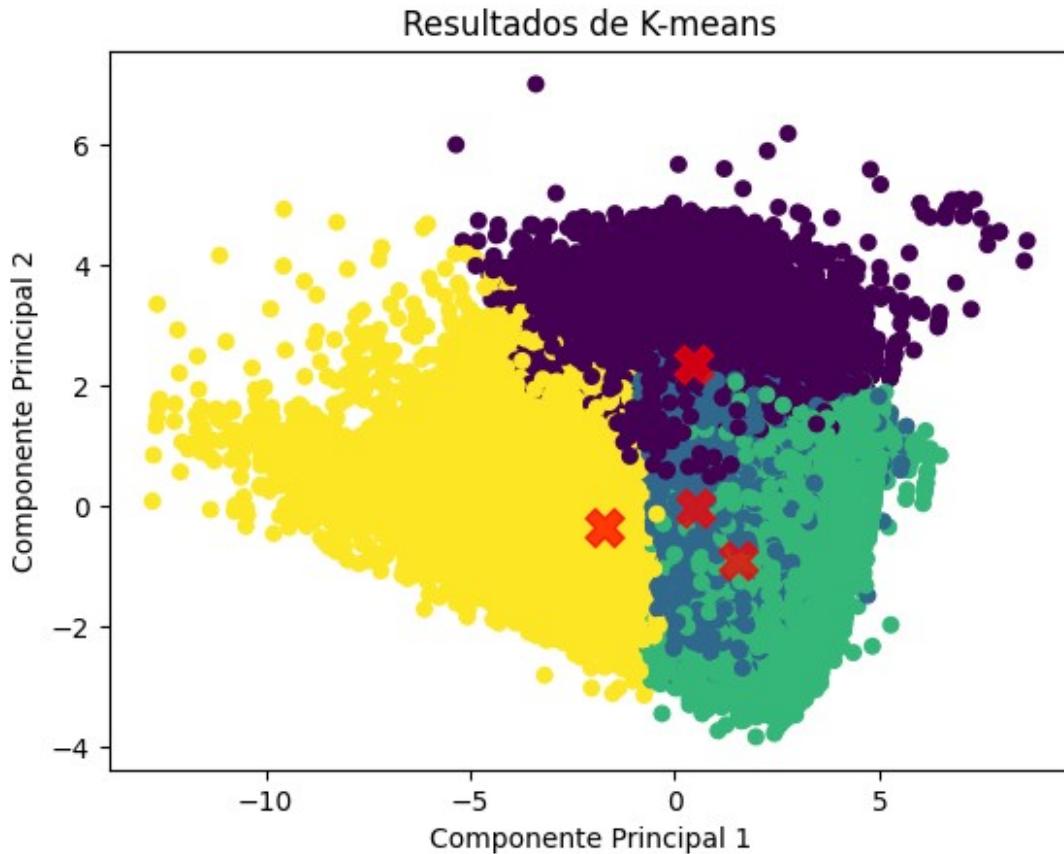
# Elimina columnas no numéricas o irrelevantes (ajusta según tu
dataset)
df2 = df2.select_dtypes(include=[np.number])

# Normaliza los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df2)

# Eliminar filas con valores NaN
df2_cleaned = df2.dropna()

# Normalizar los datos después de limpiar
```

```
X_scaled =  
scaler.fit_transform(df2_cleaned.select_dtypes(include=[np.number]))  
  
# Define el número de clusters  
kmeans = KMeans(n_clusters=4, random_state=0)  
kmeans.fit(X_scaled)  
  
# Predecir los clusters  
df2['Cluster'] = kmeans.labels_  
  
from sklearn.decomposition import PCA  
  
# Reducir dimensionalidad  
pca = PCA(n_components=2)  
X_pca = pca.fit_transform(X_scaled)  
  
# Visualizar los resultados  
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df2['Cluster'], s=30,  
cmap='viridis')  
centers = pca.transform(kmeans.cluster_centers_)  
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75,  
marker='X')  
plt.title('Resultados de K-means')  
plt.xlabel('Componente Principal 1')  
plt.ylabel('Componente Principal 2')  
plt.show()
```



K-Means can help identify hidden patterns and segment the market, which is key to designing more effective strategies. Practical Applications

It allows grouping customers into segments based on characteristics such as:

Purchase frequency.

Product reviews.

Style or flavor preferences.

K-Means can uncover patterns among product features (flavor, aroma, appearance) and their overall evaluations. This can reveal which combinations are the most popular.

**ESPAÑOL:** K-Means puede ayudar a identificar patrones ocultos y segmentar en el mercado, lo cual es clave para diseñar estrategias más efectivas.

Aplicaciones prácticas

permite agrupar a los clientes en segmentos basados en características como Frecuencia de compra. Reseñas sobre productos. Preferencias de estilo o sabor

K-Means puede encontrar patrones entre las características de los productos (sabor, aroma, apariencia) y sus evaluaciones generales. Esto puede indicar qué combinaciones son más populares.

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Extraer características del DataFrame (excepto la última columna)
X = df2.iloc[:, :-1].values # Esto es más eficiente que trabajar con
DataFrame directamente

# Escalar los datos (normalización)
X_scaled = StandardScaler().fit_transform(X)

# Aplicar DBSCAN para detectar clústeres
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Reducir la dimensionalidad para la visualización usando PCA
X_pca = PCA(n_components=2).fit_transform(X_scaled)

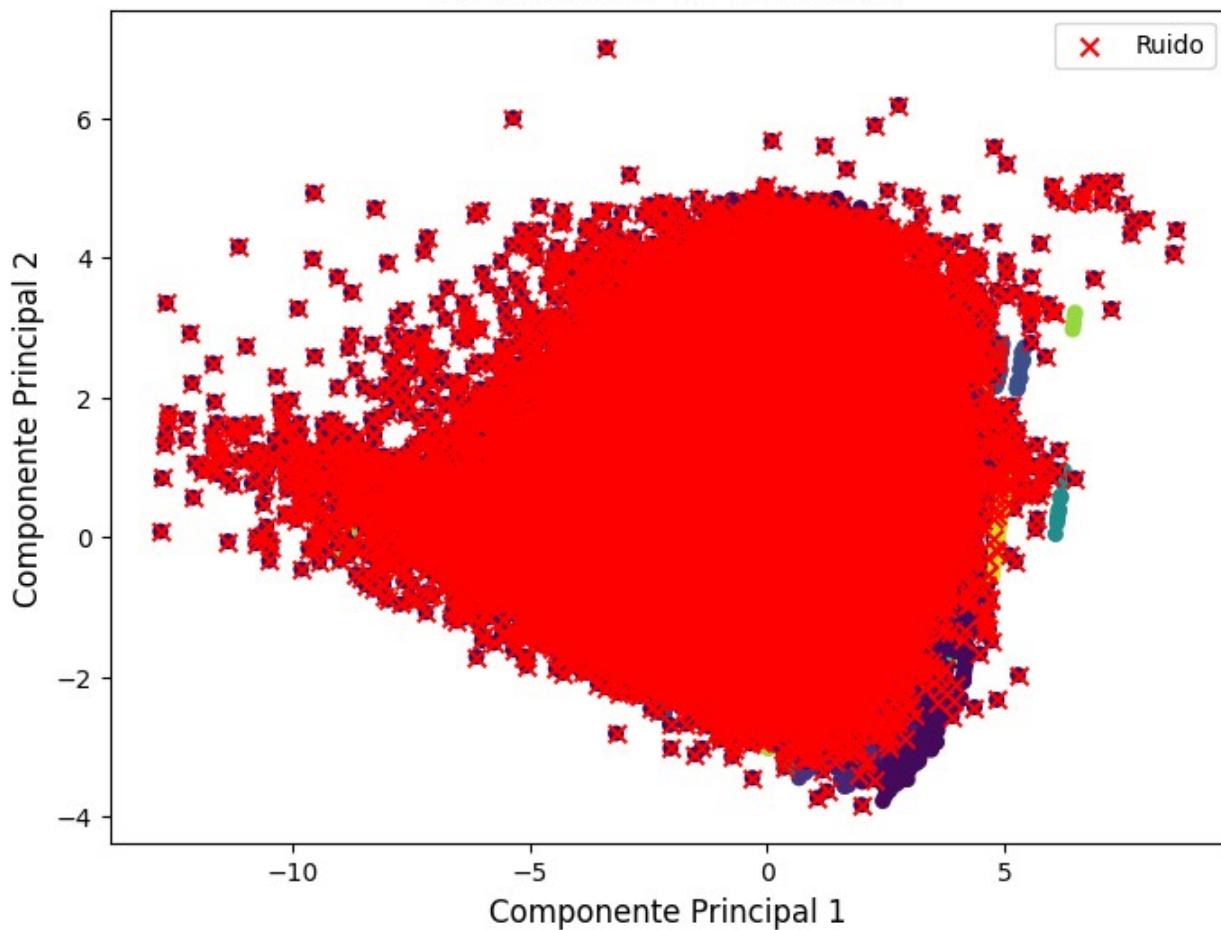
# Crear la visualización en un solo paso
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, s=30,
cmap='viridis')

# Marcar los puntos de ruido (etiquetados como -1 por DBSCAN)
plt.scatter(X_pca[dbscan_labels == -1, 0], X_pca[dbscan_labels == -1,
1],
            c='red', s=50, label='Ruido', marker='x')

# Añadir título y etiquetas
plt.title('Resultados de DBSCAN', fontsize=16)
plt.xlabel('Componente Principal 1', fontsize=12)
plt.ylabel('Componente Principal 2', fontsize=12)
plt.legend()

# Mostrar la gráfica
plt.show()
```

## Resultados de DBSCAN



DBSCAN is a clustering algorithm that identifies clusters based on data density. Unlike K-Means, it does not require specifying the number of clusters in advance and can effectively handle noise (outliers). Identification of Market Niches and Outliers

Detects niche customer groups: Customers with unique behaviors or preferences that do not fit into traditional clusters.

Identifies outliers: Exceptional products or customers with uncommon behaviors, such as infrequent but high-value purchases.

### Practical Applications

**Product Segmentation:** Groups products based on characteristics like bitterness, aroma, alcohol content, etc., to identify specific categories or unique combinations.

**Customer Behavior Analysis:** Detects customers with unusual purchase patterns, such as those who make occasional but high-impact purchases or have very specific product preferences.

**Market Opportunity Detection:** Highlights areas where current products do not fully meet demand, helping to identify opportunities to innovate with new styles or flavors.

### Strategic Value

DBSCAN provides a robust approach to uncover hidden patterns in complex data, supporting decision-making in product development, marketing, and customer engagement strategies.

ESPAÑOL: DBSCAN es un algoritmo de agrupamiento que identifica clústeres en función de la densidad de los datos. A diferencia de K-Means, no requiere especificar el número de clústeres de antemano y es capaz de manejar ruido (valores atípicos) de manera efectiva.

#### Identificación de Nichos de Mercado y Valores Atípicos

Detecta grupos de clientes nicho: Clientes con comportamientos o preferencias únicas que no encajan en los clústeres tradicionales.

Identifica valores atípicos: Productos o clientes excepcionales con comportamientos fuera de lo común, como compras infrecuentes pero de alto valor

#### Aplicaciones Prácticas

Segmentación de Productos: Agrupa productos según características como amargor, aroma, contenido de alcohol, etc., para identificar categorías específicas o combinaciones únicas.

Análisis de Comportamiento de Clientes: Detecta clientes con patrones de compra inusuales, como aquellos que realizan compras ocasionales pero de alto impacto, o que tienen preferencias de productos muy específicas.

Detección de Oportunidades de Mercado: Señala áreas donde los productos actuales no satisfacen completamente la demanda, ayudando a identificar oportunidades para innovar con nuevos estilos o sabores.

## K-distance

```
# Instalar librerías necesarias
!pip install scikit-learn matplotlib

# Importar las librerías necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors

# Aquí cargamos un dataset de ejemplo de sklearn para ilustrar
from sklearn.datasets import load_iris
data = load_iris()
X = data.data # Usaremos las características

# Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Calcular la distancia k más cercana (k-distance) usando
NearestNeighbors
```

```
k = 4 # Usualmente se elige un k pequeño, como 4 o 5
neighbors = NearestNeighbors(n_neighbors=k)
neighbors.fit(X_scaled)

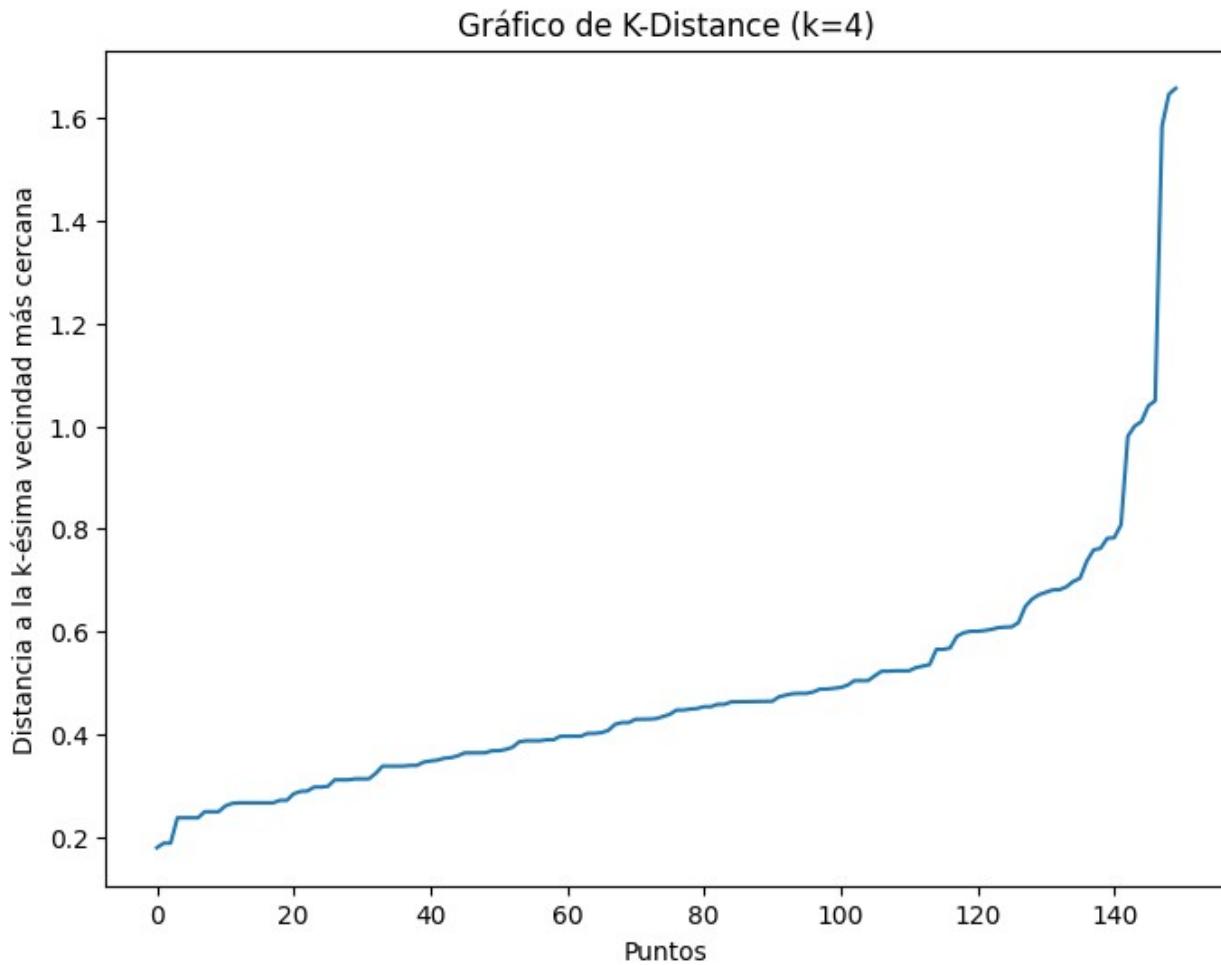
# Obtener las distancias k más cercanas para cada punto
distances, indices = neighbors.kneighbors(X_scaled)

# Ordenar las distancias k más cercanas de menor a mayor para cada punto
distances = np.sort(distances[:, -1], axis=0) # Tomamos solo la distancia k-ésima

# Graficar el gráfico de k-distance
plt.figure(figsize=(8, 6))
plt.plot(distances)
plt.title(f"Gráfico de K-Distance (k={k})")
plt.xlabel("Puntos")
plt.ylabel("Distancia a la k-ésima vecindad más cercana")
plt.show()

Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.19.5 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
```

```
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```



K-Distance (K-nearest distance) is a useful tool in clustering analysis, especially in algorithms like DBSCAN and K-Means, as it allows you to observe the density of points in the feature space and is essential for identifying the appropriate number of clusters or understanding the underlying structure of the data.

Detection of the number of clusters in DBSCAN: The K-Distance graph is particularly useful in the DBSCAN algorithm to choose the optimal epsilon ( $\epsilon$ ) value, which is the radius of the clusters. The graph shows the distances to the nearest points (usually the k-th nearest neighbor) in ascending order.

Data Density Analysis: The K-Distance graph helps observe how densely the data points are grouped. If the distance to the k nearest neighbors is small and consistent, the data is likely forming a dense cluster. If the distance increases dramatically, it may indicate that the data is sparse or there are outliers.

**Improvement of Market Segmentation:** With the K-Distance graph, you can identify areas with high density of points, helping to detect patterns of customer behavior or product characteristics that are more concentrated in certain areas. For example:

**Customers in dense segments:** You can identify segments of customers who make frequent purchases and have similar behaviors (which allows you to create more specific marketing strategies).

**Popular or unpopular products:** Identify products with many reviews or high popularity, or those that may be underserved in the market.

**ESPAÑOL:** K-Distance (distancia K más cercana) es una herramienta útil en el análisis de clustering, especialmente en algoritmos como DBSCAN y K-Means, ya que permite observar la densidad de los puntos en el espacio de características y es esencial para identificar el número adecuado de clústeres o para comprender la estructura subyacente de los datos.

**Detección de la cantidad de clústeres en DBSCAN:** El gráfico de K-Distance es especialmente útil en el algoritmo DBSCAN para elegir el valor óptimo de epsilon ( $\epsilon$ ), que es el radio de los clústeres. El gráfico muestra las distancias a los puntos más cercanos (usualmente el k-ésimo vecino más cercano) en orden ascendente.

**Análisis de la Densidad de los Datos:** El gráfico de K-Distance permite observar cuán densamente agrupados están los datos. Si la distancia a los k vecinos más cercanos es pequeña y consistente, los datos probablemente forman un clúster denso. Si la distancia aumenta de manera drástica, es posible que los datos estén dispersos o que existan valores atípicos.

**Mejora de la Segmentación del Mercado:** Con el gráfico de K-Distance, se puede identificar zonas con alta densidad de puntos, lo cual ayuda a detectar patrones de comportamiento de clientes o características de productos que están más concentrados en ciertas áreas. Por ejemplo:

**Ciencia en segmentos densos:** Puede encontrar segmentos de clientes que realizan compras frecuentes y con comportamientos similares (lo cual te permite crear estrategias de marketing más específicas).

**Productos populares o no populares:** Identificar productos con muchas reseñas o popularidad, o aquellos que pueden estar siendo desatendidos por el mercado

## dbSCAN

## PCA Y UMAP

```
# Primero instalamos las dependencias necesarias
!pip install umap-learn
!pip install matplotlib scikit-learn
```

```
X = df2.iloc[:, :-1].values # Ajusta según las columnas de
```

```

características
y = df2.iloc[:, -1].values # Ajusta según la columna de etiquetas
(si tienes)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import umap

from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

# Escalamos los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# -----
# PCA - Reducción de dimensionalidad
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Gráfico de PCA
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', s=30)
plt.title('Reducción de Dimensionalidad con PCA')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.colorbar(label='Clase')
plt.show()

# -----
# UMAP - Reducción de dimensionalidad
umap_model = umap.UMAP(n_components=2)
X_umap = umap_model.fit_transform(X_scaled)

# Gráfico de UMAP
plt.figure(figsize=(8, 6))
plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y, cmap='viridis', s=30)
plt.title('Reducción de Dimensionalidad con UMAP')
plt.xlabel('Componente UMAP 1')
plt.ylabel('Componente UMAP 2')
plt.colorbar(label='Clase')
plt.show()

```

```
Requirement already satisfied: umap-learn in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (0.5.7)
Requirement already satisfied: numpy>=1.17 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from umap-learn) (2.0.2)
Requirement already satisfied: scipy>=1.3.1 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from umap-learn) (1.14.1)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from umap-learn) (1.5.1)
Requirement already satisfied: numba>=0.51.2 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from umap-learn) (0.60.0)
Requirement already satisfied: pynndescent>=0.5 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from umap-learn) (0.5.13)
Requirement already satisfied: tqdm in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from umap-learn) (4.66.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from numba>=0.51.2->umap-learn) (0.43.0)
Requirement already satisfied: joblib>=0.11 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from pynndescent>=0.5->umap-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from scikit-learn>=0.22->umap-learn) (3.5.0)
Requirement already satisfied: colorama in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from tqdm->umap-learn) (0.4.6)
```

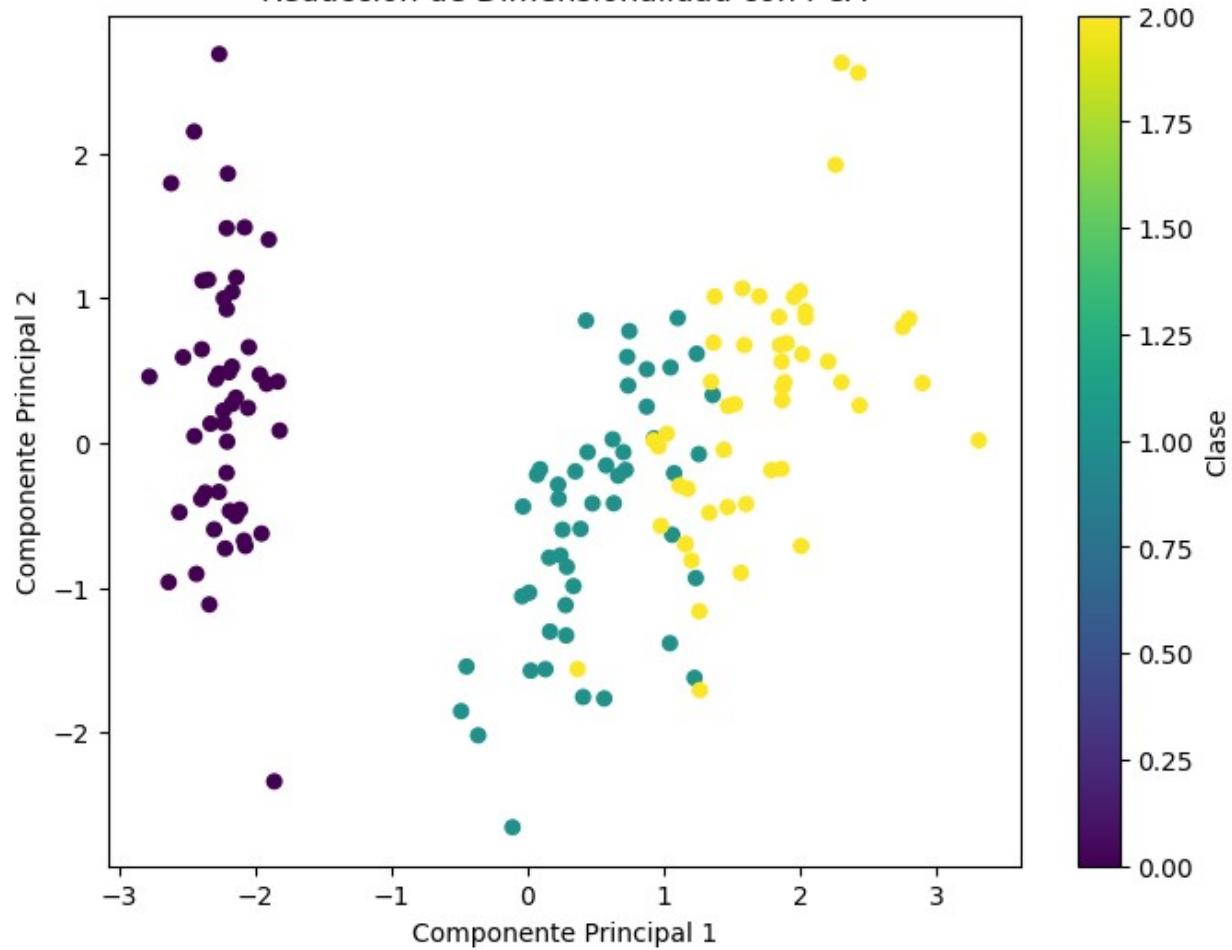
```
[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

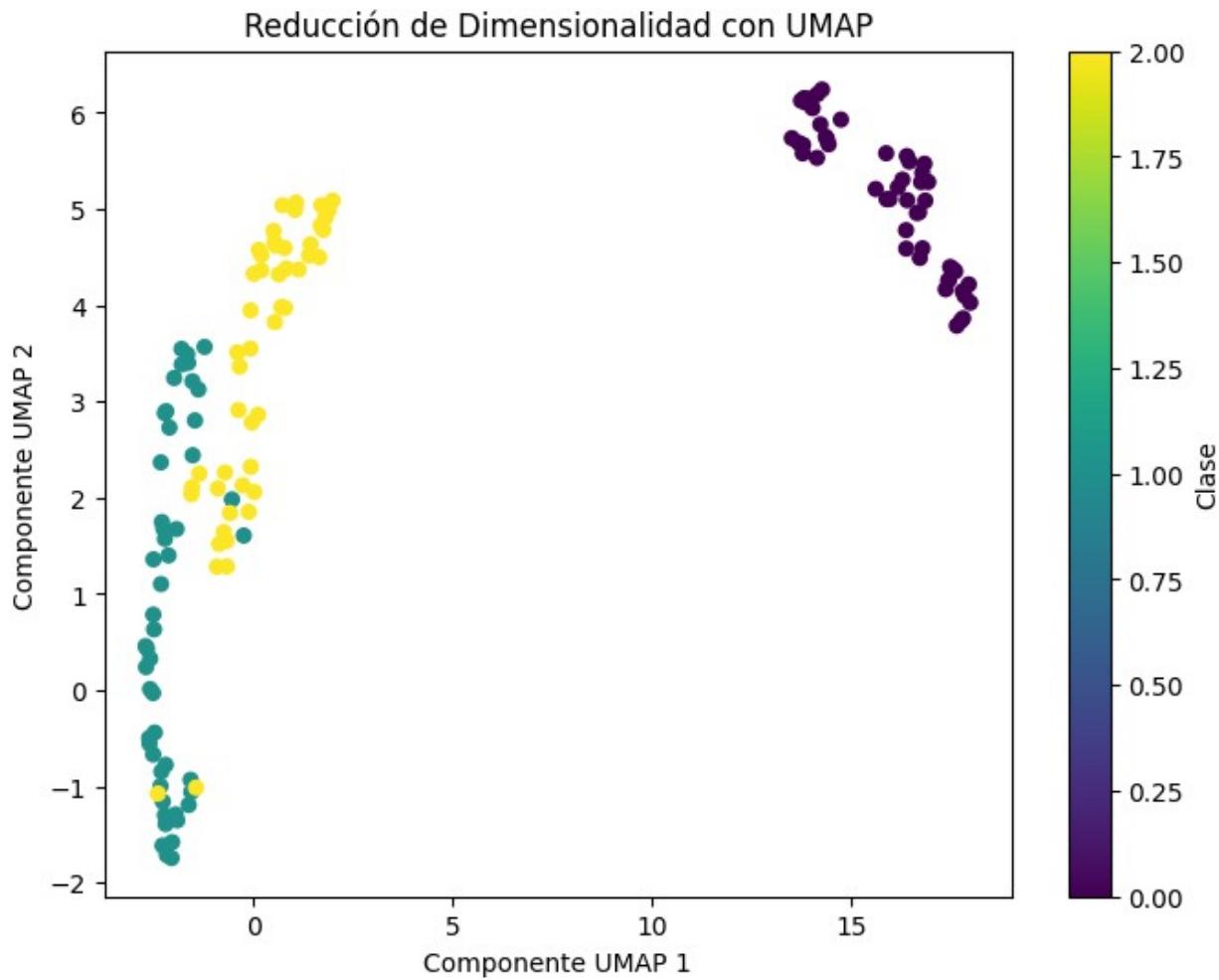
```
Requirement already satisfied: matplotlib in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (3.9.2)
Requirement already satisfied: scikit-learn in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (1.5.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (4.53.1)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy>=1.23 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: scipy>=1.6.0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\users\massr\desktop\reviewscerveceriakedro\venv\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

### Reducción de Dimensionalidad con PCA





**PCA:**

It is a linear approach that seeks to find the direction of maximum variance in the data. While it is very fast and efficient, it sometimes fails to capture nonlinear relationships between variables.

The graph generated by PCA shows how the points are separated according to the first two principal components, but it may not capture the structure of the data well if there are complex nonlinear relationships.

**UMAP:**

It is a more recent technique that uses manifold approximations to preserve both local and global relationships. It is more flexible than PCA for handling nonlinear structures and is capable of capturing more complex patterns.

UMAP plots often provide a more accurate representation of the data structure when there are nonlinear relationships between features.

**ESPAÑOL:**

**PCA:**

Es un enfoque lineal que busca encontrar la dirección de mayor varianza en los datos. Aunque es muy rápido y eficiente, a veces no captura las relaciones no lineales entre las variables.

El gráfico generado por PCA muestra cómo se separan los puntos según las dos primeras componentes principales, pero puede no capturar bien la estructura de los datos si hay relaciones no lineales complejas.

UMAP:

Es una técnica más reciente que utiliza aproximaciones de variedades para preservar tanto las relaciones locales como globales. Es más flexible que PCA para manejar estructuras no lineales y es capaz de capturar patrones más complejos.

Los gráficos de UMAP suelen ofrecer una representación más precisa de la estructura de los datos cuando hay relaciones no lineales entre las características.

## Cluster jerárquico

```
# Instalación (si fuera necesario)
!pip install scikit-learn scipy matplotlib

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (1.13.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.19.5 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in
```

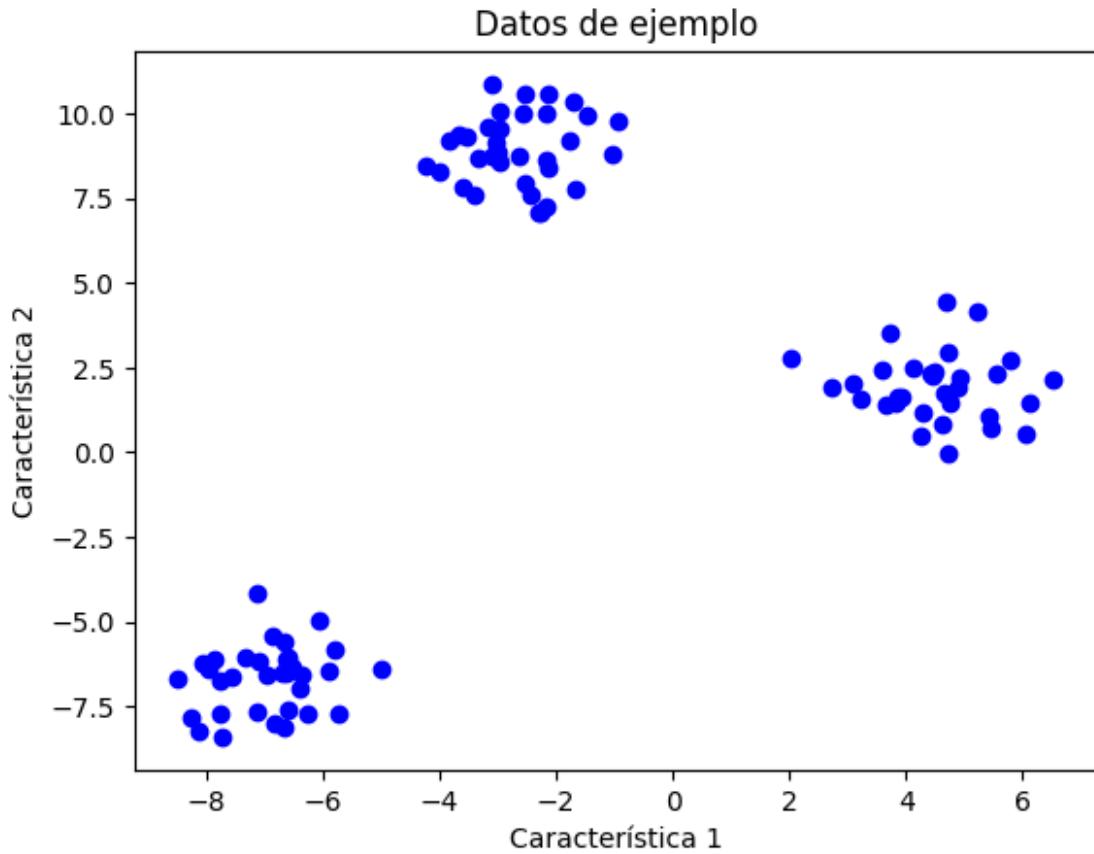
```
/usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib) (1.16.0)
```

Se cargan los datos necesarios para realizar el análisis de Cluster jerárquico. Este proceso permite organizar y agrupar los datos de manera efectiva, basándose en sus similitudes y características comunes.

The necessary data is loaded in order to perform the Hierarchical Clustering analysis. This process allows us to organize and group the data effectively, based on their similarities and common characteristics.

```
# Generamos datos de ejemplo
from sklearn.datasets import make_blobs

# Visualizamos los datos
plt.scatter(X[:, 0], X[:, 1], c='blue')
plt.title("Datos de ejemplo")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.show()
```

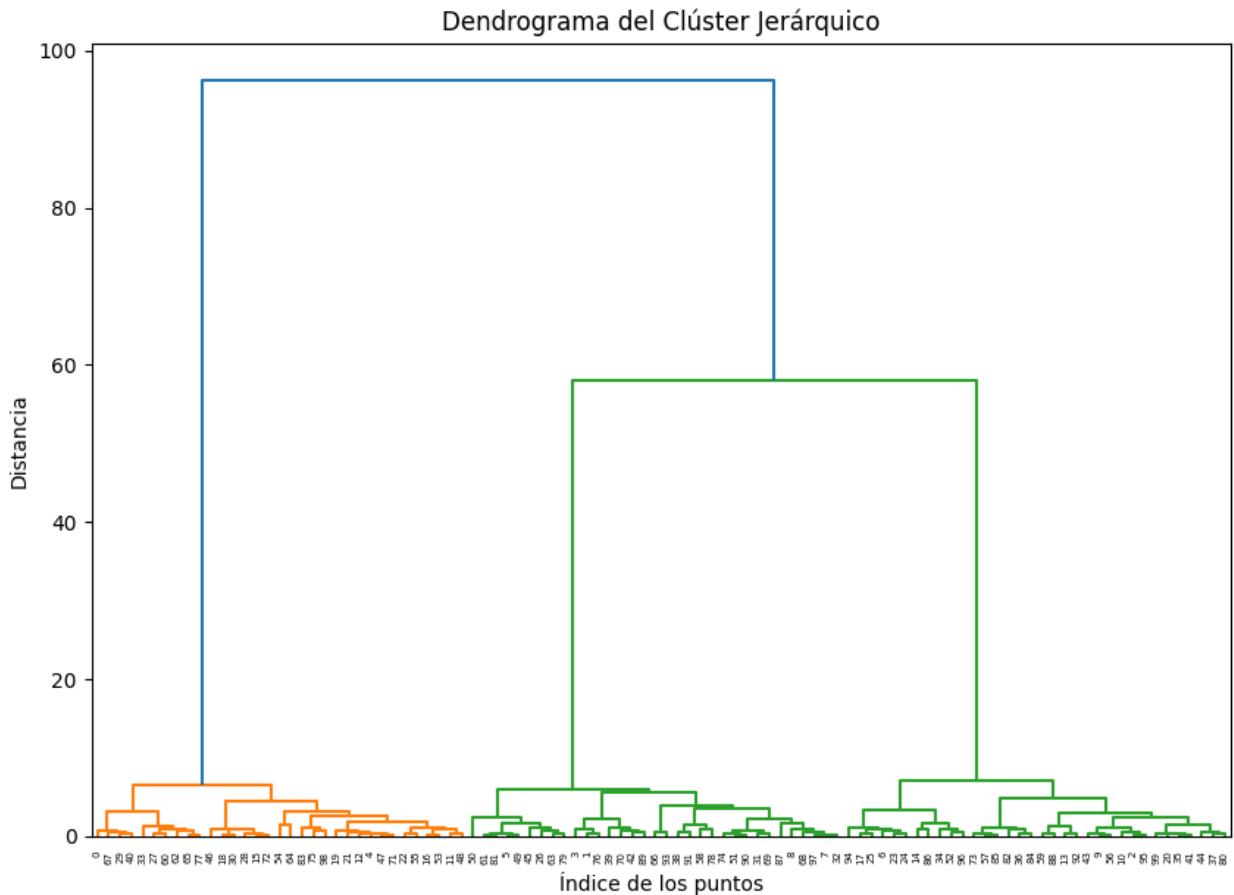


El archivo CSV de la cervecería contiene todos los datos sin ningún tipo de organización previa. A continuación, mostramos la cantidad total de registros que tenemos en el archivo, los cuales, como se puede observar, están completamente desordenados y no presentan una estructura clara.

The CSV file from the brewery contains all the data without any prior organization. Below, we show the total number of records in the file, which, as can be seen, are completely unordered and do not have a clear structure.

```
# Generamos el linkage matrix
Z = linkage(X, method='ward') # 'ward' es un método para calcular distancias

# Visualizamos el dendrograma
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title("Dendrograma del Clúster Jerárquico")
plt.xlabel("Índice de los puntos")
plt.ylabel("Distancia")
plt.show()
```



Para proceder con el análisis, tomamos los mismos datos del CSV y los representamos gráficamente en un dendrograma. Un dendrograma es una herramienta visual que nos permite observar cómo se agrupan los diferentes elementos en función de sus similitudes. Este gráfico nos mostrará las relaciones entre los datos, destacando los grupos que se forman y la cantidad de agrupaciones posibles. El objetivo aquí es dividir los datos en grupos o clusters de manera más efectiva y lógica, buscando una organización que tenga sentido desde el punto de vista del análisis.

To proceed with the analysis, we take the same data from the CSV and represent it graphically in a dendrogram. A dendrogram is a visual tool that helps us observe how the different elements are grouped based on their similarities. This graph will show the relationships between the data, highlighting the clusters that are formed and the number of possible groupings. The goal here is to divide the data into groups or clusters in a more effective and logical way, seeking an organization that makes sense from an analytical point of view.

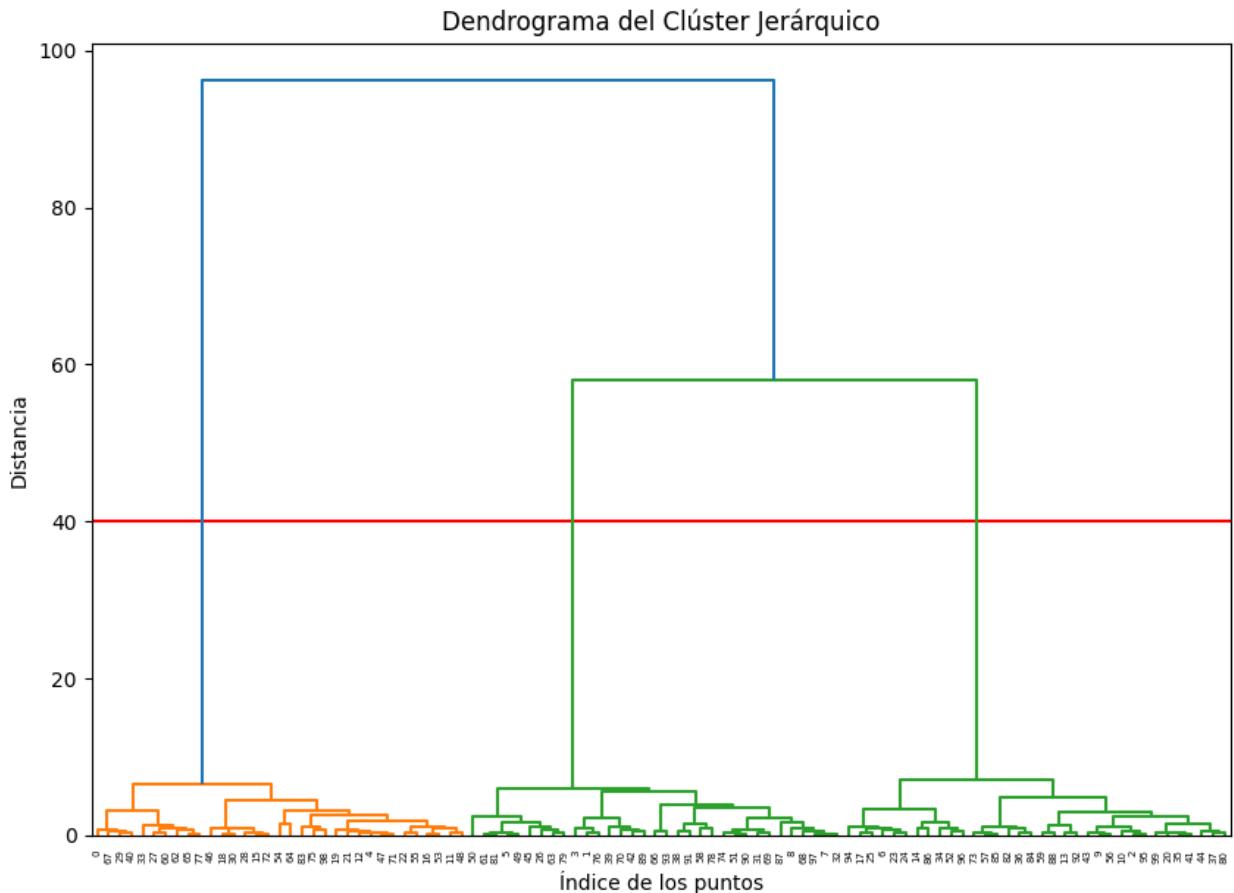
```
# Generamos el linkage matrix
Z = linkage(X, method='ward') # 'ward' es un método para calcular distancias

# Visualizamos el dendrograma
plt.figure(figsize=(10, 7))
plt.axhline(40, c='r')
```

```

dendrogram(Z)
plt.title("Dendrograma del Clúster Jerárquico")
plt.xlabel("Índice de los puntos")
plt.ylabel("Distancia")
plt.show()

```



Para mejorar la segmentación de los datos, se ajusta el umbral de corte del dendrograma en el punto 40. Este punto se elige porque, en este lugar del gráfico, los datos ya no se solapan entre sí de manera significativa, lo que facilita una separación más clara y manejable de los clusters. Al cortar el dendrograma en este punto, logramos una división que nos permitirá trabajar con grupos de datos más coherentes y fáciles de analizar.

To improve the data segmentation, we adjust the cut-off threshold of the dendrogram at the 40 point. This point is chosen because, at this place on the graph, the data no longer overlap significantly, which facilitates a clearer and more manageable separation of the clusters. By cutting the dendrogram at this point, we achieve a division that will allow us to work with more coherent and easier-to-analyze data groups.

```

from scipy.cluster.hierarchy import fcluster
# Definir un umbral para cortar el dendrograma (por ejemplo, a una
# distancia de 10)

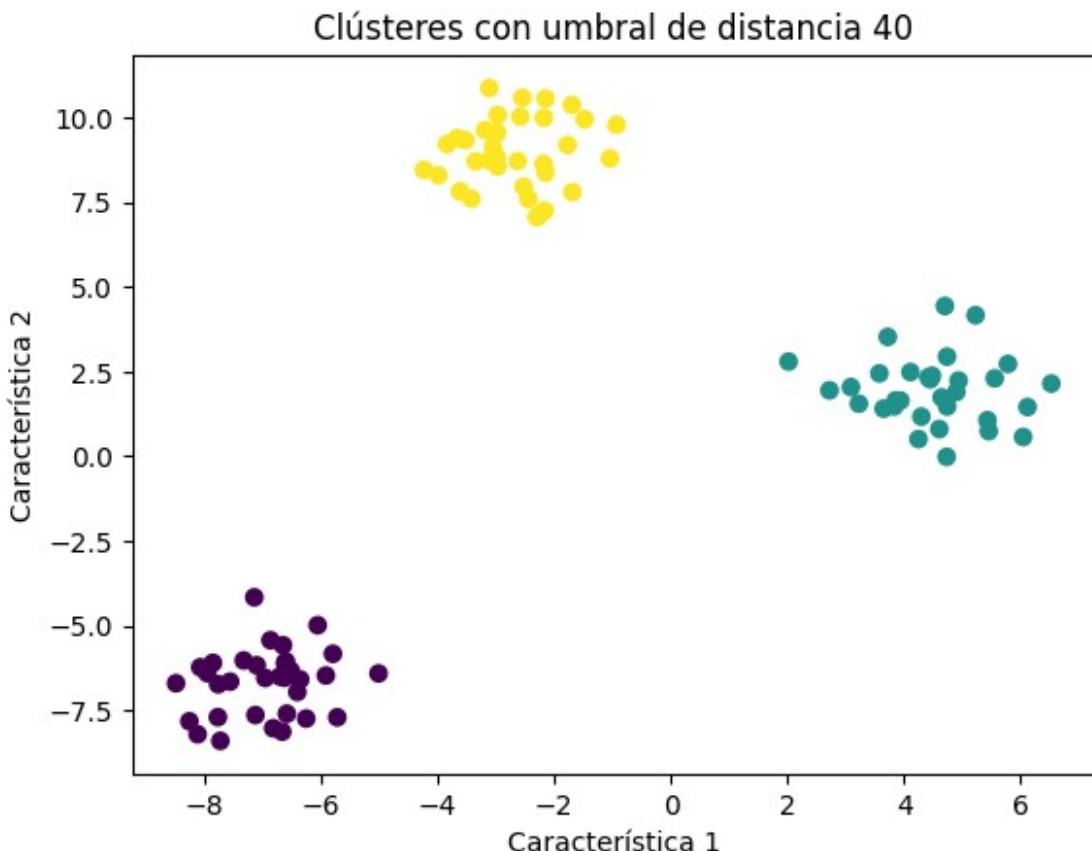
```

```

max_d = 40
clusters = fcluster(Z, max_d, criterion='distance')

# Visualizamos los clústeres
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis')
plt.title(f"Clústeres con umbral de distancia {max_d}")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.show()

```



Una vez realizada esta división, podemos identificar de manera clara a qué grupo pertenece cada dato, sin riesgo de confusión. De esta manera, hemos logrado una organización eficiente y lógica de los datos, lo que facilita su interpretación y análisis posterior.

Once this division is made, we can clearly identify which group each data point belongs to, with no risk of confusion. In this way, we have achieved an efficient and logical organization of the data, making it easier to interpret and analyze in the next steps.

# Metodo elbow

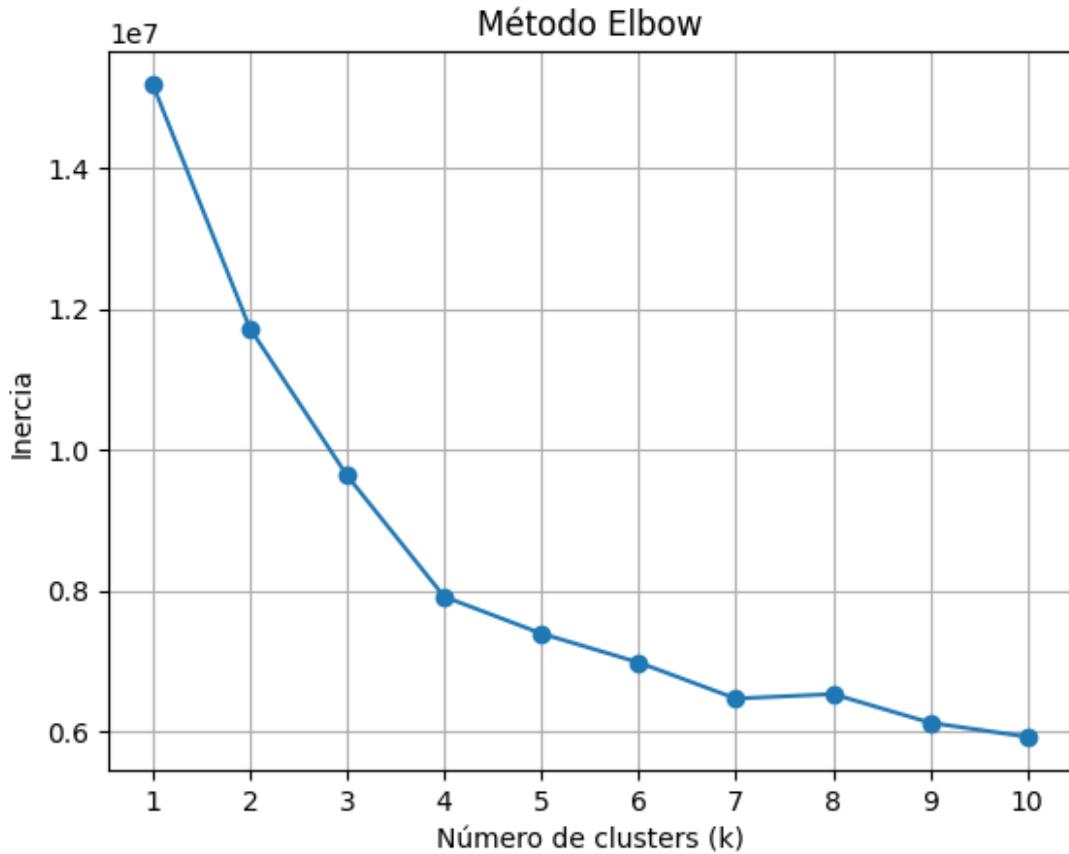
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Normaliza los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.select_dtypes(include=[np.number]))

# Calcular la inercia para diferentes valores de k
inertia = []
k_values = range(1, 11) # Probar de 1 a 10 clusters

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Graficar el método Elbow
plt.plot(k_values, inertia, marker='o')
plt.title('Método Elbow')
plt.xlabel('Número de clusters (k)')
plt.ylabel('Inercia')
plt.xticks(k_values)
plt.grid()
plt.show()
```



Como se puede observar en el gráfico generado mediante el método del Elbow (Codo), el análisis de los datos sugiere que el número óptimo de clusters para este conjunto de datos es 4. Esto se debe a que, al aumentar el número de clusters más allá de 4, se observa una disminución drástica en la variación de los valores de los datos, lo que indica que los clusters adicionales no aportan un valor significativo a la segmentación. En otras palabras, a partir de 4 clusters, la mejora en la agrupación de los datos se vuelve marginal, lo que nos lleva a concluir que 4 es el número adecuado para este análisis.

## Metodo silueta

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import numpy as np

# Ejemplo de DataFrame (reemplázalo con tus propios datos)
# Supón que tienes un DataFrame llamado 'df'
# df = pd.read_csv('tus_datos.csv') # Si tienes los datos en un
# archivo CSV, por ejemplo
```

```

# Definir X (usando solo las columnas de características)
X = df2.values # O df[['columna1', 'columna2', ...]].values si solo
quieres columnas específicas

# Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Definir los posibles valores de k (por ejemplo, de 2 a 10 clusters)
k_values = range(2, 4)

# Lista para almacenar las puntuaciones Silhouette
silhouette_scores = []

# Calcular el coeficiente Silhouette para diferentes valores de k
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X_scaled) # Usamos los datos escalados
    silhouette_avg = silhouette_score(X_scaled, kmeans.labels_)
    silhouette_scores.append(silhouette_avg)

# Graficar el método Silhouette
plt.plot(k_values, silhouette_scores, marker='o')
plt.title('Método Silhouette para diferentes valores de k')
plt.xlabel('Número de clusters (k)')
plt.ylabel('Puntuación Silhouette')
plt.xticks(k_values)
plt.grid(True)
plt.show()

```

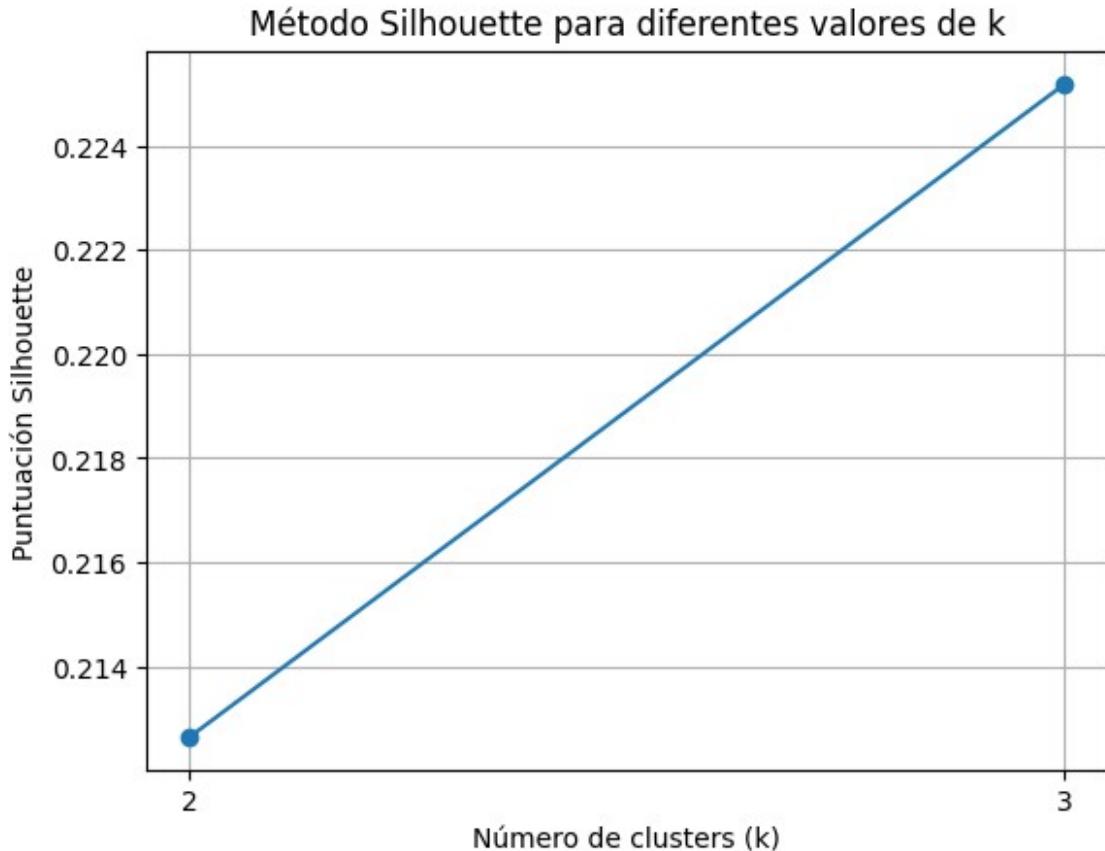


Grafico que se genero despues del analizis de los datos de silhouette (No el final)

```
# Importar las bibliotecas necesarias
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples

X = df2.select_dtypes(include=[np.number])

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # Normalizar los datos

# Aplicar el algoritmo K-means para realizar clustering
# Aquí elegimos 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)

# Calcular las puntuaciones de Silhouette para cada punto
sample_silhouette_values = silhouette_samples(X_scaled, y_kmeans)
```

```

# Crear un gráfico de Silhouette
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_xlim([-1, 1])
ax.set_ylim([0, len(X_scaled) + (4 + 1) * 10])

y_lower = 10
for i in range(4): # En este caso, tenemos 4 clusters (grupos)
    # Obtener los puntos que pertenecen al grupo i
    ith_cluster_silhouette_values = sample_silhouette_values[y_kmeans == i]
    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

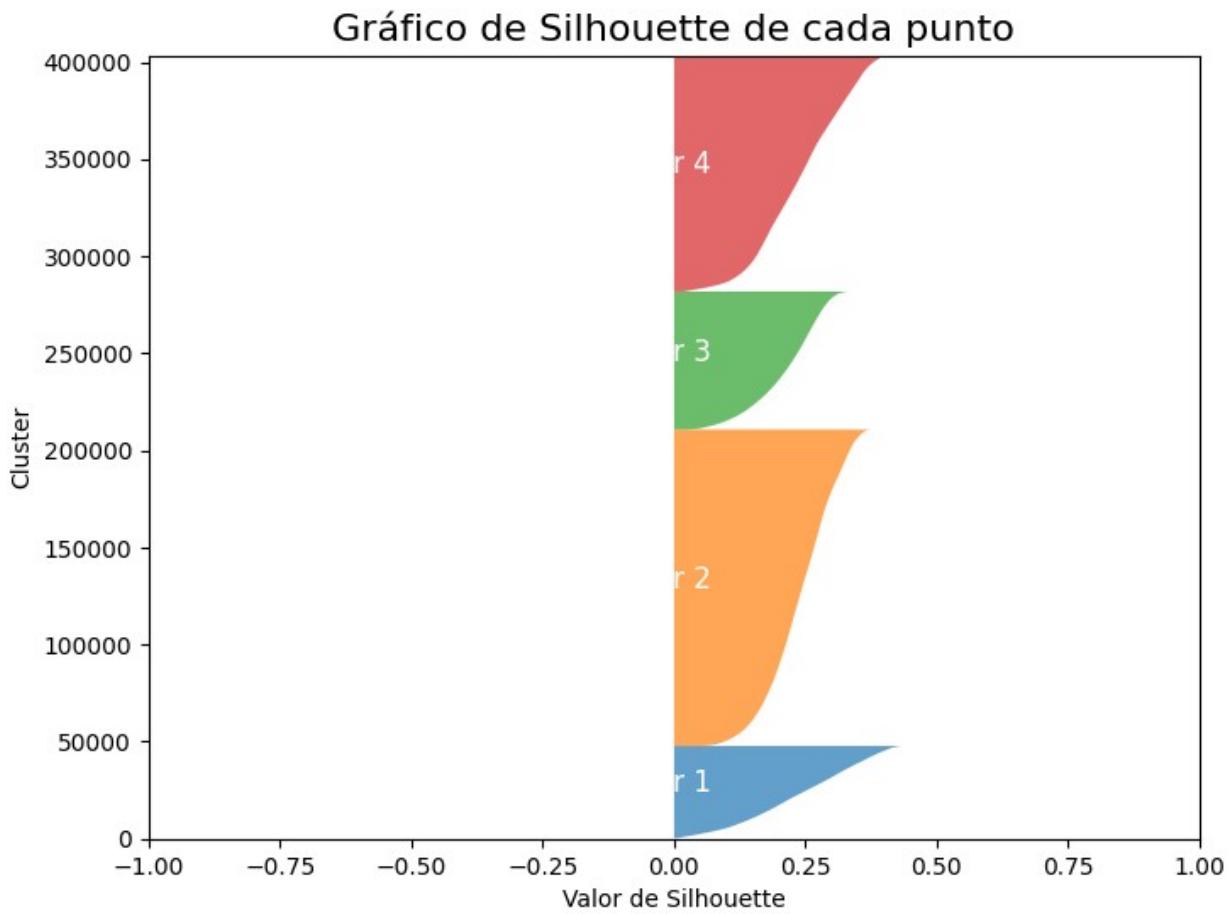
    # Dibujar los valores de Silhouette de cada punto
    ax.fill_betweenx(np.arange(y_lower, y_upper), 0,
ith_cluster_silhouette_values, alpha=0.7)

    # Marcar el promedio de Silhouette del cluster
    ax.text(-0.05, (y_lower + y_upper) / 2, f'Cluster {i+1}', horizontalalignment='center', fontsize=12, color='white')

    # Actualizar la posición de y_lower para el siguiente cluster
    y_lower = y_upper + 10

# Mostrar el gráfico
plt.title("Gráfico de Silhouette de cada punto", fontsize=16)
plt.xlabel("Valor de Silhouette")
plt.ylabel("Cluster")
plt.show()

```



Como se puede observar en el gráfico de Silhouette, los datos agrupados en los cuatro clusters con los que hemos estado trabajando en esta parte del proyecto muestran una distribución excelente. Los valores de Silhouette cercanos a +1 indican que los puntos dentro de cada cluster están muy cerca unos de otros, lo que sugiere que cada grupo está bien formado y es coherente. Esto significa que los elementos dentro de cada cluster tienen características similares, lo cual es un signo de que el modelo ha logrado agruparlos adecuadamente según sus similitudes.

Además, no solo los puntos dentro de cada cluster son consistentes, sino que los clusters están claramente separados entre sí. Es decir, los puntos de diferentes clusters están suficientemente distanciados unos de otros, lo que indica que los grupos no se solapan y que las diferencias entre ellos son significativas. Este buen nivel de separación es crucial, ya que sugiere que el algoritmo de clustering ha logrado identificar subgrupos distintos dentro del conjunto de datos, con pocos o ningún cruce entre ellos.

En resumen, el gráfico de Silhouette nos muestra que el modelo de clustering ha sido eficaz y preciso en la segmentación de los datos, lo que nos da confianza en que los 4 clusters seleccionados son una división adecuada y bien definida de los datos en este proyecto.