cf_project

November 7, 2023

1 Projecto Bootcamp Ciencia de Datos - Código Facilito

1.1 Introducción

Mi motivación para este proyecto era de interés personal y qería investigar datos relacionados con recetas disponibles en internet y su popularidad. Para ello encontré un dataset que contenía recetas y también el tráfico que genera cada una de ellas para así poder predecir su popularidad ya que hice la suposición de que a más tráfico más popularidad.

1.2 Validación de los datos

Este conjunto de datos consta de 947 filas y 8 columnas (sería una matriz de 947x8). De las columnas existentes en el dataset y, antes de proceder con su limpieza y transformaciones, 6 son numéricas y dos son de tipo string. Tras un estudio pormenorizado, pasamos a indicar el nombre de las columnas y la limpieza y/o transformación que apliqué sobre ellas:

1. **recipe**: Es un identificador único, una primary key del dataset. Contiene 947 valores únicos, y como el dataset tiene 947 filas esto quiere decir que no es necesario limpiar esta columna ya que no contiene duplicados. El valor mínimo es 1 y el valor máximo es 947. Es de tipo integer. Dado que el índice del dataframe que construímos a partir de los datos, tiene el mismo propósito, he decidido eliminar esta columna.

2. **calories, carbohydrate, sugar y protein**: Contienen 895 valores no nulos ergo, hay 52 valores nulos. Como veremos más adelante al crear la matriz `missingno`, estos valores nulos están en las mismas filas, por lo que he decido aplicar la media de cada columna en función de la categoría de alimentos, ya que he pensado que podría enriquecer de este modo los datos nulos en función de una característica relevante. Todas estas variables son de tipo float y ninguna de ellas tiene valores negativos, tal y como debería ser (lo contrario no tendría sentido).

3. **category**: Todas las filas de esta columna contienen valores no nulos. Ahora bién si observamos sus valores únicos, podemos ver que hay diferentes categorías para meat, chicken, chicken breast y pork. He decido transformar los tres últimos en una misma categoría: meat (carne), ya que creo que esta sería su categoría lógica. Hice lo mismo con vegetable y potato. Transformé el tipo de datos en categoría.

4. **servings**: Mi suposición es que debería ser un tipo numérico pero aparece como objeto en su lugar. Observé que hay dos valores que contienen un sufijo con la cadena " as a snack". He decido eliminar esa cadena final y transformar el tipo de datos a categórico (category).

5. **high_traffic**: Su valor es o bién 'High' o un valor null (nulo). He decido transformarlos a 1 y 0 para comprobar de manera más clara su distribución y poder preparar la variable para los modelos de Machine Learning. También lo he transformado un a tipo categórico (category).

Después de la transformación, obtendremos un dataframe de 947x7 con 4 columnas de tipo float y 3 columnas de tipo categórico (categorical) sin valores nulos.

[6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as style
import seaborn as sns
import missingno as msno
from sklearn.preprocessing import PowerTransformer, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer

# Configuramos el estilo de las gráficas de Matplotlib.
plt.style.use('ggplot')

# Se suprimen los FutureWarnings para mejorar la legibilidad del código.
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

[7]:
```
# Cargamos el conjunto de datos desde un archivo CSV en un DataFrame llamado␣
 ↪'df'.
df = pd.read_csv('data.csv')

# Mostramos las primeras filas del DataFrame para inspeccionar los datos␣
 ↪iniciales.
df.head()
```

[7]:

	recipe	calories	carbohydrate	sugar	protein	category	servings	high_traffic
0	1	NaN	NaN	NaN	NaN	Pork	6	High
1	2	35.48	38.56	0.66	0.92	Potato	4	High
2	3	914.28	42.68	3.09	2.88	Breakfast	1	
3	4	97.03	30.56	38.63	0.02	Beverages	4	
4	5	27.05	1.85	0.80	0.53	Beverages	4	

```
2          NaN
3         High
4          NaN
```

[8]:
```
# Obtenemos el número de filas y columnas del DataFrame.
df.shape
```

[8]:
```
(947, 8)
```

[9]:
```
# Información general del Dataframe. Se muestra información general sobre las␣
  ↪columnas,
# incluyendo los tipos de datos y la presencia de valores nulos.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 947 entries, 0 to 946
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   recipe        947 non-null    int64
 1   calories      895 non-null    float64
 2   carbohydrate  895 non-null    float64
 3   sugar         895 non-null    float64
 4   protein       895 non-null    float64
 5   category      947 non-null    object
 6   servings      947 non-null    object
 7   high_traffic  574 non-null    object
dtypes: float64(4), int64(1), object(3)
memory usage: 59.3+ KB
```

[10]:
```
# Comprobamos el número de valores únicos que contiene la columna 'recipe'.
# Verificamos que la columna 'recipe' contiene 947 valores únicos.
df['recipe'].nunique()
```

[10]:
```
947
```

[11]:
```
# Eliminamos la columna 'recipe' ya que podemos utilizar el índice del␣
  ↪Dataframe.
df.drop(['recipe'],axis=1, inplace=True)
```

[12]:
```
# La columna 'servings' debería ser numérica, pero nos retorna que es de tipo␣
  ↪object (dtype=object)
# Investigamos el porqué.
df['servings'].unique()
```

[12]:
```
array(['6', '4', '1', '2', '4 as a snack', '6 as a snack'], dtype=object)
```

[13]:
```
# Comprobamos que contiene el sufijo ' as a snack'. Eliminamos el sufijo ' as a␣
 ↪snack' ya que no aporta nada
# y transformamos la columna via astype en un tipo categórico (category).
df['servings'] = df['servings'].str.replace(' as a snack','').astype('category')
```

[14]:
```
# Comprobamos cuáles son los valores únicos para la columna 'category'
df['category'].unique()
```

[14]:
```
array(['Pork', 'Potato', 'Breakfast', 'Beverages', 'One Dish Meal',
       'Chicken Breast', 'Lunch/Snacks', 'Chicken', 'Vegetable', 'Meat',
       'Dessert'], dtype=object)
```

[15]:
```
# Agrupamos las categorías que representan carne (meat) y las que representan␣
 ↪vegetables.
# Se observan varias categorías relacionadas con alimentos. Decidimos agrupar␣
 ↪'Pork',
# 'Chicken Breast', y 'Chicken' bajo la categoría 'Meat', y 'Potato' bajo la␣
 ↪categoría 'Vegetable'.
df['category'] = df['category'].replace(['Pork','Chicken␣
 ↪Breast','Chicken'],'Meat')
df['category'] = df['category'].replace('Potato','Vegetable')

# Transfomamos la columna 'high_traffic' como una columna categórica␣
 ↪(categorical column).
df['category'] = df['category'].astype('category')
```

[16]:
```
# Utilizamos el método describe() para comprobar las variables numéricas y␣
 ↪observamos que no contienen
# valores negativos.
df.describe()
```

[16]:

	calories	carbohydrate	sugar	protein
count	895.000000	895.000000	895.000000	895.000000
mean	435.939196	35.069676	9.046547	24.149296
std	453.020997	43.949032	14.679176	36.369739
min	0.140000	0.030000	0.010000	0.000000
25%	110.430000	8.375000	1.690000	3.195000
50%	288.550000	21.480000	4.550000	10.800000
75%	597.650000	44.965000	9.800000	30.200000
max	3633.160000	530.420000	148.750000	363.360000

[17]:
```
# Comprobamos los valores únicos para la columna 'high_traffic'.
# Se observa que esta columna contiene el valor 'High' y valores nulos (NaN).
df['high_traffic'].unique()
```

[17]:
```
array(['High', nan], dtype=object)
```

[18]:
```
# Modificamos los valores de la columna 'high_traffic' de manera que los␣
 ↪valores con el literal 'High' los
# transformamos al valor 1, y los valores nulos al valor 0.
# Finalmente modificamos la columna 'high_traffic' a un tipo categórico.
reps = {'High':1, np.nan:0}
df['high_traffic'] = df['high_traffic'].replace(reps).astype('category')
```

[19]:
```
# Visualizamos la distribution de los valores de la columna 'high_traffic'.
df['high_traffic'].value_counts()
```

[19]:
```
1.0    574
0.0    373
Name: high_traffic, dtype: int64
```

[20]:
```
# Comprobamos si los valores nulos en las columnas 'calories', 'carbohydrate',␣
 ↪'sugar' y 'protein'
# se encuentran en las mismas filas y, vemos que se verifica que los valores␣
 ↪nulos de estas columnas
# están en las mismas filas.
df[df['calories'].isna()].head()
```

[20]:

	calories	carbohydrate	sugar	protein	category	servings	high_traffic
0	NaN	NaN	NaN	NaN	Meat	6	1.0
23	NaN	NaN	NaN	NaN	Meat	2	0.0
48	NaN	NaN	NaN	NaN	Meat	4	0.0
82	NaN	NaN	NaN	NaN	Meat	4	1.0
89	NaN	NaN	NaN	NaN	Meat	6	1.0

###**Librería missingno**

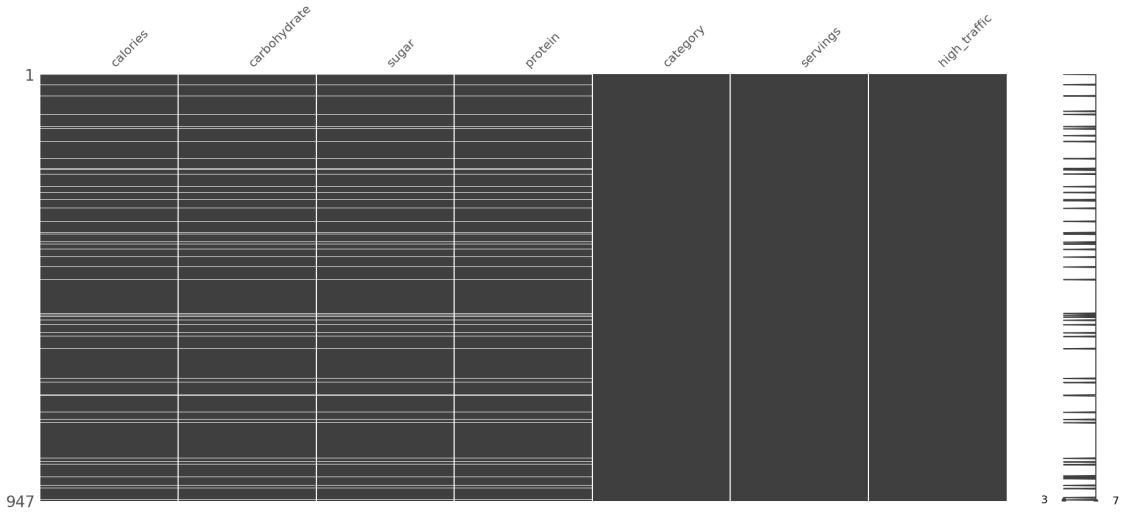
Utilizamos la librería **missingno** para visualizar los valores nulos en el DataFrame. Nos ayudará a comprender la distribución de estos valores 'especiales' en un DataFrame proporcionando una vista general de la ubicación de los valores nulos. Así podemos identificar patrones visuales, es decir podemos ver qué filas o columnas contienen valores nulos.

El método `msno.matrix(df)` muestra una gráfica con los colores blanco (valores nulos) y negro (valores no nulos) para representar los valores nulos en cada fila y columna. Así podemos buscar patrones y detectar, por ejemplo, si la presencia o ausencia de datos en una columna está relacionada con otra columna.

Dependiendo de la cantidad y de los patrones que podemos indentificar respecto a los valores nulos, podemos decidir qué estrategia utilizar, por ejemplo, si debemos eliminar ciertas filas o columnas, o imputar los valores faltantes con algún estimador como la media.

[21]:
```
# Utilizamos la libreria 'missingno' para visualizar los valores nulos en el␣
 ↪Dataframe.
msno.matrix(df)
```

[21]:
```
<Axes: >
```



###**Observación**

Los datos nulos se encuentran en las columnas 'calories', 'carbohydrate', 'sugar' y 'protein', y siempre en las mismas filas por lo tanto podemos sugerir la estrategia de imputar los valores nulos con la media.

Enumeramos las razones para ello:

1. **Inspección de los datos nulos**: Dado que los datos nulos estan situados en las mismas filas en el caso de las columnas mencionadas, es razonable asumir que estos valores están relacionados y siguen un patrón común. Esto sugiere que no se trata de valores nulos aleatorios.
2. **Preservación de datos**: Al imputar con la media, informaremos los valores nulos con estimaciones basadas en las observaciones disponibles. Esto nos permitirá conservar todas las filas y no perderemos datos valiosos que podremos usar en posteriores análisis y en los modelos de Machine Learning.
3. **Agrupación por 'category'**: Agruparemos los datos por la columna 'category' antes de imputar con la media. Los valores nulos se dan en un conjunto específico de filas y están relacionados con recetas específicas. Dado que las recetas se agrupan en diferentes categorías usaremos esta información para realizar la imputación. Al agrupar los datos en función de la columna 'category', esto significa que las recetas se dividen en grupos según su categoría, lo que crea subconjuntos de datos separados para cada categoría. Por ejemplo, todas las recetas de carne se agrupan en un conjunto, todas las recetas de verduras en otro, y así sucesivamente. Dentro de cada grupo, calculamos la media de las columnas 'calories', 'carbohydrate', 'sugar' y 'protein' y cada valor nulo (por columna) se informa con la media obtenida respecto al grupo al que pertenece la receta.

[22]:
```
# Como hemos observado, los valores nulos se encuentran ubicados en las columnas
# 'calories','carbohydrate','sugar' y 'protein'.
cols_na = ['calories','carbohydrate','sugar','protein']
```

```
# Para enriquecer el Dataframe y no perder filas, imputamos los valores nulos␣
 ↪que encontramos en las
# columnas 'calories','carbohydrate','sugar' y 'protein' por la media de las␣
 ↪columnas resultante de
# agrupar por la columna 'category'.
for col in cols_na:
    df[col] = df.groupby('category', observed=True)[col].transform(lambda x: x.
 ↪fillna(x.mean()))
```

[23]:
```
# Observamos el Dataframe resultante después de aplicar estas modificaciones.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 947 entries, 0 to 946
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   calories      947 non-null    float64
 1   carbohydrate  947 non-null    float64
 2   sugar         947 non-null    float64
 3   protein       947 non-null    float64
 4   category      947 non-null    category
 5   servings      947 non-null    category
 6   high_traffic  947 non-null    category
dtypes: category(3), float64(4)
memory usage: 33.2 KB
```

[24]:
```
# Estos pasos nos han permitido limpiar y preparar nuestro DataFrame para su␣
 ↪posterior análisis y modelado.
# Miramos las primeras 5 filas del DataFrame.
df.head()
```

[24]:

	calories	carbohydrate	sugar	protein	category	servings	high_traffic
0	577.808129	25.366226	6.091032	45.082355	Meat	6	1.0
1	35.480000	38.560000	0.660000	0.920000	Vegetable	4	1.0
2	914.280000	42.680000	3.090000	2.880000	Breakfast	1	0.0
3	97.030000	30.560000	38.630000	0.020000	Beverages	4	1.0
4	27.050000	1.850000	0.800000	0.530000	Beverages	4	0.0

1.3 Resúmen

Hasta ahora hemos realizado una serie de pasos esenciales, como la eliminación de columnas innecesarias, la corrección de tipos de datos, la agrupación de categorías relacionadas y la imputación de valores nulos.

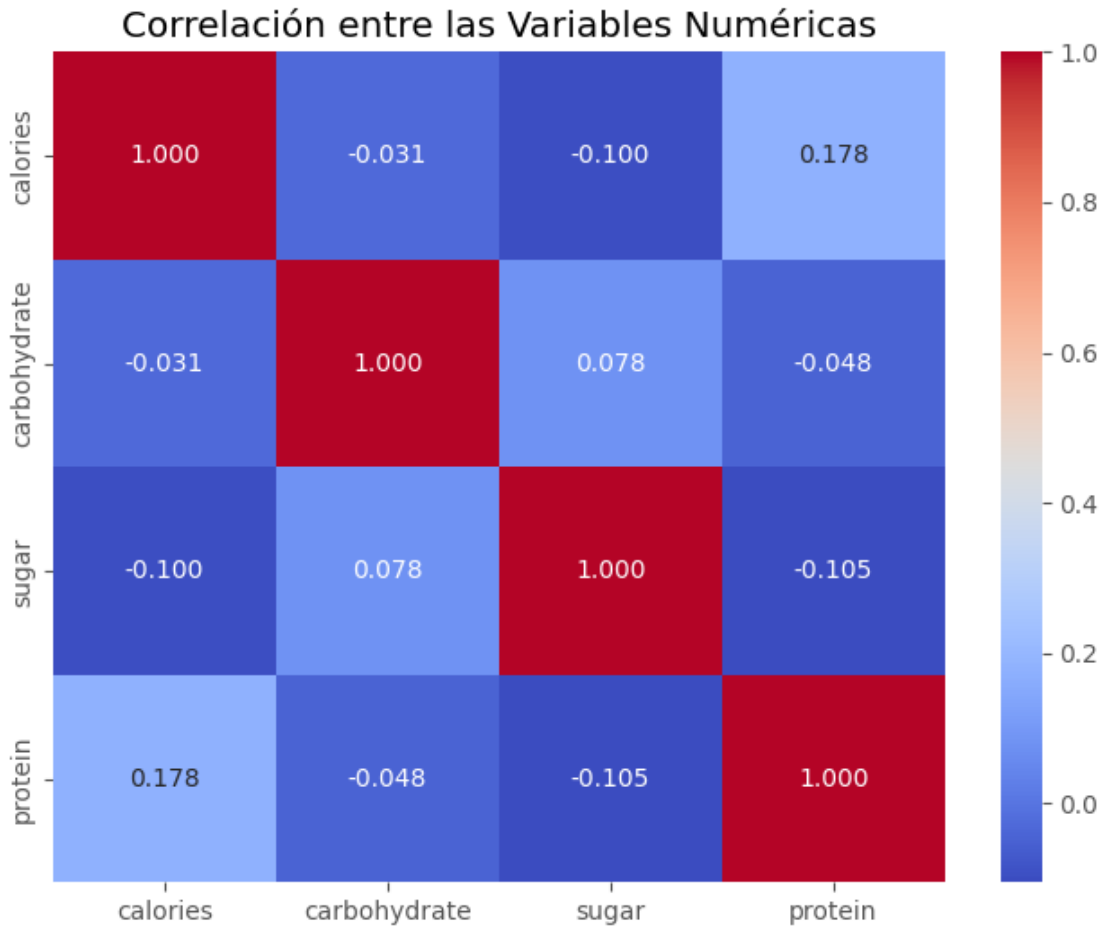
Ahora bién, como ya sabemos el análisis de datos y la construcción de modelos de machine learning es un proceso iterativo, continuamos pués explorando y refinando los datos disponibles para lograr unos mejores resultados.

1.4 Análisis Exploratorio

Inicialmente no podemos afirmar que exista una correlación positiva o negativa entre las variables numéricas. En general, existen correlaciones débiles, siendo la más relevante la correlación entre las proteínas y las calorías (0.178).

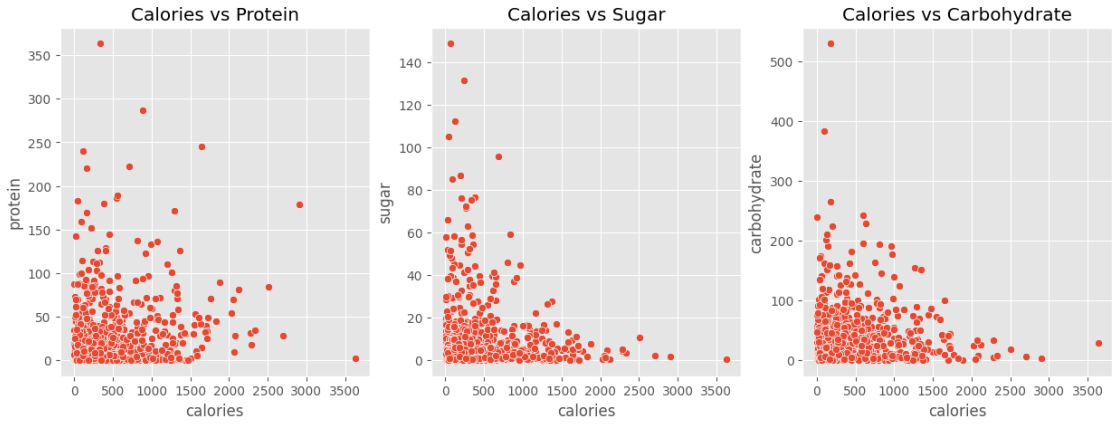
Podemos observar que las variables numéricas (sin considerar la variable 'servings') tienen una distribución significativa hacía la derecha, tal y como podemos apreciar en los histogramas, especialmente la columna 'protein'. Muchos algoritmos de Machine Learning necesitan que las variables numéricas tengan una distribución que se acerque lo máximo posible a una distribución gaussiana/normal. Esta es la razón por la que hemos hecho una transformación usando `PowerTransformer` de la librería `scikit-learn`. Una vez realizada la transformación, los histogramas muestran el antes y el después de esta transformación (a la izquierda el antes y a la derecha el después).

[25]:
```
# Usamos un gráfico de tipo Heatmap de la librería Seaborn para comprobar la␣
 ↪correlación entre
# las columnas numéricas.

# Aumentamos el tamaño del gráfico para mostrar los valores de correlación de␣
 ↪manera más clara
plt.figure(figsize=(8, 6))

# Calculamos la matriz de correlación
numeric = df[['calories','carbohydrate','sugar','protein']]
correlation_matrix = numeric.corr()

# Creamos el mapa de calor con anotaciones de valores
sns.heatmap(correlation_matrix, annot=True, fmt=".3f", cmap="coolwarm",␣
 ↪cbar=True)

# Establece el título
plt.title("Correlación entre las Variables Numéricas")

# Muestra el gráfico
plt.show()
```



[26]:
```
# El gráfico pair plot de la librería Seaborn nos proporciona información sobre␣
 ↪la distribución de los datos y
# mejora su visualización al incorporar el color para representar la␣
 ↪distribución de cada característica.
# Al incluir el color, podemos observar de manera efectiva dimensiones␣
 ↪adicionales y patrones dentro del gráfico.

sns.pairplot(df.loc[:, ['calories', 'carbohydrate', 'sugar', 'protein',␣
 ↪'high_traffic']], hue='high_traffic');
```

[27]:
```
# Utilizamos los scatter plots de Seaborn para comprobar la distribución de la␣
 ↪columna 'calories' vs. las otras
# columnas numéricas ie, 'protein', 'sugar' y 'carbohydrate'.

fig, axes = plt.subplots(1,3,figsize=(15,5))
sns.scatterplot(y=df['protein'],x=df['calories'],ax=axes[0]).
 ↪set(title='Calories vs Protein')
sns.scatterplot(y=df['sugar'],x=df['calories'],ax=axes[1]).set(title='Calories␣
 ↪vs Sugar')
sns.scatterplot(y=df['carbohydrate'],x=df['calories'],ax=axes[2]).
 ↪set(title='Calories vs Carbohydrate');
```



###Transformación

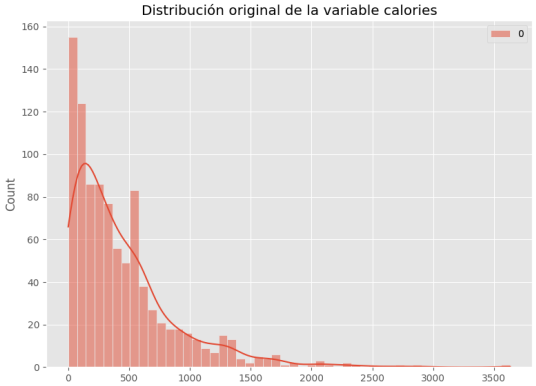
El uso de `PowerTransformer` y la normalización de valores de las variables numéricas, como 'calories', 'sugar', 'protein' y 'carbohydrate', es importante de caras a la preparación de los datos para poder aplicarmodelos de Machine Learning.

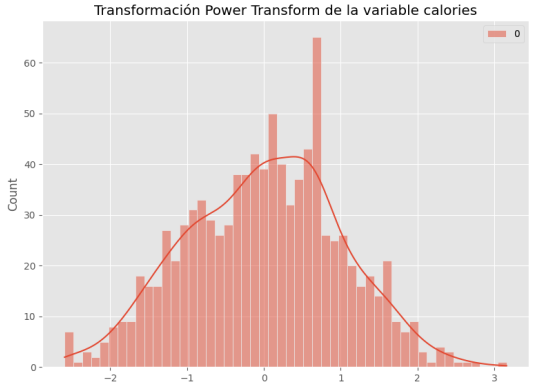
Las variables numéricas a menudo siguen una distribución que se aleja de una distribución normal o gaussiana pero resulta que los modelos de Machine Learning que queremos aplicar funcionan mejor cuando las variables numéricas tienen una distribución más cercana a una distribución normal. Entonces, mediante un proceso de escalado usando `PowerTransformer` transformaremos los datos de manera que la distribución de la variable resultante se acerque más a una distribución normal.

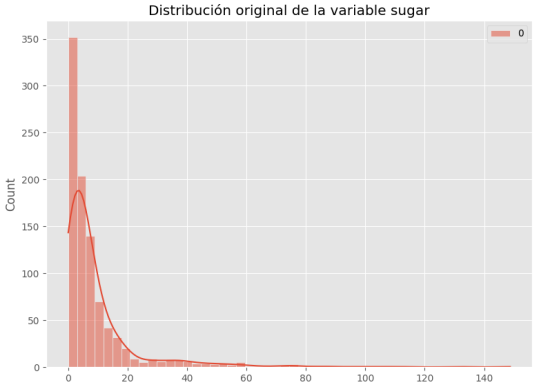
A continuación usaremos `PowerTransformer` para transformar las variables 'calories', 'sugar', 'protein', y 'carbohydrate' y mostraremos el antes y el después de la transformación de manera que podremos observar cómo se ve la distribución original de cada variable y cómo se ve después de transformarla aplicando `PowerTransformer`. Los histogramas antes y después de la transformación mostrarán cómo `PowerTransformer` puede ayudar a normalizar las distribuciones de las variables.

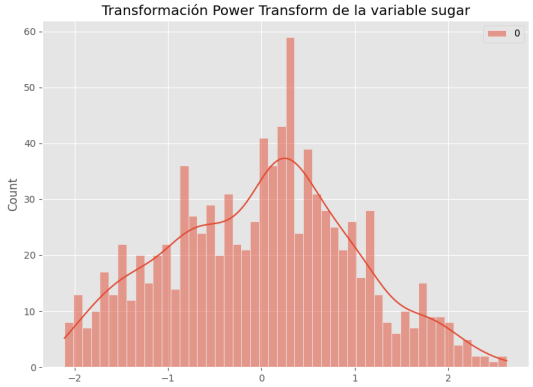
[28]:
```
# Ahora realizamos la transformacion de las variables aplicando␣
 ↪PowerTransformer y 'normalizando'
# así su distribución.

cols = ['calories', 'sugar', 'protein', 'carbohydrate']

def test_transformers(columns):
    my_pt = PowerTransformer()
    fig = plt.figure(figsize=(20,30))
    j = 1
    for i in columns:
        array = np.array(df[i]).reshape(-1, 1)
        y = my_pt.fit_transform(array)

        plt.subplot(4,2,j)
        sns.histplot(array, bins = 50, kde = True)
```

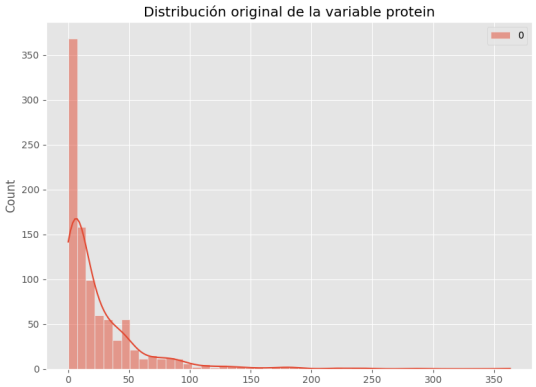
```
        plt.title(f"Distribución original de la variable {i}")

        plt.subplot(4,2,j+1)
        sns.histplot(y, bins = 50, kde = True)
        plt.title(f"Transformación Power Transform de la variable {i}")
        j += 2

# Mostramos las distribuciones antes y después de aplicar el Power Transform
test_transformers(cols)
```

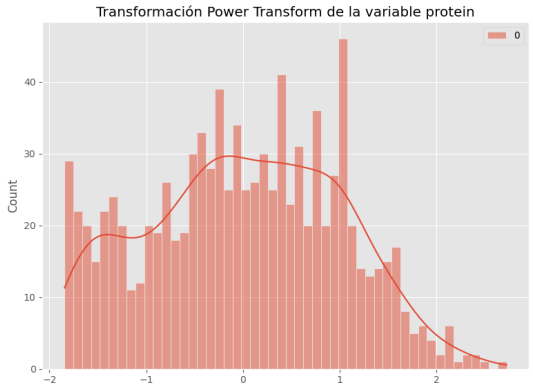

Distribución original de la variable calories
Count
0

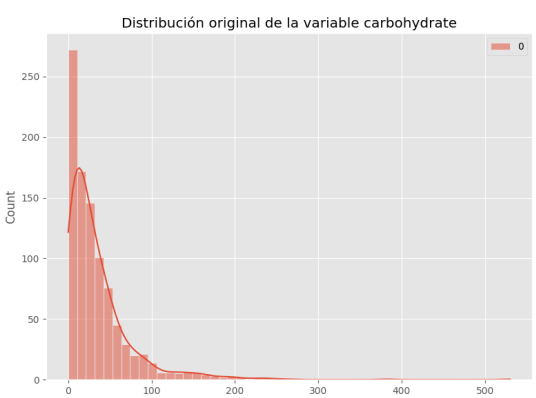

Transformación Power Transform de la variable calories
Count
0

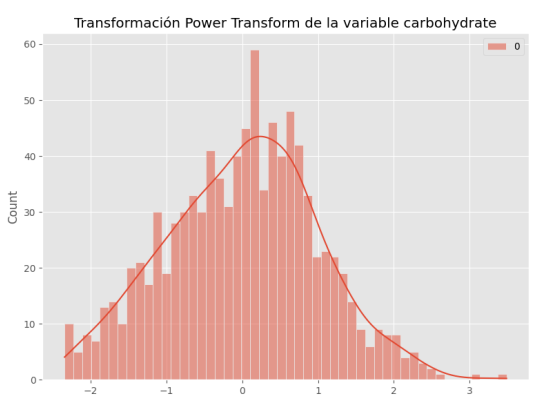

Distribución original de la variable sugar
Count
0


Transformación Power Transform de la variable sugar
Count
0


Distribución original de la variable protein
Count
0


Transformación Power Transform de la variable protein
Count
0


Distribución original de la variable carbohydrate
Count
0


Transformación Power Transform de la variable carbohydrate
Count
0

1.5 Resúmen

En esta parte del proyecto, hemos realizado un análisis exploratorio del dataset y hemos aplicado una transformación a las variables numéricas utilizando `PowerTransformer` de `scikit-learn` para poder ajustar sus distribuciones.

Algunas observaciones clave:

1. **Matriz de Correlación**: He utilizado un heatmap para visualizar la correlación entre las variables numéricas. Esto es útil para identificar relaciones entre las variables. He observado que las correlaciones son en su mayoría débiles, lo que indica que no hay una dependencia lineal fuerte entre estas variables. He comprobado que la mayor correlación se da entre las variables 'calories' y 'protein'.
2. **Gráfico Pair Plot**: El gráfico pair plot es una herramienta útil para explorar la distribución de datos y observar cómo se relacionan entre sí. Lo he utilizado para visualizar la distribución de las variables numéricas y cómo se relacionan con la variable objetivo 'high_traffic'. Esto puede ayudarnos a identificar patrones visuales en los datos.
3. **Scatter Plots**: La creación de scatter plots me ha permitido analizar la relación entre la variable 'calories' y las otras variables numéricas ('protein', 'sugar' y 'carbohydrate'). Estos gráficos proporcionan una representación visual de cómo estas variables se relacionan entre sí. Por ejemplo, he observado cómo la variable 'calories' se relaciona con las variables 'protein', 'sugar' y 'carbohydrate'.
4. **Transformación PowerTransformer**: He aplicado PowerTransformer a las variables numéricas para ajustar sus distribuciones. Esto es importante ya que muchos algoritmos de machine learning funcionan mejor con datos que siguen una distribución gaussiana o normal. La comparación de las distribuciones antes y después de la transformación muestra claramente cómo esta técnica puede mejorar la simetría de los datos.

En general, este análisis exploratorio es esencial para comprender los datos y cómo se distribuyen. La transformación de las variables numéricas es un paso importante para preparar los datos para la construcción de modelos de machine learning.

1.6 Desarrollo del modelo

Nuestro objetivo es comprobar si podemos clasificar o no una receta en función del tráfico (high_traffic) que recibe (alto: 1 o bajo: 0), es decir lo que quiero es construir modelos de machine learning para predecir si una receta generará un tráfico alto o no. Por lo tanto, estamos ante un caso de **clasificación** (tráfico alto o bajo) y algunos modelos usados para la clasificación binaria son la **Logistic Regression**, **Máquinas de Soporte Vectorial (SVM)**, **Bosques Aleatorios**, y otros. Utilizaré para ello como modelo base la **Logistic Regression** y después el modelo **SVM** y, finalmente los compararé.

Para evaluar los modelos, he decido utilizar las siguientes métricas: **accuracy, F1, precision and recall**. En las siguientes líneas, veremos cómo he preparado los datos para poder aplicar los modelos de Machine Learning, el ajuste del modelo y los resultados obtenidos. Observaremos también qué tipos de recetas tienen éxito al aumentar el tráfico a la web.

He utilizado un flujo de trabajo (pipeline en el argot) que contiene una transformación a nivel de escala, una instancia del modelo y un selector para obtener las características más relevantes. He escogido también una lista de parámetros adaptados para la Logistic Regression.

Luego obtuve las métricas (F1, recall, precision, accuracy), grafiqué el peso de todas las características utilizadas y, finalmente, filtré los valores de predicción asociados a un tráfico alto con el fin de mostrar qué tipo de recetas tienen más popularidad y éxito, usando para ello un gráfico de barras que muestra las categorías más comunes.

1.6.1 Modificando y transformando los datos: Preparación

Aplicaremos diferentes métodos de preparación y transformación al constatar que tenemos diferentes tipos de variables, por un lado tenemos variables numéricas y por otro categóricas.

Para las variables categóricas, utilizaremos la codificación `one-hot`, ya que proporciona una mayor precisión al aplicar los modelos.

En el caso de las variables numéricas y después de los resultados que hemos observado anteriormente, utilizaremos `PowerTransformer`.

[29]:
```
from sklearn.preprocessing import OneHotEncoder

# Las columnas categóricas son
categoric_cols = ['category','servings']

# Instanciamos un objeto de la clase OneHotEncoder
ohe = OneHotEncoder()

# Transformación de las columnas categóricas
encoded_cols = ohe.fit_transform(df[categoric_cols])
encoded_df = pd.DataFrame(encoded_cols.toarray(),
                          columns=ohe.get_feature_names_out(categoric_cols))

df = pd.concat([df, encoded_df], axis=1)

# Eliminamos las columnas categóricas iniciales
df_enc = df.drop(categoric_cols, axis=1)

# Mostramos el Dataframe después de las transformaciones aplicadas sobre él
df_enc.head()
```

[29]:

	calories	carbohydrate	sugar	protein	high_traffic	\
0	577.808129	25.366226	6.091032	45.082355	1.0	
1	35.480000	38.560000	0.660000	0.920000	1.0	
2	914.280000	42.680000	3.090000	2.880000	0.0	
3	97.030000	30.560000	38.630000	0.020000	1.0	
4	27.050000	1.850000	0.800000	0.530000	0.0	

	category_Beverages	category_Breakfast	category_Dessert	\
0	0.0	0.0	0.0	

```
1                 0.0                 0.0              0.0
2                 0.0                 1.0              0.0
3                 1.0                 0.0              0.0
4                 1.0                 0.0              0.0

   category_Lunch/Snacks  category_Meat  category_One Dish Meal  \
0                    0.0            1.0                     0.0
1                    0.0            0.0                     0.0
2                    0.0            0.0                     0.0
3                    0.0            0.0                     0.0
4                    0.0            0.0                     0.0

   category_Vegetable  servings_1  servings_2  servings_4  servings_6
0                 0.0         0.0         0.0         0.0         1.0
1                 1.0         0.0         0.0         1.0         0.0
2                 0.0         1.0         0.0         0.0         0.0
3                 0.0         0.0         0.0         1.0         0.0
4                 0.0         0.0         0.0         1.0         0.0
```

[30]:
```
# Ahora vamos a aplicar la transformación PowerTransformer
my_pt = PowerTransformer()
df_enc[['calories','protein','carbohydrate','sugar']] = my_pt.
 ↪fit_transform(df_enc[['calories','protein','carbohydrate','sugar']])

# Creamos las variables correspondientes. En el axis=1 eliminamos la que hace␣
 ↪referencia a
# 'high_traffic' y añadimos la 'y' que contiene a 'high_traffic'.
X = df_enc.drop('high_traffic', axis=1)
y = df_enc['high_traffic']
```

1.7 Resúmen

Hasta ahora mi objetivo ha sido intentar realizar una preparación lo más adecuada posible de los datos disponibles antes de aplicar modelos de machine learning para la clasificación binaria de recetas en función de su tráfico.

La codificación `one-hot` de las variables categóricas permite a los modelos trabajar con estas variables de manera más efectiva. Además, he aplicado la transformación `PowerTransformer` a las variables numéricas. Esto es importante para asegurar que sigan una distribución lo más cercana posible a una distribución normal, y así poder optimizar el rendimiento de algunos algoritmos de machine learning.

###Aplicamos el modelo de Logistic Regression

Consideraciones a tener en cuenta.

####GridSearchCV

GridSearchCV es una técnica utilizamos para buscar los mejores hiperparámetros asociados a un modelo de machine learning. Los hiperparámetros son configuraciones que pueden ser modificadas

tales como la elección del kernel, la tasa de aprendizaje en redes neuronales o la profundidad del árbol en árboles de decisión.

El papel de GridSearchCV es vital para ayudar a encontrar la combinación óptima de hiperparámetros que maximice el rendimiento del modelo. Funciona de la siguiente manera:

1. Define un conjunto de hiperparámetros y sus valores posibles. Por ejemplo, los hiperparámetros podrían incluir el tipo de kernel, el valor de C (regularización), y otros parámetros específicos del modelo.
2. GridSearchCV realiza una búsqueda exhaustiva a través de todas las combinaciones posibles de valores de hiperparámetros. Puedes especificar qué métrica de evaluación deseas optimizar, como 'precision', 'recall', 'accuracy', etc.
3. Se entrena y evalúa el modelo con cada combinación de hiperparámetros utilizando una validación cruzada.
4. Al final, GridSearchCV devuelve la mejor combinación de hiperparámetros que maximiza la métrica de evaluación especificada.

Utilizaré GridSearchCV para encontrar la mejor combinación de hiperparámetros para el modelo de Regresión Logística.

####**Matriz de confusión**

La matriz de confusión es una herramienta fundamental para evaluar el rendimiento de un modelo de clasificación. Muestra la relación entre las predicciones del modelo y el valor real en el conjunto de datos de prueba.

La matriz de confusión se divide en cuatro partes:

1. **Verdaderos positivos (True Positives - TP)**: Representa los casos en los cuáles el modelo predijo correctamente el valor como positivo (high_traffic = 1) cuando el verdadero valor era positivo.
2. **Falsos positivos (False Positives - FP)**: Representa los casos en los que el modelo predijo incorrectamente el valor como positivo (high_traffic = 1) cuando el valor real era negativo (high_traffic = 0).
3. **Verdaderos negativos (True Negatives - TN)**: Representa los casos en los que el modelo predijo correctamente el valor como negativo (high_traffic = 0) cuando el valor real era negativo.
4. **Falsos negativos (False Negatives - FN)**: Representa los casos en los que el modelo predijo incorrectamente el valor como negativo (high_traffic = 0) cuando el valor real era positivo (high_traffic = 1).

La matriz de confusión nos permite evaluar aspectos como la precisión (accuracy), la precisión (precision) y el recall (también llamado sensibilidad). Estas métricas se calculan a partir de los valores obtenidos en la matriz de confusión y ayudan a entender el rendimiento del modelo en tareas de clasificación.

- **Precisión (Precision)**: Mide la proporción de casos positivos predichos **correctamente** en comparación con todos los casos positivos predichos. Se calcula como TP / (TP + FP).

- **Recall (Sensibilidad)**: Mide la proporción de casos positivos predichos **correctamente** en comparación con todos los casos positivos reales en el conjunto de datos. Se calcula como TP / (TP + FN).
- **Exactitud (Accuracy)**: Mide la proporción de predicciones **correctas** en comparación con todas las predicciones. Se calcula como (TP + TN) / (TP + TN + FP + FN).

La matriz de confusión y estas métricas son esenciales para identificar si el modelo tiende a cometer errores de falsos positivos o falsos negativos y ajustarlo en consecuencia.

Utilizaré la matriz de confusión para visualizar cómo se distribuyen las predicciones del modelo de Regresión Logística en comparación con los valores reales en los datos de prueba. Esto me permitirá evaluar y comprender mejor el rendimiento del modelo según verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos.

[31]:
```
from sklearn.linear_model import LogisticRegression

# Separacion de los datos en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=17)

# Definimos el flujo o pipeline que usamos para aplicar el modelo
my_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('selector', SelectKBest(f_classif)),
    ('classifier', LogisticRegression())
])

# Definición de los hyperparametros para GridSearchCV
parameters = {
    'selector__k': ['all'],
    'classifier__C': [0.001, 0.1, 1, 10, 100, 1000],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__solver': ['liblinear','saga']
}

# Instanciamos GridSearchCV con 5-fold cross-validation
my_grid_search = GridSearchCV(
                  my_pipeline,
                  parameters,
                  cv=5,
                  scoring='precision'
                 )

# Con los datos de train realizamos un fit de GridSearchCV
my_grid_search.fit(X_train, y_train)

# Obtenemos el mejor estimador
my_best_estimator = my_grid_search.best_estimator_
```

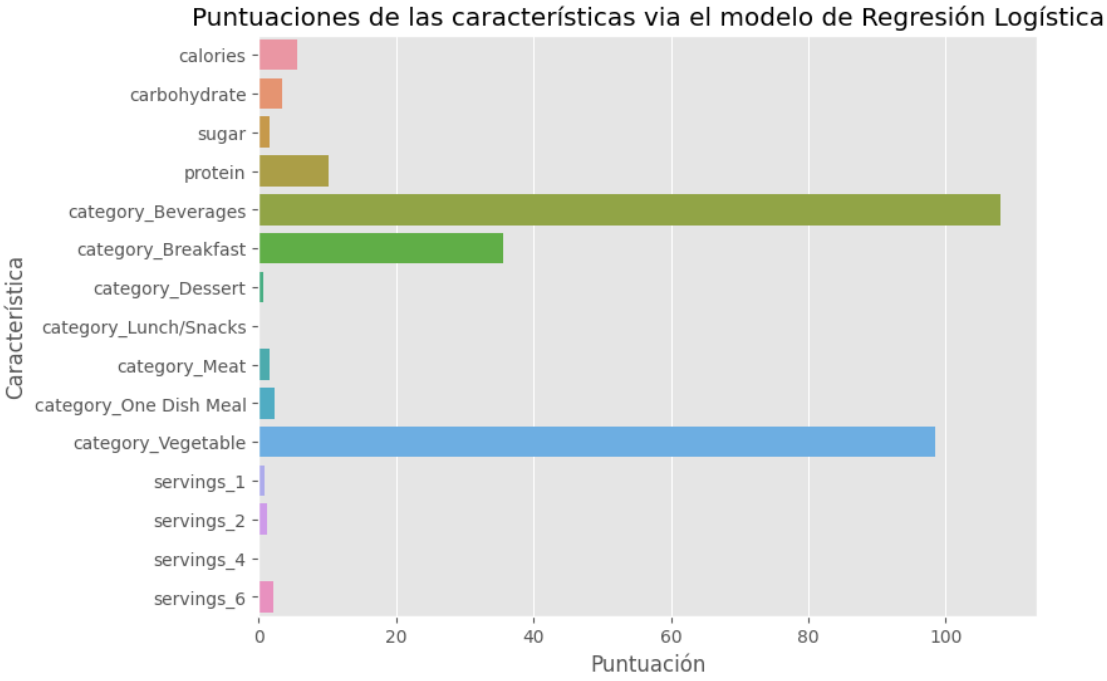
```
# Mostramos los parámetros del estimador obtenido
print("Parámetros del mejor estimador:", my_best_estimator.get_params())

# Ahora vamos a evaluar el estimador usando los datos de test
y_prediccion_proba = my_best_estimator.predict_proba(X_test)
my_threshold = 0.6
y_prediccion = (y_prediccion_proba[:,1] > my_threshold).astype(int)

accuracy = accuracy_score(y_test, y_prediccion)
f1 = f1_score(y_test, y_prediccion)
precision = precision_score(y_test, y_prediccion)
recall = recall_score(y_test, y_prediccion)

# Mostramos los resultados de la Regresión Logística
print('Resultados de la Regresión Logística:')
print("Accuracy:", accuracy)
print("F1:", f1)
print("Precision:", precision)
print("Recall:", recall)

# Obtenemos los indices de las recetas con 'high_traffic' respecto a los datos␣
 ↪de test
my_high_traffic_indexs = np.where(y_prediccion == 1)[0]

# Conseguimos las recetas (recipes) que tienen un 'high_traffic' alto
my_high_traffic_recipes = X_test.iloc[my_high_traffic_indexs]

# Vamos a graficar las puntuaciones respecto a las características seleccionadas
selector = my_best_estimator.named_steps['selector']
selected_indices = selector.get_support(indices=True)

# 'selected_scores': Contiene las puntuaciones (scores) de las características␣
 ↪(features) seleccionadas por el
# modelo de Regresión Logística. Estas puntuaciones indican el peso de las␣
 ↪características para predecir
# si una receta generará tráfico alto o bajo.
selected_scores = selector.scores_[selected_indices]

# 'selected_features': Aquí tenemos las características (features)␣
 ↪seleccionadas que se corresponden con
# las columnas del dataset original que se utilizaron como características para␣
 ↪entrenar el modelo.
# Cada característica se asocia con su respectiva puntuación.
selected_features = X.columns[selected_indices]
```

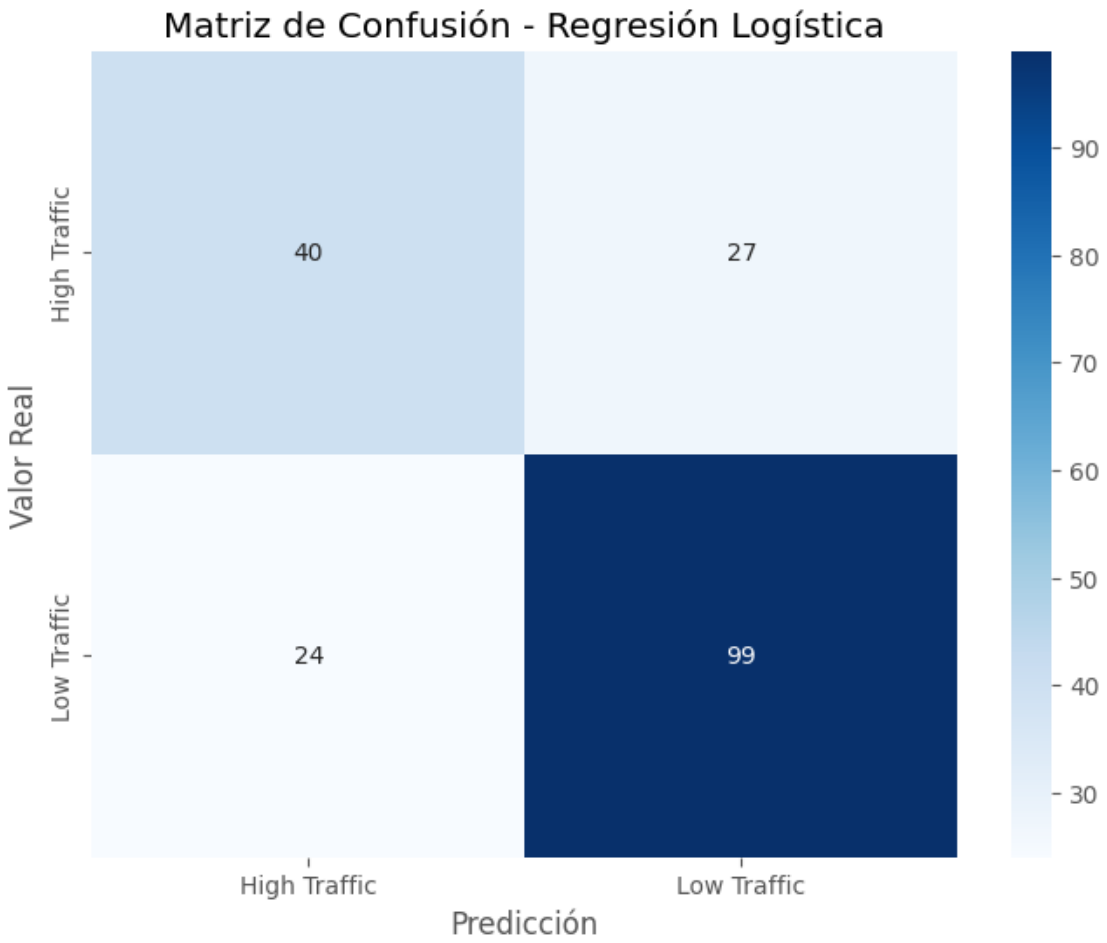
```python
# Mostramos un gráfico de tipo bar plot con las puntuaciones respecto a las␣
↪características estudidadas
plt.figure(figsize=(8, 6))
sns.barplot(x=selected_scores, y=selected_features)
plt.title('Puntuaciones de las características via el modelo de Regresión␣
↪Logística')
plt.xlabel('Puntuación')
plt.ylabel('Característica')
plt.show()

from sklearn.metrics import confusion_matrix

# Matriz de confusión del modelo SVM
lr_confusion = confusion_matrix(y_test, y_prediccion)

# Visualización de la matriz de confusión
plt.figure(figsize=(8, 6))

sns.heatmap(
    lr_confusion,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=['High Traffic', 'Low Traffic'],
    yticklabels=['High Traffic', 'Low Traffic']
)

plt.title("Matriz de Confusión - Regresión Logística")
plt.xlabel("Predicción")
plt.ylabel("Valor Real")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(

Parámetros del mejor estimador: {'memory': None, 'steps': [('scaler',
StandardScaler()), ('selector', SelectKBest(k='all')), ('classifier',
LogisticRegression(C=10, penalty='l1', solver='liblinear'))], 'verbose': False,
'scaler': StandardScaler(), 'selector': SelectKBest(k='all'), 'classifier':
LogisticRegression(C=10, penalty='l1', solver='liblinear'), 'scaler__copy':
True, 'scaler__with_mean': True, 'scaler__with_std': True, 'selector__k': 'all',
'selector__score_func': <function f_classif at 0x7a726f5dbd90>, 'classifier__C':
10, 'classifier__class_weight': None, 'classifier__dual': False,
'classifier__fit_intercept': True, 'classifier__intercept_scaling': 1,
'classifier__l1_ratio': None, 'classifier__max_iter': 100,
'classifier__multi_class': 'auto', 'classifier__n_jobs': None,
'classifier__penalty': 'l1', 'classifier__random_state': None,
'classifier__solver': 'liblinear', 'classifier__tol': 0.0001,
'classifier__verbose': 0, 'classifier__warm_start': False}
Resultados de la Regresión Logística:
Accuracy: 0.7315789473684211
F1: 0.7951807228915663
Precision: 0.7857142857142857
```

```
Recall: 0.8048780487804879
```





###**Explicación**

####**Gráfico de barras**

El gráfico de barras representa las puntuaciones (scores en el argot) de las características (features en el argot) utilizadas en el modelo de Regresión Logística.

Al entrenar un modelo de machine learning, como en el caso que nos ocupa de la Regresión Logística, es fundamental entender cuál es el peso de cada característica o variable en la probabilidad del modelo para realizar predicciones correctas y precisas. Cada característica influirá en la capacidad de predicción del modelo, y algunas características tendrán más peso que otras.

En nuestro contexto, nuestro objetivo es predecir si una receta generará un tráfico alto o bajo (en una web) y, para ello, se han tenido en cuenta diversas características o variables relacionadas con las recetas que las definen, como las calorías, las proteínas, el azúcar, etc.

Cuando entrenamos el modelo, se acaban obteniendo puntuaciones (scores) para cada característica (feature), y esto nos permite obtener cuánto contribuye cada característica a las predicciones del modelo. Las características con puntuaciones (scores) más altas son más importantes puesto que tienen un mayor impacto en las predicciones del modelo.

El modelo busca identificar patrones en las características de las recetas que estén asociados con un tráfico alto, es decir, lo que buscamos es cuantificar qué recetas son populares en función de la cantidad de visitantes que la consultan en la web, y que características componen esta receta. Por otro lado, también buscamos que recetas tienen un tráfico bajo o nulo, que evidentemente, serán recetas que consideraremos que no son populares ya que atraen menos visitantes.

En resumen, cuanto mayor sea la puntuación (Eje Y) de una característica (Eje X) en el gráfico, más influencia tiene en las predicciones del modelo y, por lo tanto, es más relevante para determinar si una receta generará tráfico alto o bajo. Esta visualización ayuda a comprender qué características son las más determinantes en la clasificación de las recetas según su popularidad.

####**Matriz de Confusión**

Vamos a explicar que es lo que nos indican los valores obtenidos en la matriz de confusión:

- **40 Verdaderos Positivos (True Positives - TP)**: Estos son casos en los que el modelo ha predicho correctamente un valor positivo (high_traffic = 1) y su valor real era también positivo. En otras palabras, el modelo identificó correctamente 40 casos.
- **99 Verdaderos Negativos (True Negatives - TN)**: Estos son casos en los que el modelo ha predicho correctamente el valor negativo (high_traffic = 0) y el valor real era negativo (high_traffic = 0). El modelo identificó correctamente 99 casos.
- **27 Falsos Positivos (False Positives - FP)**: Estos son casos en los que el modelo ha predicho incorrectamente un valor positivo (high_traffic = 1) cuando el valor real era negativo (high_traffic = 0). El modelo cometió 27 errores al clasificar estos casos.
- **24 Falsos Negativos (False Negatives - FN)**: Estos son casos en los que el modelo ha predicho incorrectamente un valor negativo (high_traffic = 0) cuando el valor real era positivo (high_traffic = 1). El modelo cometió 24 errores al clasificar estos casos.

En resumen, estos valores indican cómo se ha comportado el modelo de regresión logística que hemos implementado para realizar una clasificación binaria. El modelo identificó correctamente 40 casos como positivos, 99 casos como negativos, cometió 27 errores al clasificar negativos como positivos y cometió 24 errores al clasificar positivos como negativos. Estos valores son esenciales para calcular métricas como precisión, recall, exactitud y F1-score, que proporcionan una visión más completa del rendimiento del modelo.

1.8 Evaluando el modelo y sus resultados

Vemos que modelo de Regresión Logística ha tenido buenos resultados.

Vemos algo más a fondo estos resultados:

- **Accuracy**: el modelo de Regresión Logística ha obtenido alrededor de un 73% de precisión (accuracy). Debemos tener en cuenta que estamos priorizando la precisión sobre la exactitud.
- **Precision**: el modelo de Regresión Logística obtuvo cerca del 79%.
- **Recall**: el modelo de Regresión Logística ha obtenido alrededor del 80%.
- **F1**: al ser la media entre las métricas precision y recall obtiene un valor un poco superior al 80%.

####**Mostramos gráficos con las variables categóricas**

Ahora analizo las categorías de las recetas con tráfico alto ('high_traffic') y visualizo con la ayuda de un barplot/gráfico de barras la cantidad de recetas que pertenecen a cada categoría ('category_') y después a cada 'servings_'.

Para ello creamos un nuevo DataFrame: df_categoricas. Este contiene la información sobre las diferentes variables de tipo 'category_'. Asignamos a este DataFrame los datos de las recetas que tenemos con tráfico alto (my_high_traffic_recipes) y se filtra por 'category' para poder crear una gráfica con sólo las variables categóricas que referencian a 'category_'.

A continuación, aplicamos el método .sum() a este DataFrame filtrado que nos permite obtener el total de la suma del número de recetas con tráfico alto por cada categoría.

El resultado anterior es un objeto Serie de Pandas que ordenamos de manera descendente (de mayor a menor) según el número de recetas para cada categoría así veremos en orden de mayor a menor las categorías, primero la que tiene la suma total más alta de recetas hasta la última que tiene la suma menor de recetas por esa categoría.

Renombramos la serie a 'Count' para que el eje Y sea más descriptivo. Utilizamos el parámetro inplace=True para actualizar el DataFrame df_categoricas directamente.

Finalmente, utilizamos Seaborn (sns) para crear el gráfico de barras. En el eje X (x), constan las categorías (los índices del DataFrame df_categoricas), y en el eje Y (y), tenemos el total de recetas para cada categoría.

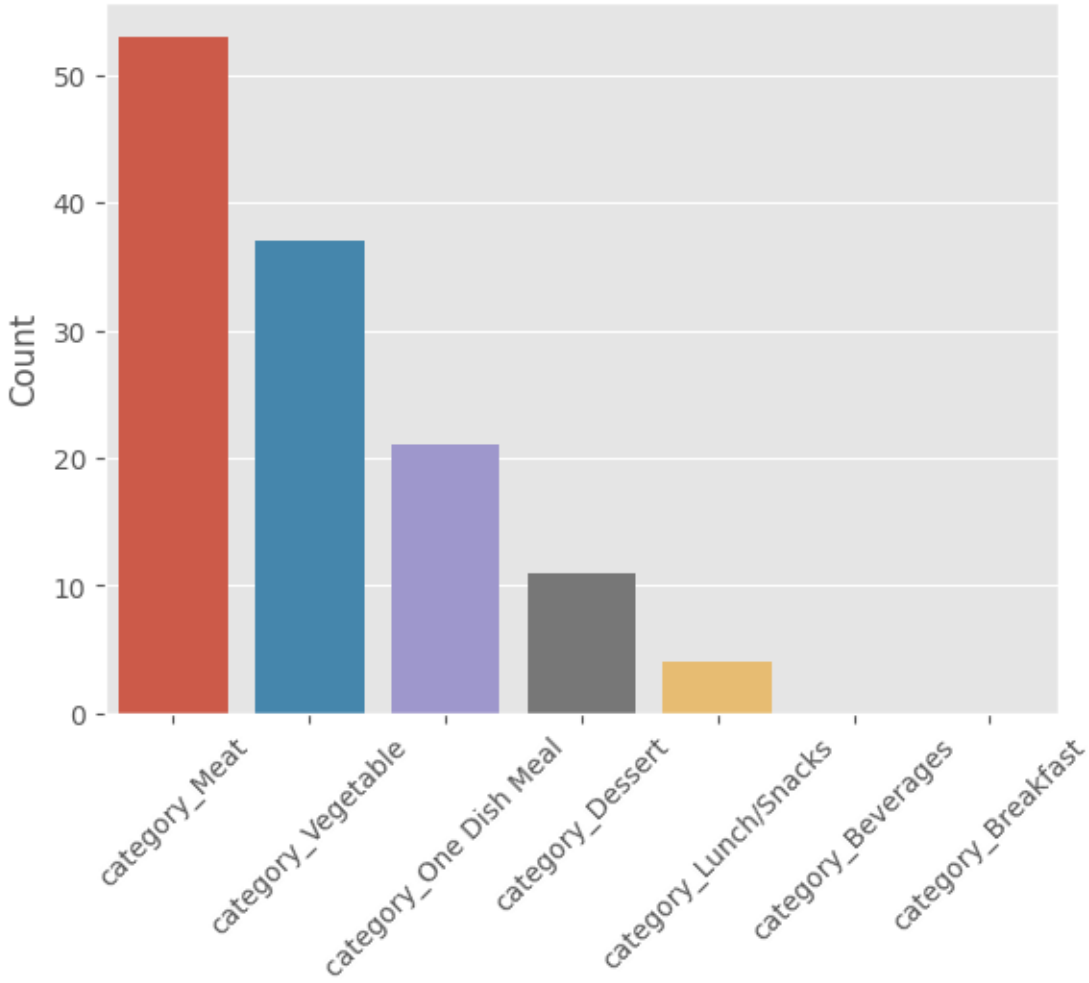
El resultado es un gráfico de barras que muestra las categorías en el eje X y el número de recetas en cada categoría en el eje Y. Esto permite visualizar cuáles son las categorías más comunes entre las recetas con tráfico alto.

Usamos el mismo procedimiento para poder mostrar una gráfica de barras para el caso de las variables categóricas 'servings_'.

[32]:
```
# Columna/Variable categóricas: 'category_'
df_categoricas = pd.DataFrame(
                    my_high_traffic_recipes.filter(like='category')
                                           .sum()
                                           .sort_values(ascending=False)
                )

df_categoricas.rename(columns={0:'Count'}, inplace=True)

# Mostramos la media de las categorías
sns.barplot(x=df_categoricas.index, y='Count', data=df_categoricas)
plt.xticks(rotation=45)
```

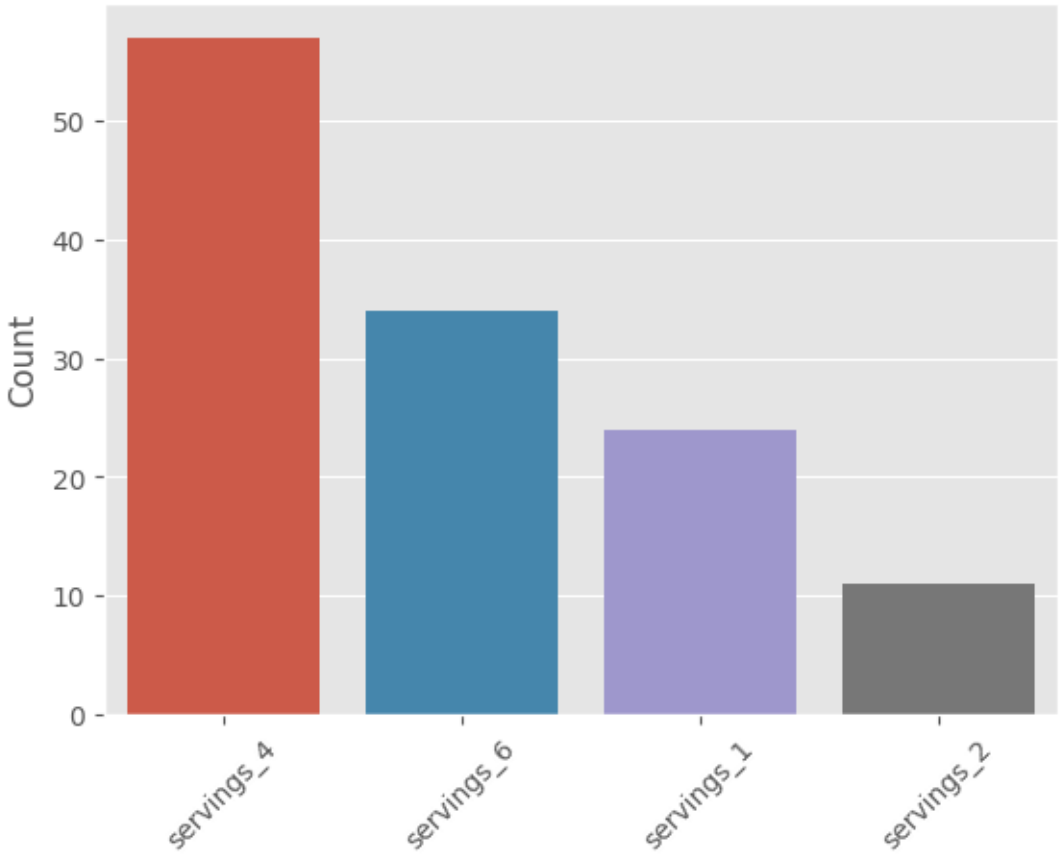
[32]:
```
(array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, 'category_Meat'),
  Text(1, 0, 'category_Vegetable'),
  Text(2, 0, 'category_One Dish Meal'),
  Text(3, 0, 'category_Dessert'),
  Text(4, 0, 'category_Lunch/Snacks'),
  Text(5, 0, 'category_Beverages'),
```

```
Text(6, 0, 'category_Breakfast')])
```



[33]:
```
# Columna/Variable categórica: 'servings_'
df_categoricas = pd.DataFrame(
                    my_high_traffic_recipes.filter(like='servings')
                                           .sum()
                                           .sort_values(ascending=False)
                )

df_categoricas.rename(columns={0:'Count'}, inplace=True)

# Mostramos la media
sns.barplot(x=df_categoricas.index, y='Count', data=df_categoricas)
plt.xticks(rotation=45)
```

[33]:
```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'servings_4'),
  Text(1, 0, 'servings_6'),
  Text(2, 0, 'servings_1'),
  Text(3, 0, 'servings_2')])
```



[34]:
```
# Mostramos las primeras 15 filas que contienen recetas con un alto␣
↪'high_traffic'
my_high_traffic_recipes.head(15)
```

[34]:

	calories	carbohydrate	sugar	protein	category_Beverages	\
191	0.763438	0.026470	1.786144	0.293981	0.0	
142	-0.878671	0.169324	1.415928	-1.183086	0.0	
702	0.535543	-1.125059	0.846604	0.800821	0.0	
242	1.839798	1.162527	-0.886894	0.955123	0.0	
710	-0.306516	-1.750015	0.016751	0.217193	0.0	
428	1.923837	1.507962	-0.392515	0.208507	0.0	
779	-1.890301	-0.536137	0.462115	0.646005	0.0	
762	0.209599	-1.342623	1.201602	0.602817	0.0	

```
582  0.390667     -0.075501 -0.043824 -0.485952                0.0
753  1.620401      0.114213  1.493760  1.547120                0.0
369 -0.338156     -0.691949  0.542787 -0.065480                0.0
671  0.754111     -0.758064 -0.644094  1.176514                0.0
123 -1.390774     -0.422936  0.819712  1.380399                0.0
293  0.297706      0.067859 -1.138896  0.400248                0.0
69   1.002609      0.066721 -0.236052  0.415723                0.0

     category_Breakfast  category_Dessert  category_Lunch/Snacks  \
191                 0.0               1.0                    0.0
142                 0.0               1.0                    0.0
702                 0.0               0.0                    0.0
242                 0.0               0.0                    0.0
710                 0.0               0.0                    0.0
428                 0.0               0.0                    0.0
779                 0.0               0.0                    0.0
762                 0.0               0.0                    0.0
582                 0.0               0.0                    0.0
753                 0.0               0.0                    0.0
369                 0.0               0.0                    0.0
671                 0.0               0.0                    0.0
123                 0.0               0.0                    0.0
293                 0.0               0.0                    0.0
69                  0.0               0.0                    0.0

     category_Meat  category_One Dish Meal  category_Vegetable  servings_1  \
191            0.0                     0.0                 0.0         0.0
142            0.0                     0.0                 0.0         0.0
702            1.0                     0.0                 0.0         0.0
242            1.0                     0.0                 0.0         0.0
710            0.0                     1.0                 0.0         0.0
428            1.0                     0.0                 0.0         1.0
779            1.0                     0.0                 0.0         0.0
762            0.0                     1.0                 0.0         1.0
582            0.0                     0.0                 1.0         0.0
753            1.0                     0.0                 0.0         0.0
369            1.0                     0.0                 0.0         0.0
671            1.0                     0.0                 0.0         0.0
123            1.0                     0.0                 0.0         1.0
293            1.0                     0.0                 0.0         0.0
69             0.0                     1.0                 0.0         1.0

     servings_2  servings_4  servings_6
191         0.0         0.0         1.0
142         0.0         1.0         0.0
702         0.0         0.0         1.0
242         0.0         1.0         0.0
```

```
710         0.0         0.0         1.0
428         0.0         0.0         0.0
779         0.0         0.0         1.0
762         0.0         0.0         0.0
582         0.0         0.0         1.0
753         0.0         1.0         0.0
369         0.0         0.0         1.0
671         0.0         1.0         0.0
123         0.0         0.0         0.0
293         0.0         0.0         1.0
69          0.0         0.0         0.0
```

###Modelo SVM

El Support Vector Machine (Máquina de Vectores de Soporte) es un algoritmo de Machine Learning utilizado para tareas de clasificación y regresión. En el contexto de clasificación, SVM se utiliza para separar dos clases distintas de datos, lo que lo convierte en una herramienta eficaz para la clasificación binaria. Además, SVM ofrece un alto grado de precisión y es una excelente opción cuando la precisión es una prioridad.

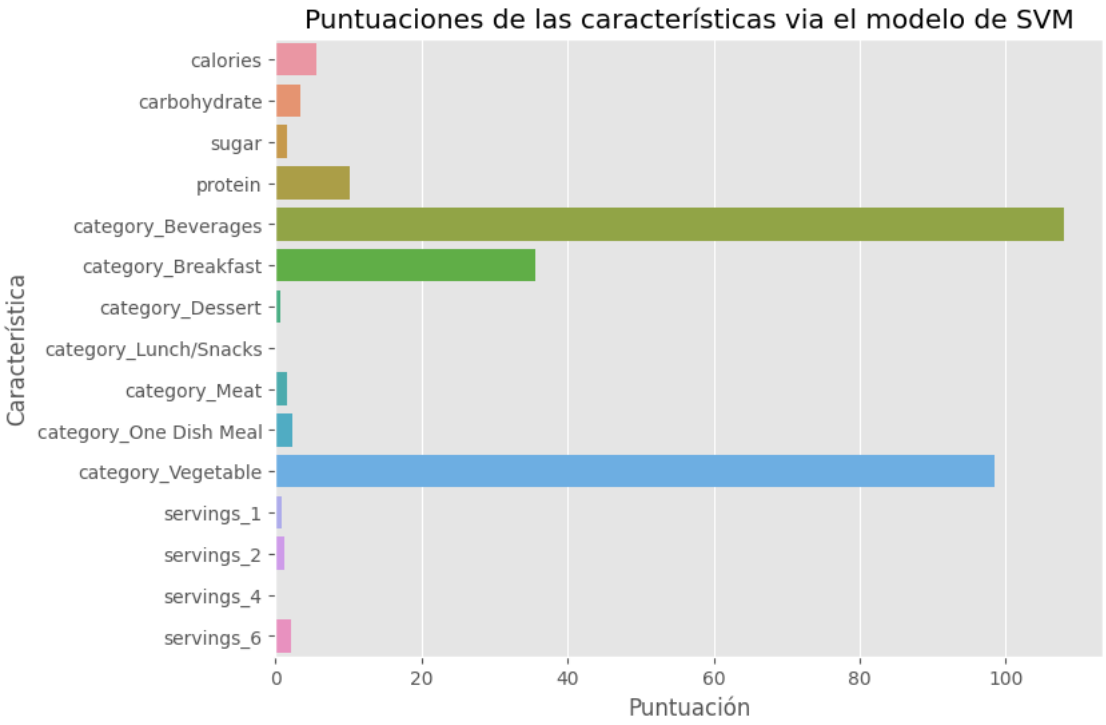
Utilizaré SVM para clasificar recetas en función de su tráfico, es decir, para predecir si una receta generará un tráfico alto o bajo.

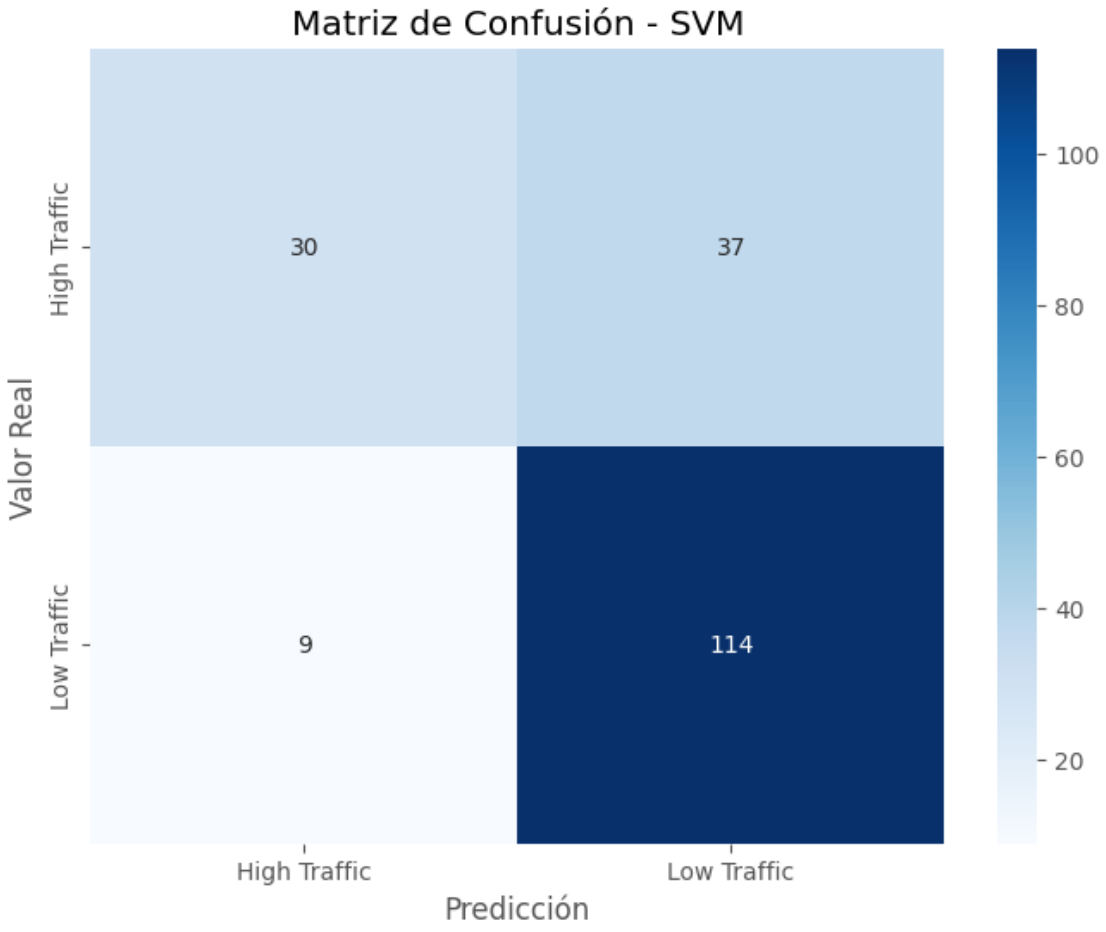
[35]:
```
from sklearn.svm import SVC

# Definir un flujo de trabajo para SVM
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('selector', SelectKBest(f_classif)),
    ('classifier', SVC(probability=True))
])

# Definir parámetros para búsqueda en cuadrícula
svm_parameters = {
    'selector__k': ['all'],
    'classifier__C': [0.1, 1, 10],
    'classifier__kernel': ['linear', 'rbf'],
}

# Instanciar GridSearchCV
svm_grid_search = GridSearchCV(svm_pipeline, svm_parameters, cv=5,␣
 ↪scoring='precision')

# Entrenar el modelo SVM
svm_grid_search.fit(X_train, y_train)

# Evaluar el modelo SVM en el conjunto de prueba
svm_best_estimator = svm_grid_search.best_estimator_
```

```
svm_y_pred_prob = svm_best_estimator.predict_proba(X_test)[:, 1]
my_threshold = 0.55
svm_y_pred = (svm_y_pred_prob >= my_threshold).astype(int)

# Calcular métricas para el modelo SVM
svm_accuracy = accuracy_score(y_test, svm_y_pred)
svm_f1 = f1_score(y_test, svm_y_pred)
svm_precision = precision_score(y_test, svm_y_pred)
svm_recall = recall_score(y_test, svm_y_pred)

# Mostrar resultados del modelo SVM
print('Resultados del modelo SVM:')
print("Accuracy:", svm_accuracy)
print("F1:", svm_f1)
print("Precision:", svm_precision)
print("Recall:", svm_recall)

# Índices de las recetas con tráfico alto respecto el conjunto de test
high_traffic_inds = np.where(svm_y_pred == 1)[0]

# Recetas con un tráfico alto
high_traffic_recs = X_test.iloc[high_traffic_inds]

# Vamos a graficar las puntuaciones respecto a las características seleccionadas
selector_s = svm_best_estimator.named_steps['selector']
selected_inds = selector_s.get_support(indices=True)
selected_scores_s = selector_s.scores_[selected_inds]
selected_features_s = X.columns[selected_inds]

# Mostramos un gráfico de tipo bar plot con las puntuaciones respecto a las␣
 ↪características estudidadas
plt.figure(figsize=(8, 6))
sns.barplot(x=selected_scores_s, y=selected_features_s)
plt.title('Puntuaciones de las características via el modelo de SVM')
plt.xlabel('Puntuación')
plt.ylabel('Característica')
plt.show()

# Matriz de confusión del modelo SVM
svm_confusion = confusion_matrix(y_test, svm_y_pred)

# Visualización de la matriz de confusión
plt.figure(figsize=(8, 6))

sns.heatmap(
    svm_confusion,
    annot=True,
```

```
    fmt="d",
    cmap="Blues",
    xticklabels=['High Traffic', 'Low Traffic'],
    yticklabels=['High Traffic', 'Low Traffic']
)

plt.title("Matriz de Confusión - SVM")
plt.xlabel("Predicción")
plt.ylabel("Valor Real")
plt.show()
```

```
Resultados del modelo SVM:
Accuracy: 0.7578947368421053
F1: 0.8321167883211679
Precision: 0.7549668874172185
Recall: 0.926829268292683
```





###**Explicación**

####**Gráfico de barras**

La explicación es similar a la que hemos dado para la Regresión Logística pero en el contexto del modelo SVM.

####**Matriz de Confusión**

Vamos a explicar que es lo que nos indican los valores obtenidos en la matriz de confusión:

- **30 Verdaderos Positivos (True Positives - TP)**: Estos son casos en los que el modelo ha predicho correctamente un valor positivo (high_traffic = 1) y su valor real era también positivo. En otras palabras, el modelo identificó correctamente 30 casos.
- **114 Verdaderos Negativos (True Negatives - TN)**: Estos son casos en los que el modelo ha predicho correctamente el valor negativo (high_traffic = 0) y el valor real era negativo (high_traffic = 0). El modelo identificó correctamente 114 casos.
- **37 Falsos Positivos (False Positives - FP)**: Estos son casos en los que el modelo ha predicho incorrectamente un valor positivo (high_traffic = 1) cuando el valor real era negativo (high_traffic = 0). El modelo cometió 37 errores al clasificar estos casos.

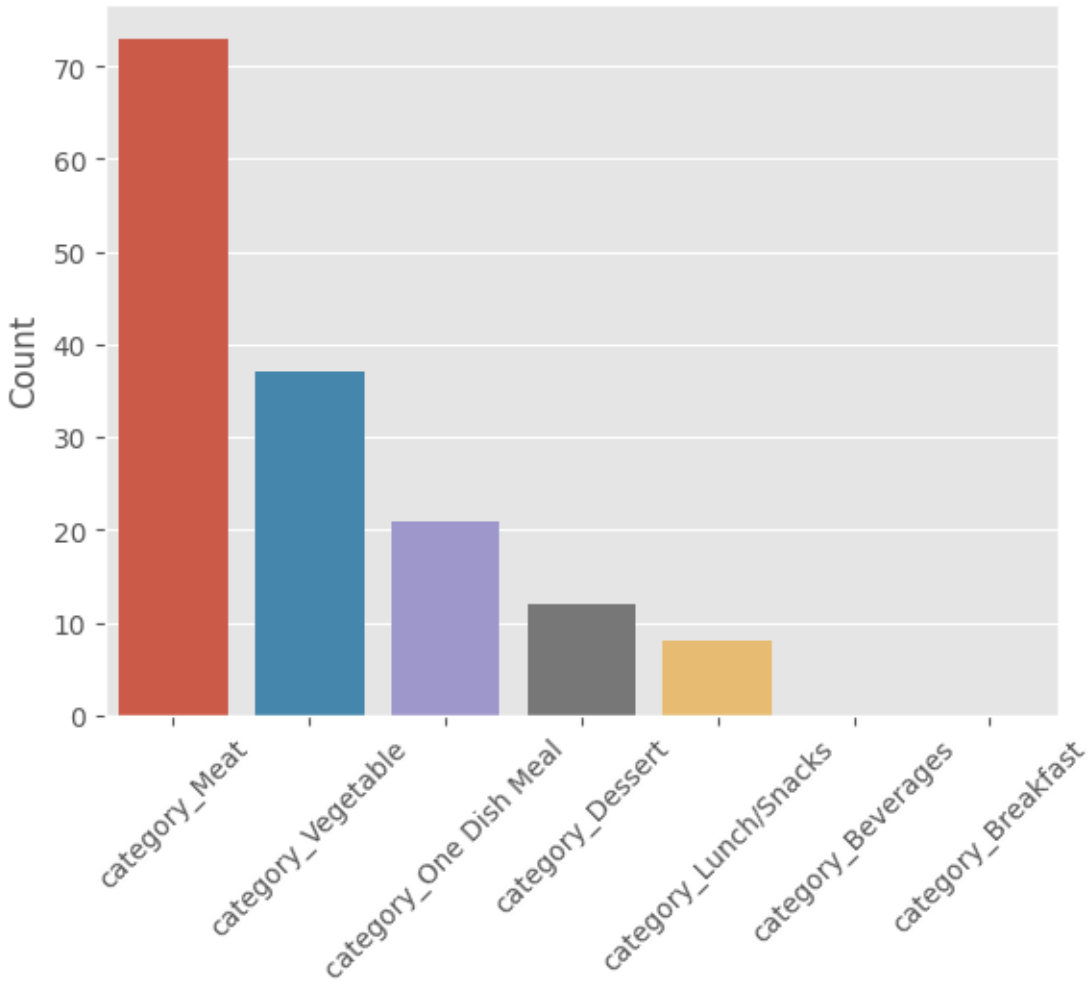
- **9 Falsos Negativos (False Negatives - FN)**: Estos son casos en los que el modelo ha predicho incorrectamente un valor negativo (high_traffic = 0) cuando el valor real era positivo (high_traffic = 1). El modelo cometió 9 errores al clasificar estos casos.

En resumen, estos valores indican cómo se ha comportado el modelo de SVM que hemos implementado para realizar una clasificación binaria. El modelo identificó correctamente 30 casos como positivos, 114 casos como negativos, cometió 37 errores al clasificar negativos como positivos y cometió 9 errores al clasificar positivos como negativos. Estos valores son esenciales para calcular métricas como precisión, recall, exactitud y F1-score, que proporcionan una visión más completa del rendimiento del modelo.

[35]:

[36]:
```
# Columna categórica de 'category_'
df_category = pd.DataFrame(high_traffic_recs.filter(like='category').sum().
 ↪sort_values(ascending=False))
df_category.rename(columns={0:'Count'}, inplace=True)

# Mostramos la gráfica de la media
sns.barplot(x=df_category.index, y='Count', data=df_category)
plt.xticks(rotation=45)
```

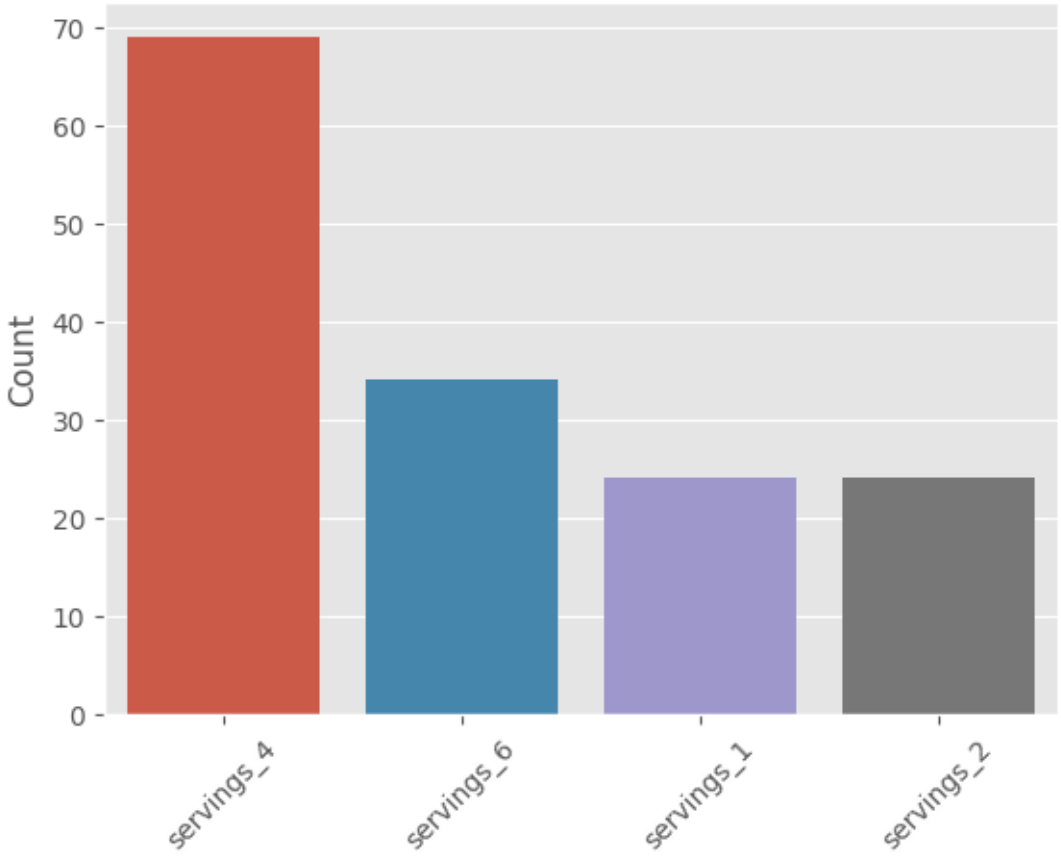
[36]:
```
(array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, 'category_Meat'),
  Text(1, 0, 'category_Vegetable'),
  Text(2, 0, 'category_One Dish Meal'),
  Text(3, 0, 'category_Dessert'),
  Text(4, 0, 'category_Lunch/Snacks'),
  Text(5, 0, 'category_Beverages'),
  Text(6, 0, 'category_Breakfast')])
```



[36]:

[37]:
```
# Columna categórica de 'servings_'
df_category = pd.DataFrame(high_traffic_recs.filter(like='servings').sum().
 ↪sort_values(ascending=False))
df_category.rename(columns={0:'Count'}, inplace=True)

# Mostramos la gráfica de la media
sns.barplot(x=df_category.index, y='Count', data=df_category)
plt.xticks(rotation=45)
```

[37]:
```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'servings_4'),
  Text(1, 0, 'servings_6'),
  Text(2, 0, 'servings_1'),
  Text(3, 0, 'servings_2')])
```



###**Comparación de los dos modelos, Regresión Logística vs SVC**

[38]:
```
# Mostramos los resultados de la Regresión Logística
print('Resultados de la Regresión Logística:')
print("Accuracy:", accuracy)
print("F1:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Resultados de la Regresión Logística:
Accuracy: 0.7315789473684211
F1: 0.7951807228915663
Precision: 0.7857142857142857
Recall: 0.8048780487804879
```

[39]:
```
# Mostrar resultados del modelo SVM
print('Resultados del modelo SVM:')
print("Accuracy:", svm_accuracy)
print("F1:", svm_f1)
print("Precision:", svm_precision)
```

```
print("Recall:", svm_recall)
```

```
Resultados del modelo SVM:
Accuracy: 0.7578947368421053
F1: 0.8321167883211679
Precision: 0.7549668874172185
Recall: 0.926829268292683
```

Resultados de la Regresión Logística:

- Accuracy: 0.7316
- F1: 0.7952
- Precision: 0.7857
- Recall: 0.8049

Resultados del modelo SVM:

- Accuracy: 0.7579
- F1: 0.8321
- Precision: 0.7550
- Recall: 0.9268

Veamos una comparativa de las diferentes métricas obtenidas según el modelo empleado:

1. **Accuracy (Exactitud)**: El modelo SVM tiene una accuracy ligeramente superior (0.7579) en comparación con la Regresión Logística (0.7316). Esto significa que el modelo SVM clasifica correctamente una mayor proporción de ejemplos en el conjunto de test.
2. **F1-Score**: El F1-score combina precisión y recall en una sola métrica. El modelo SVM tiene un F1-score más alto (0.8321) en comparación con la Regresión Logística (0.7952), lo que indica un mejor equilibrio entre precisión y recall para este modelo.
3. **Precision (Precisión)**: El modelo de Regresión Logística tiene una precisión ligeramente superior (0.7857) en comparación con el SVM (0.7550). Esto significa que cuando el modelo de Regresión Logística predice la clase positiva, tiende a ser más preciso.
4. **Recall (Sensibilidad)**: El modelo SVM tiene un recall significativamente más alto (0.9268) en comparación con la Regresión Logística (0.8049). Esto significa que el SVM es más efectivo en identificar correctamente todos los ejemplos del valor real positivo en comparación con la Regresión Logística.

En general, los resultados sugieren que el modelo SVM supera al modelo de Regresión Logística en términos de F1-score y recall, lo que indica que es mejor para identificar casos con valor positivo. Sin embargo, la Regresión Logística tiene una precisión ligeramente superior, lo que significa que cuando predice un valor positivo, tiende a ser más precisa que el modelo SVM. La elección entre estos dos modelos depende de si damos más peso a la métrica precisión respecto a la métrica recall, así como a la importancia relativa de los falsos positivos y falsos negativos en el contexto específico de nuestro caso de estudio.