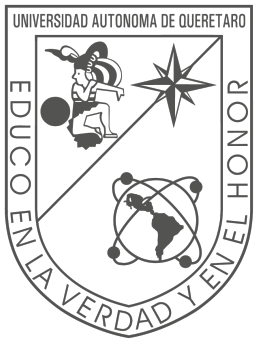
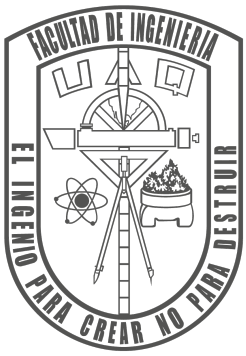
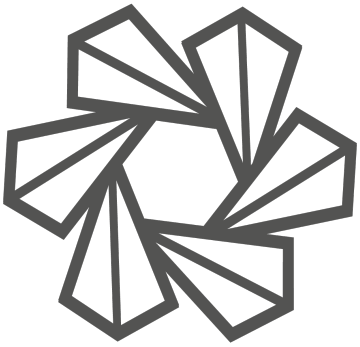
Universidad Autónoma de Querétaro

Facultad de Ingeniería
División de Investigación y Posgrado







Práctica 3

Normalización y generación de datos sintéticos

Maestría en Ciencias en Inteligencia Artificial
Optativa de especialidad IV - Machine Learning

Aldo Cervantes Marquez
Expediente: 262775
Profesor: Dr. Marco Antonio Aceves Fernández

Santiago de Querétaro, Querétaro, México
Semestre 2023-1
24 de Marzo de 2023

Índice

1. Objetivo

Esta practica tiene como principal objetivo el de realizar una normalización de datos tabulares a una base de datos, con el fin de observar los tipos de normalización que existen y en que momentos se deben utilizar. Además como segunda parte de la práctica, se realizará un balance de clases dentro de la base de datos mediante la generación de datos sintéticos con el fin de tener una mejor calidad en los datos para futuras aplicaciones en aprendizaje automático.

2. Introducción

La presente práctica se divide en 2 secciones. La primera consiste en realizar la normalización de atributos y justificar, aplicando dos métodos de normalización, de entre los siguientes: min-max, z-score,u-score, por logaritmo, por frecuencia acumulada, sigmoide, unidad compleja, etc. Observando su distribución antes y después de la normalización.

En la segunda parte, se aplicará una técnica de generación de datos sintéticos con el fin de equilibrar clases desbalanceadas en un atributo categórico binario. En este caso se trabajará con el atributo llamado *Response"* el cual indica si el cliente compró mediante la campaña. Pues este atributo se encuentra muy desvalanceado (aproximadamente 90 % no compró).

3. Marco Teórico

Para la realización de la práctica fueron necesarios los siguientes conceptos, los cuales fueron obtenidos principalmente de [1].

3.1. Métodos de normalización

Para este tipo de casos, en datos continuos, se utilizaron los siguientes métodos [2, 3]: [3]

3.1.1. Normalización min-max

Esta normalización consiste en realizar un escalamiento entre valores definidos a, b y los datos de serán transformadas a ese rango, donde el valor mínimo estará en a y el máximo en b. Se calcula mediante la siguiente fórmula:

$$v_{norm} = a + \frac{(v_i - v_{min})(b - a)}{v_{max} - v_{min}} \tag{1}$$

3.1.2. Normalización z-score

Este método consiste en aproximar los valores a una función que explica la desviación de la media de los datos [3, 4], este método es muy útil cuando se tiene en consideración algunos otros factores

intrínsecos de los propios datos, se calcula mediante la siguiente formula:

$$v_{norm} = \frac{v_i - \bar{v}}{\sigma_v} \tag{2}$$

donde:

- $\bar{v}$ es la media de los datos
- σ_v es la desviación estándar de los datos

Es bastante útil como por ejemplo:

Se tienen calificaciones de las materias de educación fisica 4 5 5 6 7 8 7 7 6 6 7 5 9 1 0 8 6 7 8 3 8 7 y matemáticas 1 5 6 3 2 0 6 7 9 8 1 0 1 3 5 3 4 7 8 3 2 4. obteniendo su media y desviación estándar se obtuvo $\bar{v}_{ef} = 6.61$, $\sigma_{vef} = 1.65$ y $\bar{v}_m = 4.61$, $\sigma_v = 2.81$. Entonces al normalizar el mismo valor (5) se obtiene

$$z_{ef} = \frac{5-6.61}{1.65} = -1 \quad z_m = \frac{5-4.61}{2.81} = 0.01 \tag{3}$$

Como vemos ambas calificaciones no son equivalentes, pues mientras un 5 en la primera asignatura tiene por encima el 76 % de las calificaciones de la clase. En la otra asignatura es una medida muy cercana a la media de la clase.

Pudiendo concluir que es menos probable sacar un 5 en educación física que en matemáticas.

3.1.3. Normalización por sigmoide

Este método consiste en ajustar los valores al rango de una sigmoide (0,1), tomando en cuenta la media y la desviación estándar de los datos mediante la formula:

$$v_{norm} = \frac{1}{1 + e^{-\frac{v_i - \bar{v}}{\sigma_v}}} \tag{4}$$

3.2. Generación de datos sintéticos

El uso de métodos para generar datos sinteticos es bastante útil cuando se tienen clases desbalanceadas, pues se tiene bastante incertidumbre al momento de entrenar y probar modelos. Puede ser engañoso, pues la exactitud puede ser alta, pero hay otras métricas que nos muestran el rendimiento del modelo. Por lo que al aplicar el F1 score y recall, se observa un valor muy bajo, lo que muestra que existe un desbalance de clases y el modelo entrenado no es del todo robusto [5]

Se utilizará el método SMOTE, puesto que es el más adecuado para balancear clases en las que hay un claro desbalance mayor a 30 % [2, 5]. Sin embargo, este método es bastante flexible y se extrajeron los 2 métodos más adecuados para la aplicación requerida.

El método SMOTE (Synthetic Minority Over-sampling Technique) funciona mediante los siguientes pasos 1.

1. Identificar la clase minoritaria

2. Seleccionar una instancia aleatoria y medir distancias con otras instancias de la clase minoritaria (puede ser distancia euclidiana, Gower, Manhattan, etc).

3. Seleccionar otro punto segun sea el criterio (knn, aleatorio, etc).

4. Realizar una interpolación o extrapolación entre los dos puntos.

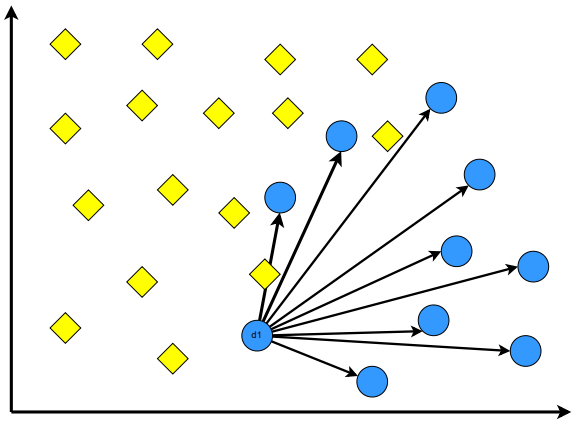


Figura 1: Funcionamiento de algoritmo SMOTE general.

3.2.1. SMOTE entre puntos aleatorios

El método selecciona n instancias aleatorias de la clase minoritaria (generalmente 2) y se realizan técnicas de interpolación (véase Figura 3), en este caso se propone utilizar una interpolación lineal con un valor aleatorio, obteniendo un punto aleatorio del trayecto de la distancia mediante la formula:

$$S_1 = p_1 + (P(x)(p_2 - p_1)) \tag{5}$$

donde $P(x) : \mathbb{R} \in [0, 1]$ en este caso una función uniforme.

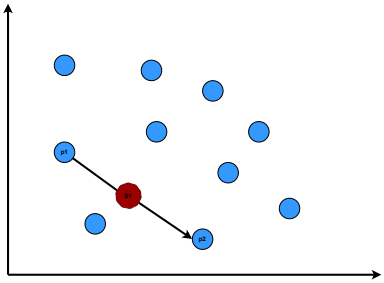


Figura 2: Algoritmo SMOTE para puntos aleatorios.

3.2.2. SMOTE basado en knn

Este método consiste en obtener un punto aleatorio y medir todas las distancias con respecto a los demás puntos, los n más cercanos, se separan y se elige uno de ellos al azar para realizar la interpolación de la ecuación (5).

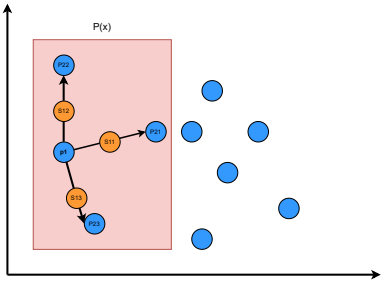


Figura 3: Algoritmo SMOTE para puntos aleatorios.

3.2.3. Métodos de distancia

Para obtener las distancias se suelen usar principalmente los siguientes métodos:

Distancia euclidiana: este método consiste en medir la distancia entre puntos de una manera recta.

$$|v| = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + \cdots + (z'_0 - z'_1)^2} \tag{6}$$

Distancia Gower: Este método es más utilizado cuando se tienen multiclases y datos mixtos en la base de datos (tanto categóricos como continuos). Por lo que se deberá acudir a [6] para poder obtener la demostración completa. Básicamente se obtienen las distancias mediante las siguientes formulas:

$$S(x_{il}(t), x_{jl}(t)) = \frac{\sum \delta_{ijl}(t) S_{ijl}(t)}{\sum \delta_{ijl}(t)} \tag{7}$$

donde $S_{ijl}(t)$ indica la similaridad para la l-ésima caracteristica entre dos objetos y $\delta_{ijl}(t) \in [0, 1]$ es un coeficiente basado en la posibilidad de que la medida de dos objetos se pierde en el tiempo t. Para variables discretas (incluyendo binarias), el componente de similitud se obtiene como:

$$S_{ijl}(t) = \begin{cases} 1 & si \quad x_{il}(t) = x_{jl}(t) \\ \\ 0 & si \quad x_{il}(t) \neq x_{jl}(t) \end{cases} \tag{8}$$

Para datos continuos:

$$S_{ijl}(t) = 1 - \frac{|x_{il}(t) - x_{jl}(t)|}{R_l(t)} \tag{9}$$

Donde $R_l(t)$ es el rango de la l-ésima variable en el tiempo t sobre todos los objetos escritos como $R_l(t) = \text{máx}_m\, X_{ml}(t) - \text{mín}_m\, X_{ml}(t)$. Correspondiente a la disimilaridad medida que puede ser obtenida o simplemente usando $D(x_i, x_j) = 1 - S(x_i, x_j)$

4. Materiales y Métodos

4.1. Materiales

4.1.1. Base de datos

La base de datos elegida para los datos no dependientes del tiempo fue obtenida de la página de kaggle, la cual consta de datos de clientes de una tienda departamental durante los años de 2017-2020, enfocando la información a 4 pilares principales:

1. Perfil del cliente.
2. Preferencias de productos.
3. Éxito/fracaso de campañas publicitarias.
4. Frecuencia de consumos.

Sin embargo en este caso solo se extrajo el atributo de ingreso, consumos por departamento y la respuesta ante las campañas publicitarias (2205 instancias).

4.1.2. Librerías y entorno de desarrollo

El análisis de los datos se llevará a cabo en el lenguaje de programación Python dentro del entorno de desarrollo de Jupyter Notebook. Ocupando las librerías matplotlib, seaborn, numpy y Pandas.

4.2. Metodología

La metodología consiste en la recopilación de las bases de datos, aplicar métodos de normalización, observar un desbalance de clases y aplicar los métodos basados en SMOTE (véase Figura 4).

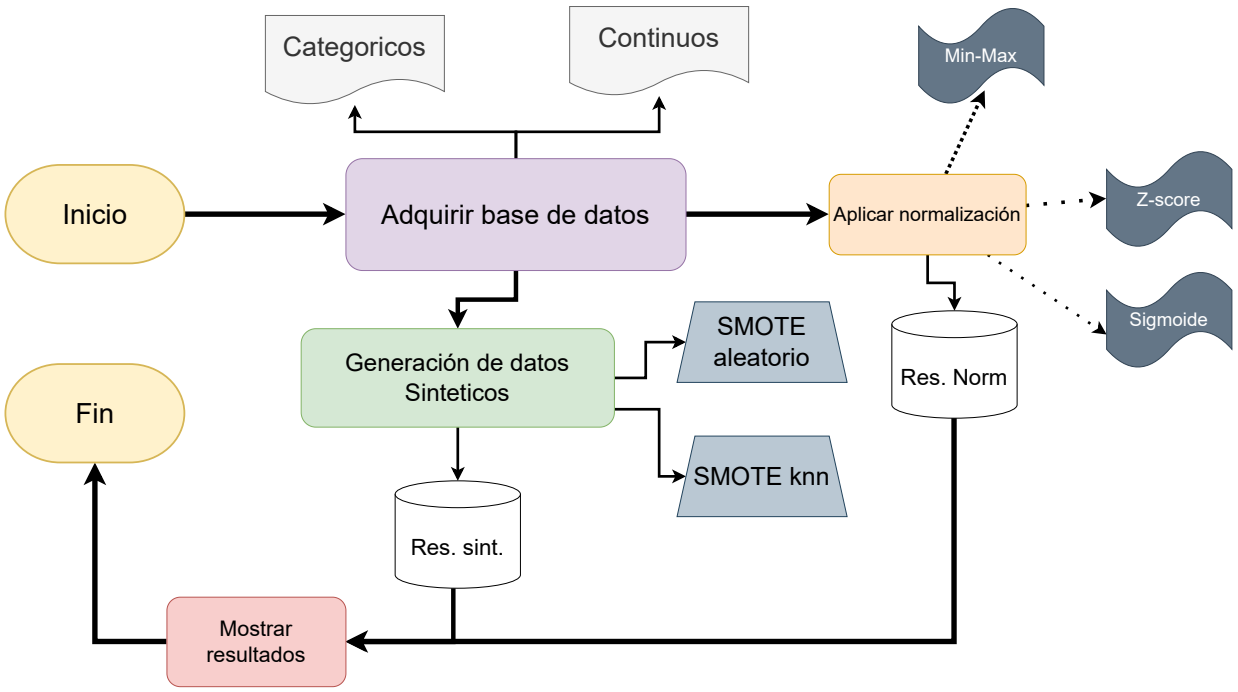


Figura 4: Metodología de la práctica.

5. Pseudocódigo

Algoritmo 1 Pseudocódigo normalización y generación de datos sinteticos.

Inicio
  funcion *SMOTE*(**DB**)
  funcion *min-max*(**DB**)
  funcion *z-score*(**DB**)
  funcion *sigmoid*(**DB**)
  Db
  Para método en *función*
    DBf$\leftarrow$ *funcion(normalización,dsint.)*
    Mostrar_resultados(***DBf***)
Fin

6. Resultados

6.1. Normalización

Se realizaron las normalizaciones del atributo de ingresos, el cual tiene una distribución bimodal y se obtuvieron los siguientes resultados en la distribución.

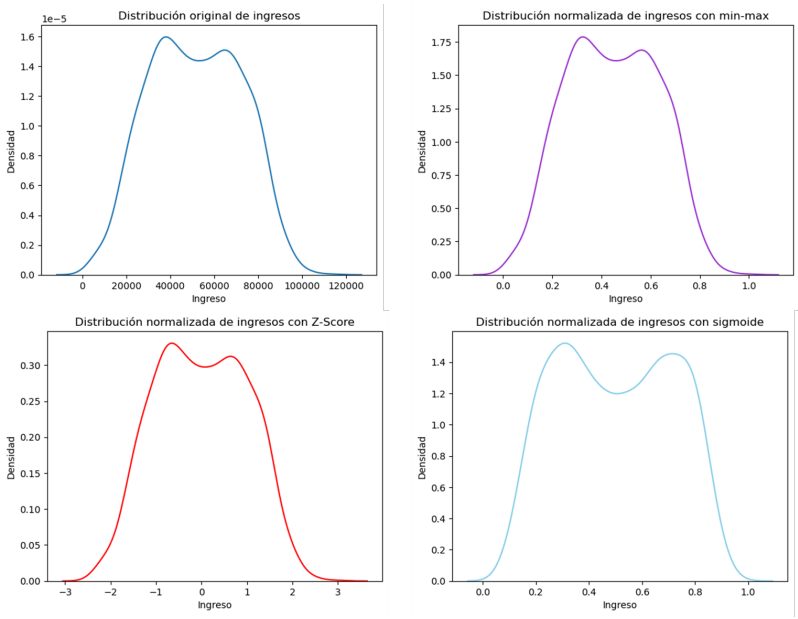


Figura 5: Resultados de la normalización de ingresos.

Como se observa en la 5 se conserva la forma de la distribución, únicamente en la distribución con la normalización con sigmoide se aplasta un poco, pero esto es más por la escala de valores que se tiene.

6.2. Generación de datos sintéticos

Se realizó una cantidad de 1539 datos sintéticos, a partir de 2205 instancias se tendrán 3744. Para determinar la cantidad de instancias a generar, se contó la cantidad instancias que contenían a cada clase (véase Figura 6). Se observaron 333 instancias que contenían a la categoría 1 (compraron en campaña) y 1872 instancias en clase 0 (no compraron en campaña), entonces la clase 1 es la minoritaria y se necesitan 1539 instancias de la clase minoritaria para equilibrarse con la mayoritaria.

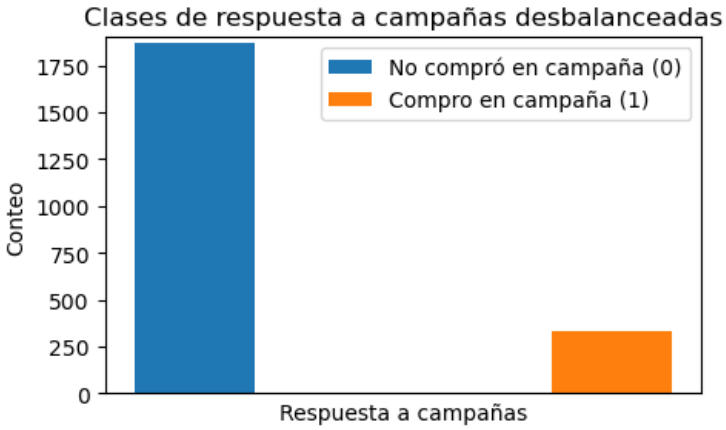


Figura 6: Frecuencia de clases original.

6.2.1. SMOTE aleatorio

Se realizó un balance de clases y se obtuvo la misma cantidad de instancias en cada clase (véase Figura 7)

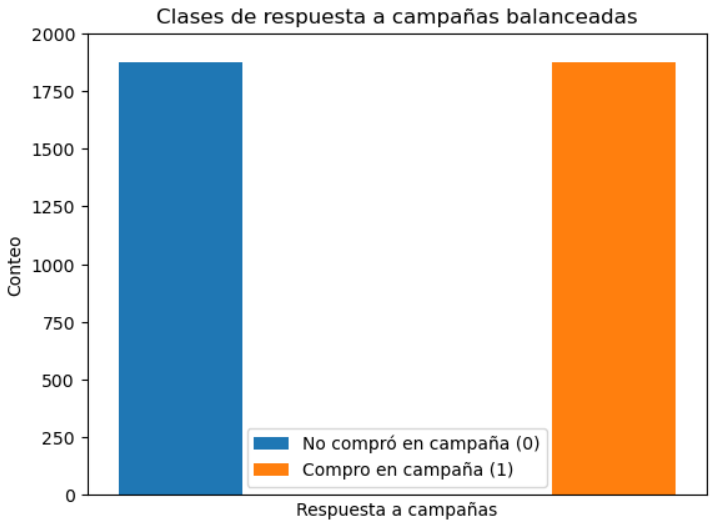


Figura 7: balance de clases en algoritmo SMOTE.

A continuación se observa las distribuciones de los demás datos generados (ventas por departamento, Figura 8) comparadas con las originales.
Cabe destacar que el tiempo de ejecución fue de 0.8070 segundos, lo cual fue bastante rápido para generar tantos datos.

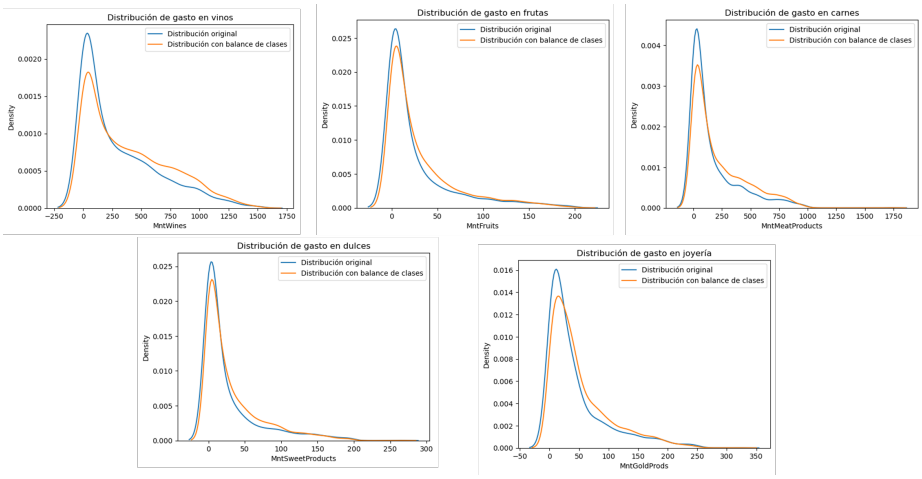


Figura 8: Distribución de los datos con SMOTE aleatorio.

Como se observa, las distribuciones no cambian de forma, únicamente se aplanan un poco, lo cual es normal, pues se anexan muchos datos a la base de datos.

6.3. SMOTE k=3

De igual manera que en el aleatorio, se obtuvo la misma cantidad de instancias en cada clase. Se encontraron las 3 instancias más cercanas y se eligió aleatoriamente una de ellas, para posteriormente hacer la interpolación. Obteniendo como resultado las siguientes distribuciones de los datos continuos (véase Figura 9).

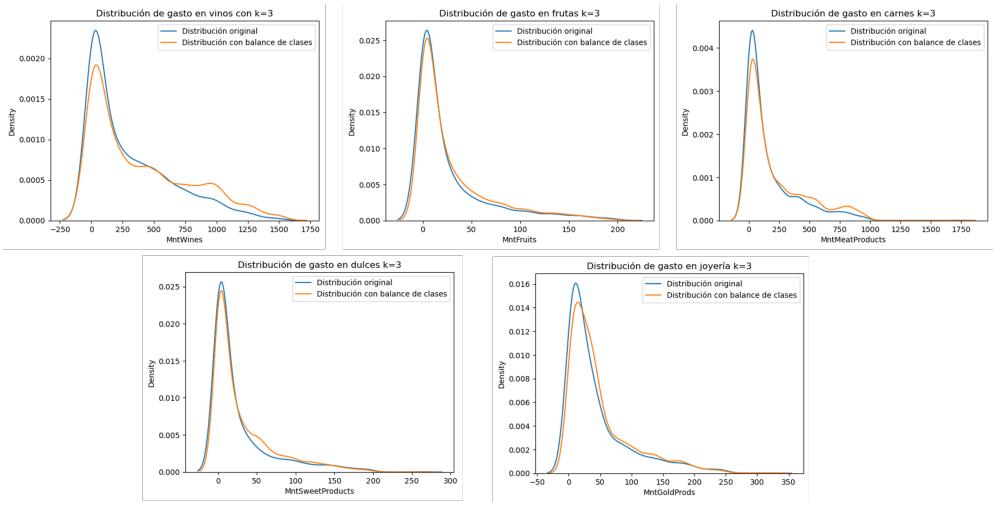


Figura 9: Distribución de los datos con SMOTE k=3.

Al igual que en el método de SMOTE aleatorio, fue posible mantener la distribución para generar todos los datos, observando que fue casi idéntico el resultado. Sin embargo, el tiempo de ejecución fue de 87.7040 segundos, muy tardado en comparación del método pasado.

6.4. SMOTE k=5

Este método al ser igual al anterior, únicamente variando la cantidad de vectores cercanos a 5, tiene los mismos resultados en cuanto a las instancias por clase, obteniendo las siguientes distribuciones en los datos continuos (véase Figura 10).

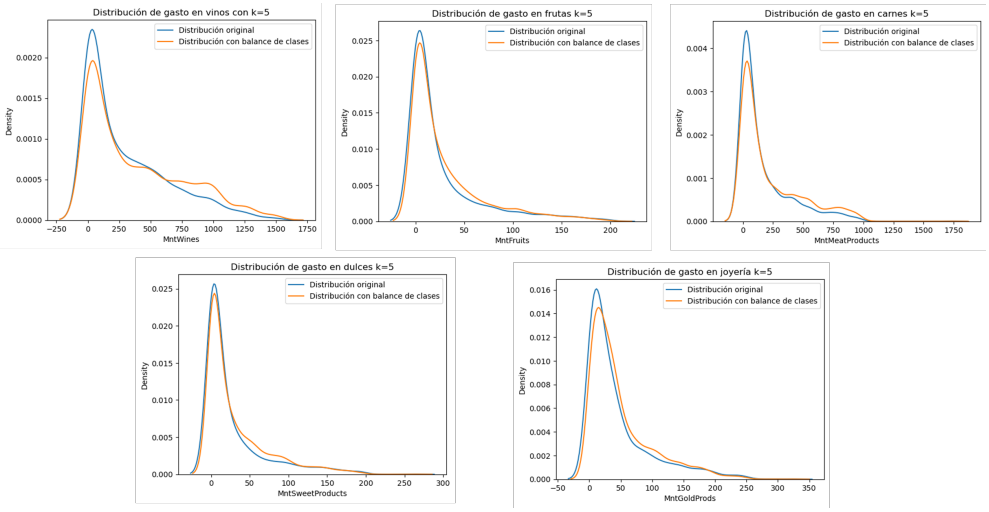


Figura 10: Distribución de los datos con SMOTE k=5.

La distribución es bastante similar a las anteriores, teniendo ligeras diferencias en algunas gráficas, pero manteniendo las formas de distribución originales. Cabe destacar que el tiempo de ejecución para este método fue de 92.5845 segundos, siendo el más tardado.

7. Conclusiones

La normalización de datos es una técnica bastante importante para poder cambiar el rango de operación de los datos, siendo de gran utilidad al momento de probar con modelos de Machine Learning, pues algunos únicamente aceptan valores entre cierto rango o forma, como es el caso del sigmoide o min-max, por lo que de esta manera es que se justifica el uso de normalización en el atributo de ingresos, tomando en cuenta que son datos con rangos muy altos y manejando ciertas unidades que tal vez no podrían servir en un análisis.

Por otro lado, se tiene la generación de datos sintéticos, que en este caso para el balance de clases, que es algo muy importante, pues al entrenar modelos con clases desbalanceadas, a simple vista no se observaría algún problema, sin embargo, se tiene que el modelo no podría aprender a generalizar con pocas instancias de una clase desbalanceada, observando problemas con otras métricas como recall o F1 score. El método de SMOTE es uno de los más utilizados para generar balance de clases cuando se tiene un desbalance tan marcado como este, teniendo 15 % de la clase minoritaria contra 85 % en la clase mayoritaria. Además de que este algoritmo ofrece una amplia variedad de modificaciones para obtener un mejor desempeño en la generación de datos sintéticos. Como es la función de interpolación, extrapolación, el método de medición de distancias, y el método de selección

de la instancia. Tomando en cuenta que se crearon 1539 datos (casi el 70 % de la base de datos total) se observa una modificación en la distribución muy ligera con respecto a la original. Además cabe mencionar que para esta aplicación, fue posible medir el tiempo de ejecución de cada método, siendo el más rápido el aleatorio, mostrando pocas diferencias entre cada variación, se puede considerar que el mejor a utilizar será dicho método.

Referencias

[1] M. Fernández, *Inteligencia Artificial para Programadores con Prisa*. Amazon Digital Services LLC - KDP Print US, 2021.

[2] S. Lasse, "Data augmentation for tabular data." https://medium.com/analytics-vidhya/data-augmentation-for-tabular-data-f75c94398c3e, November 2021. (Accessed on 03/16/2023).

[3] L. Al Shalabi and Z. Shaaban, "Normalization as a preprocessing engine for data mining and the approach of preference matrix," in *2006 International Conference on Dependability of Computer Systems*, pp. 207–214, 2006.

[4] D. SCIENCE, "¿qué es un z-score? — matemática y estadística." https://datascience.eu/es/matematica-y-estadistica/que-es-un-z-score/. (Accessed on 03/20/2023).

[5] M. Abinaya and S. Vedanth, "Data augmentation techniques for tabular data." https://www.mphasis.com/content/dam/mphasis-com/global/en/home/innovation/next-lab/Mphasis_Data-Augmentation-for-Tabular-Data_Whitepaper.pdf. (Accessed on 03/20/2023).

[6] Özlem Akay and G. Yüksel, "Clustering the mixed panel dataset using gower's distance and k-prototypes algorithms," *Communications in Statistics - Simulation and Computation*, vol. 47, no. 10, pp. 3031–3041, 2018.

P3 Aldo Cervantes Normalizacion y datos sinteticos

March 24, 2023

1 Normalización y generación de datos sintéticos

1.1 Normalización

está la normalización min-max la cúal se obtiene de la siguiente forma

$$v_{norm} = a + \frac{(v_i - v_{min})(b-a)}{v_{max} - v_{min}}$$

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

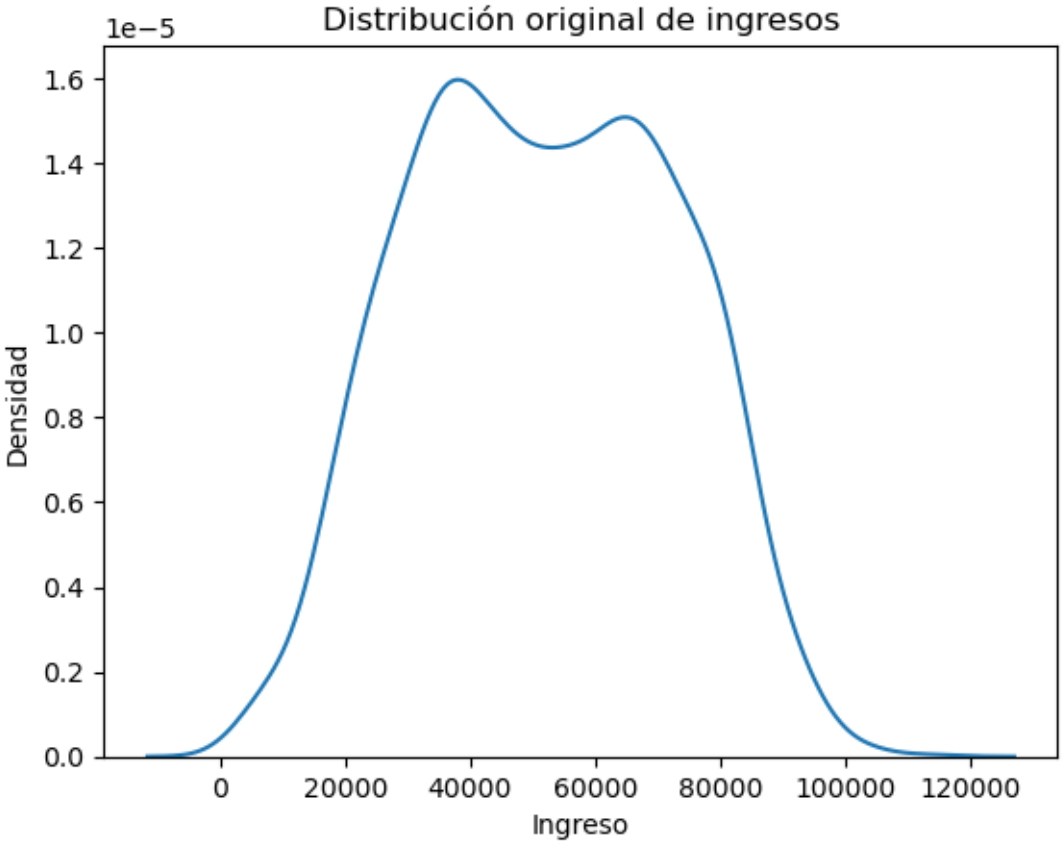
```
C:\Users\aldoa\anaconda3\envs\env2\lib\site-packages\scipy\__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[2]:
```
def norm_min_max(x,a,b): # Para todo el dataFrame
    l=list(x.columns)
    v_max=0
    v_min=0
    res=pd.DataFrame()
    for val in l:
        v_max=x[val].max()
        v_min=x[val].min()
        r_dt=v_max-v_min
        r_norm=b-a
        d=x[val]-v_min
        dpct=d/r_dt
        dnorm=r_norm*dpct
        data=a+dnorm
        aa=pd.DataFrame(data,columns=[val])
        res[val]=data
    return res
```

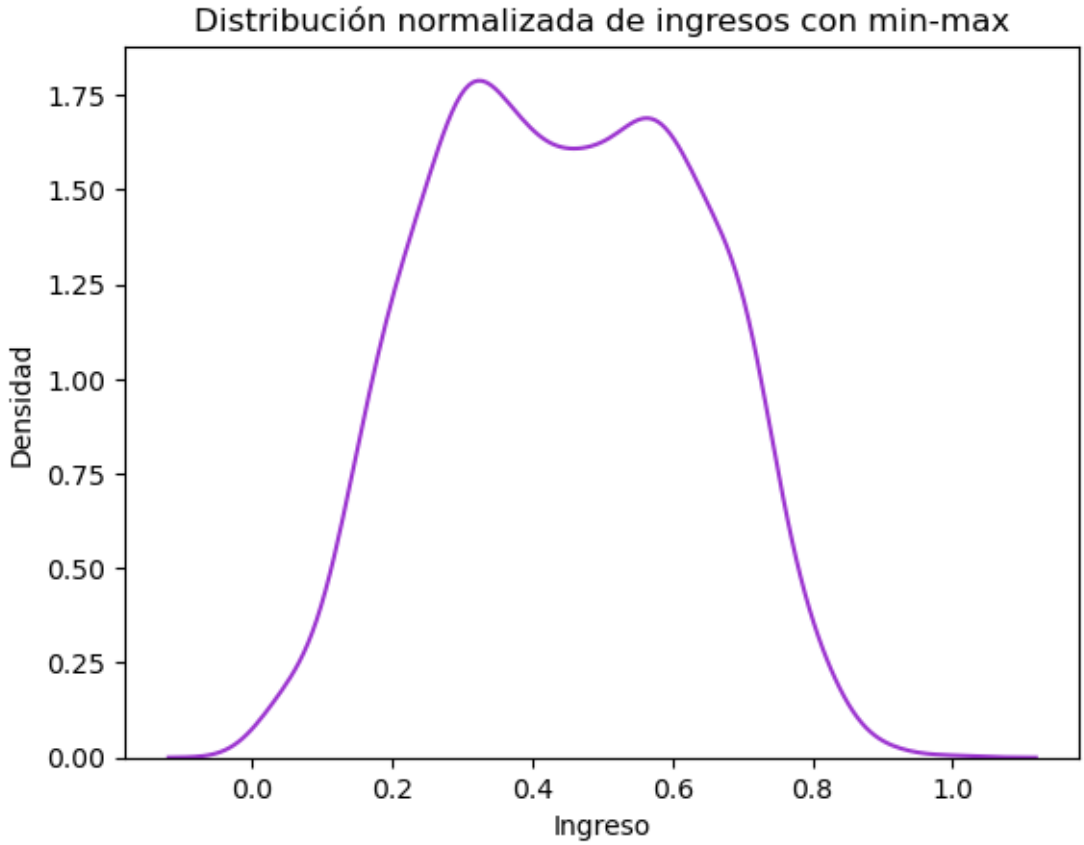
[3]:
```
database=pd.read_csv("C:\\Users\\aldoa\\Machine Learning\\Practica 1 datos tabulares\\Database_mejorada.csv")
database.columns
```

[3]:
```
Index(['Unnamed: 0', 'Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines',
       'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Response', 'Age', 'Customer_Days',
       'MntTotal', 'MntRegularProds', 'AcceptedCmpOverall', 'marital_Status',
       'education'],
      dtype='object')
```

[4]:
```
sns.kdeplot(database['Income'])
plt.title('Distribución original de ingresos')
plt.ylabel('Densidad')
plt.xlabel('Ingreso')
plt.show()
```



[5]:
```
norm1=norm_min_max(database[['Income']],0,1)
sns.kdeplot(norm1['Income'],color='#9C33D0')
plt.title('Distribución normalizada de ingresos con min-max')
plt.ylabel('Densidad')
plt.xlabel('Ingreso')
plt.show()
```



1.2 Normalización Z-score

Normaliza de una manera que los valores se encuentren en una media de 0 y una desviación estandar de 1. La fórmula es:

$$v_{norm} = \frac{v_i - \bar{v}}{\sigma_v}$$

donde: * $\bar{v} = \mu_v$ es la media de los datos * σ_v es la desviación estandar de los datos

Tiene la ventaja de poder expresar "consistencia":

Ejemplo: Tenemos dos series de notas correspondientes a dos asignaturas diferentes de un grupo de alumnos:

4 5 5 6 7 8 7 7 6 6 7 5 9 1 0 8 6 7 8 3 8 7

1 5 6 3 2 0 6 7 9 8 1 0 1 3 5 3 4 7 8 3 2 4

μ_1=6.61 , σ_1=1.65

μ_2=4.61 , σ_2=2.81

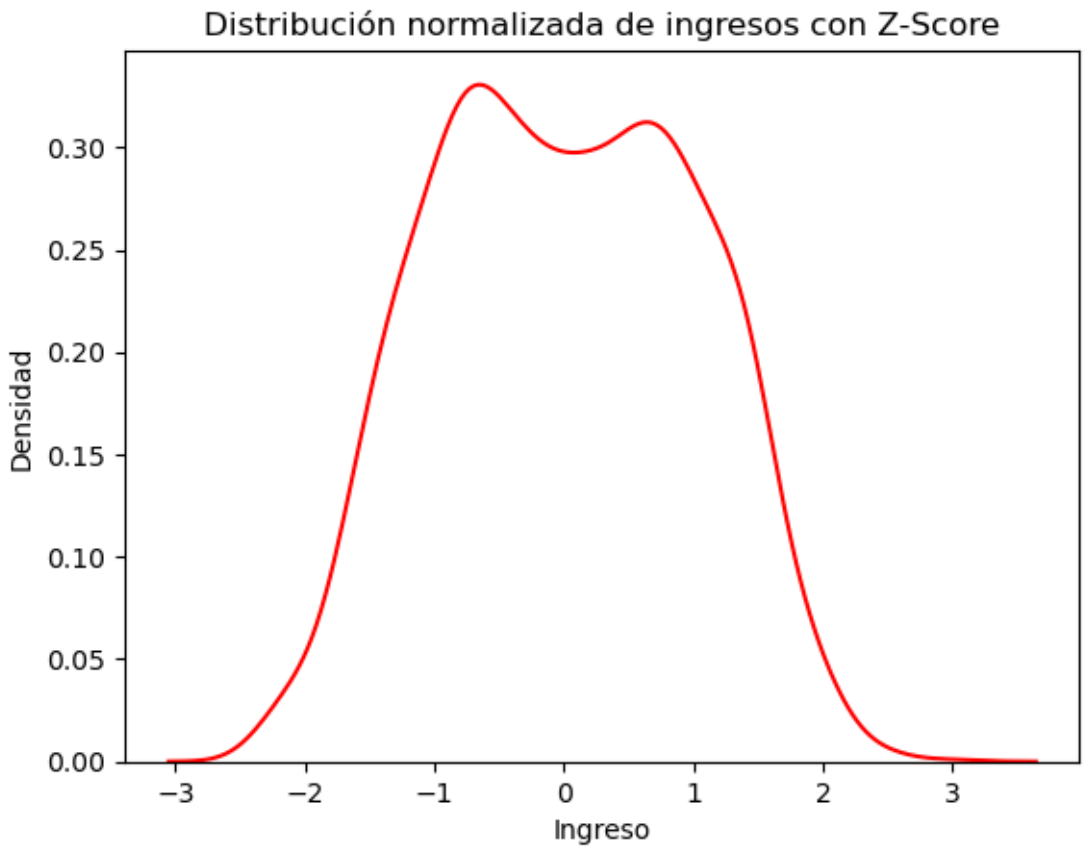
Si quisiéramos saber si el segundo sujeto (comenzando por la derecha) que ha calificado con 5 en ambas asignaturas ha sacado una puntuación equivalente tendríamos que estandarizar ambas calificaciones:

$$z_1 = \frac{5-6.61}{1.65} = -1 \quad z_2 = \frac{5-4.61}{2.81} = 0.01$$

Como vemos ambas calificaciones no son equivalentes, pues mientras un 5 en la primera asignatura tiene por encima el 76% de las calificaciones de la clase. En la otra asignatura es una medida muy cercana a la media de la clase.

---

[6]:
```
# Entonces para la gráfica de ingreso se tiene:
def z_score(db):
    media=db.mean()
    dstd=db.std()
    return (db-media)/dstd
db2=database.copy()
norm2=z_score(db2['Income'])
sns.kdeplot(norm2,color='red')
plt.title('Distribución normalizada de ingresos con Z-Score')
plt.ylabel('Densidad')
plt.xlabel('Ingreso')
plt.show()
```



[ ]:

1.3 Normalización con sigmoide

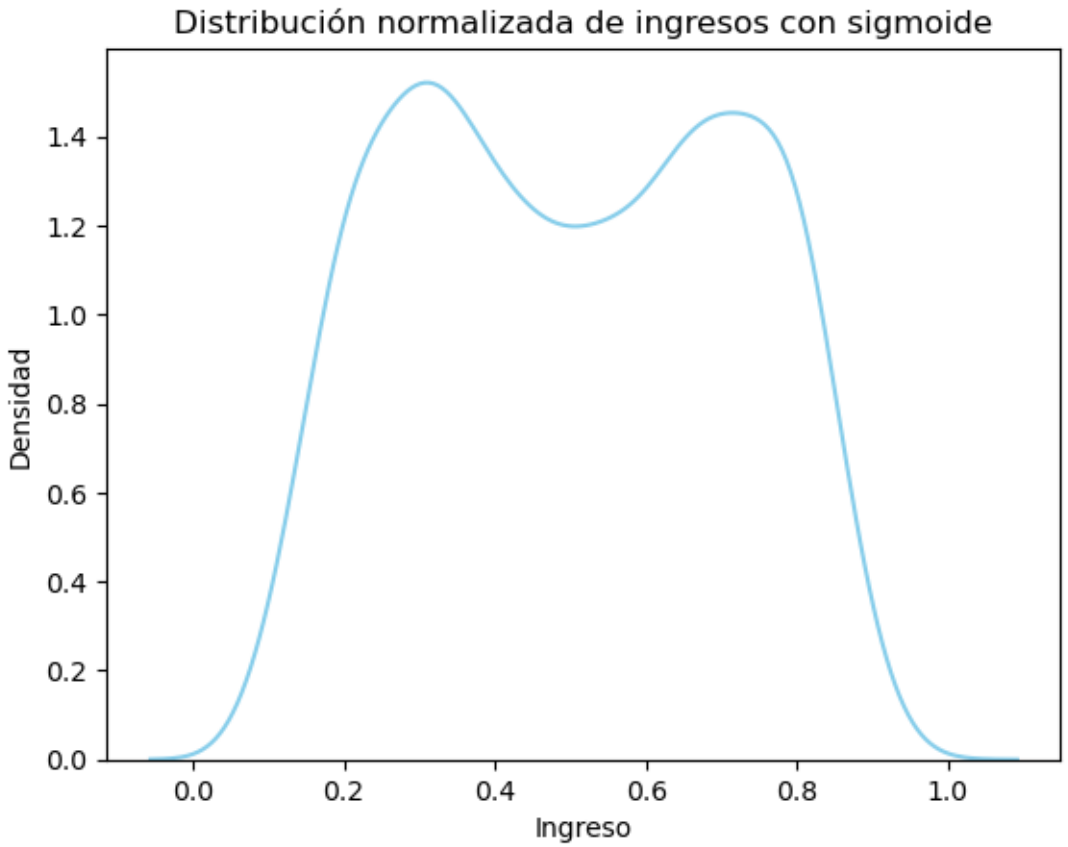
Esta normalización se basa en el calculo de la sigmoide centrada en la media y con la desviación estandar como variable Entonces se calcula mediante:

$$v_{norm} = \frac{1}{1 + e^{-\frac{v_i - \bar{v}}{\sigma_v}}}$$

Teniendo variantes como la utilización de una previa normalización min max para asegurar que se encuentre entre 0 y 1 para ingresarlo directamente

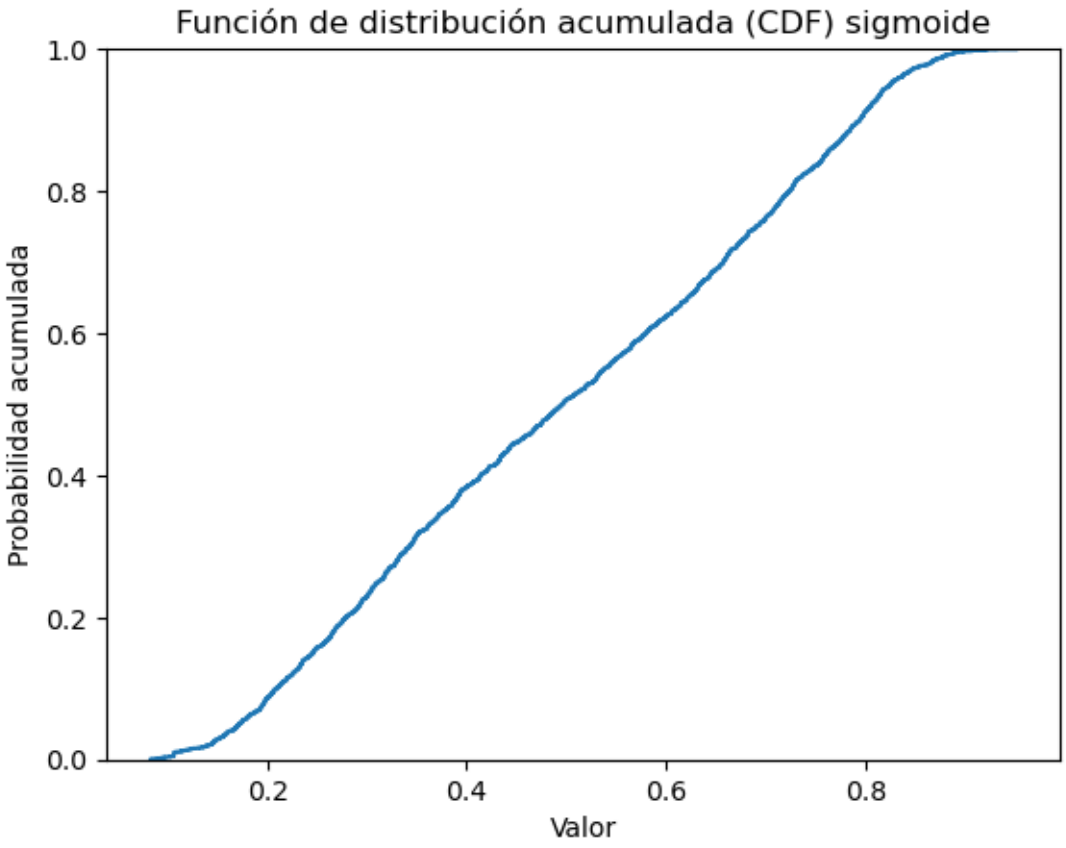
[7]:
```
def sigmoid(db):
    media=db.mean()
    dstd=db.std()
    return 1/(1+np.exp(-((db-media)/(dstd))))

db3=database.copy()
norm3=norm_min_max(db3[['Income']],0,1)
```

```
norm3=sigmoid(norm3['Income'])
sns.kdeplot(norm3,color='#87CEEB')
plt.title('Distribución normalizada de ingresos con sigmoide')
plt.ylabel('Densidad')
plt.xlabel('Ingreso')
plt.show()
print(norm3.mean())
```




```
0.4998125344861291
```

[8]:
```
sns.ecdfplot(norm3)
plt.xlabel('Valor')
plt.ylabel('Probabilidad acumulada')
plt.title('Función de distribución acumulada (CDF) sigmoide')

# Mostramos la gráfica
plt.show()
```



[ ]:

1.4 Generación de datos sintéticos

fuente (smote1.1)

towards data science

smote1.2

2 Generación de datos sinteticos con SMOTE

Primeramente se debe conocer la cantidad de valores que se requieren añadir, se debe conocer la cantidad de valores necesarios para equilibrar la clase, esto se realiza mediante el uso de una diferencia entre la cantidad de clases $n_{total} = n_{c=1} - n_{c=0}$. Donde se categoriza como clase 1 la clase mayoritaria y 0 la clase minoritaria.

2.1 SMOTE v1

Entonces para esta primer parte, se tendrán dos puntos aleatorios y se les sacará la distancia, para finalmente aplicar la formula:

$$v_{SMOTE_i} = v_{a_1} + (P(x) \times (v_{a_2} - v_{a_1}))$$

donde $P(x) : \mathbb{R} \in [0, 1]$ en este caso una función uniforme.

2.2 SMOTE basado en K-NN

El procedimiento es similar, siguiendo los siguientes pasos:

1. Seleccionar un valor aleatorio de la clase minoritaria
2. aplicar algún metodo de distancia con respecto a todos los demás puntos de la clase (euclidiana, Manhattan, Gower), esta última se utiliza cuando se tienen valores tanto categoricos como continuos aparte del objetivo.
3. Ordenar las distancias de menor a mayor y obtener las primeras k distancias menores (sin contar consigo mismo).
4. Seleccionar aleatoriamente una de las k distancias menores.
5. Aplicar la función:

$$v_{SMOTE_i} = v_{a_1} + (P(x) \times (v_{\min(k_{sel})} - v_{a_1}))$$

donde $P(x) : \mathbb{R} \in [0, 1]$ en este caso una función uniforme.

Se aplicó la distancia euclidiana definida como:

$$|n| = \sqrt{x_i - y_i}$$

La Gower como distancia gower, tomando en cuenta que hay valores tanto categoricos como continuos (datos mezclados):

$$S(x_{il}(t), x_{jl}(t)) = \frac{\sum \delta_{ijl}(t) S_{ijl}(t)}{\sum \delta_{ijl}(t)}$$

donde $S_{ijl}(t)$ indica la similaridad para la l-ésima caracteristica entre dos objetos y $\delta_{ijl}(t) \in [0, 1]$ es un coeficiente basado en la posibilidad de que la medida de dos objetos se pierde en el tiempo t. Para variables discretas (incluyendo binarias), el componente de similitud se obtiene como:

$$S_{ijl}(t) = \begin{cases} 1 & si \quad x_{il}(t) = x_{jl}(t) \\ 0 & si \quad x_{il}(t) \neq x_{jl}(t) \end{cases}$$

Para datos continuos:

$$S_{ijl}(t) = 1 - \frac{|x_{il}(t) - x_{jl}(t)|}{R_l(t)}$$

Donde $R_l(t)$ es el rango de la l-ésima variable en el tiempo t sobre todos los objetos escritos como $R_l(t) = \max_m X_{ml}(t) - \min_m X_{ml}(t)$. Correspondiente a la disimilaridad medida que puede ser obtenida o simplemente usando $D(x_i, x_j) = 1 - S(x_i, x_j)$ ...

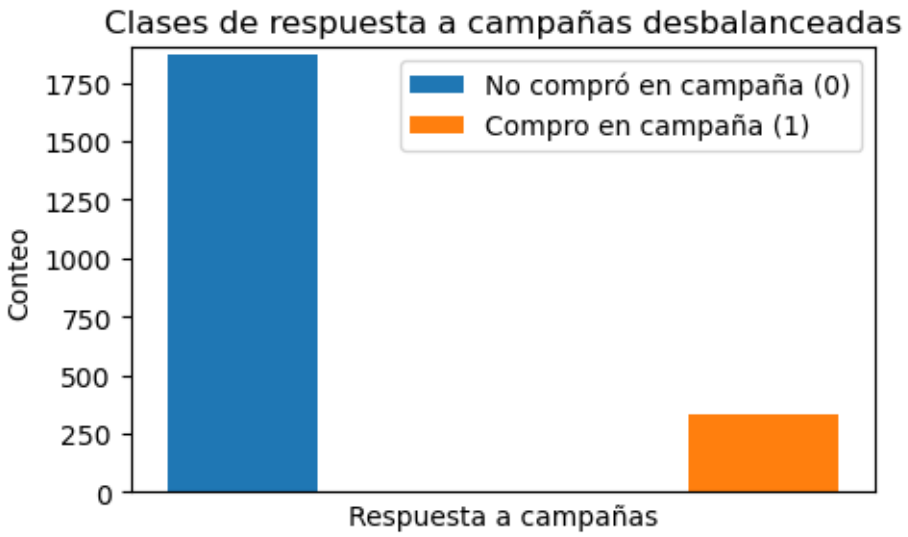
[9]:
```
data = [[0.2, 0.4, 0], [0.3, 0.5, 0], [0.4, 0.5, 0], [0.7, 0.9, 1], [0.8, 0.7,␣
↪1]]
```

[10]:
```
positive_data = []
negative_data = []

for observation in data:
    if observation[-1] == 1:
        positive_data.append(observation[:-1])
    else:
        negative_data.append(observation[:-1])
```

[11]:
```
print(negative_data)
```

```
[[0.2, 0.4], [0.3, 0.5], [0.4, 0.5]]
```

[12]:
```
data_p1=database[['MntWines',
       'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds','Response']].copy()
ren,co=data_p1.shape
ncc=[]
cc=[]
for dato in range(ren):
    if data_p1['Response'][dato] == 0:
        ncc.append(0)
    else:
        cc.append(1)
col=['No compró en campaña (0)','Compro en campaña (1)']
plt.figure(figsize=(5,3))
plt.hist([ncc,cc],bins=2,label=col)
plt.xlabel('Respuesta a campañas')
plt.ylabel('Conteo')
plt.axis([0,1,0,1900])
plt.tick_params(bottom=False,labelbottom=False)
plt.legend()
plt.title('Clases de respuesta a campañas desbalanceadas') # modificar si es␣
 ↪posible la escala de los valores
c0,c1=data_p1['Response'].value_counts()
```



[13]:
```
n_ds=abs(c0-c1)
print(n_ds)
```

```
1539
```

[14]:
```
# smote para valores continuos **********************************
def euc_dis(x1,x2):
    return np.sqrt((x1-x2)**2)

def e_dis_abs(x1,x2):
    #r,c=x1.shape
    x3=(x1-x2)**2
    x3=np.sqrt(sum(x3))
    return x3


def smotev1(minclass,val_class,ndg):
    ## Tomar dos valores aleatorios
    re,c=minclass.shape
    sdata=np.zeros((1,c))
    for a in range(ndg):
        sel_rnd=np.random.choice(re,2,replace=False)
        dis=euc_dis(minclass.iloc[sel_rnd[0]],minclass.iloc[sel_rnd[1]])
        sdata=np.vstack((sdata,(minclass.iloc[sel_rnd[0]]+(np.random.
 ↪uniform(0,1)*
                                                            (minclass.
 ↪iloc[sel_rnd[1]]-minclass.iloc[sel_rnd[0]])))))
    sdata=np.delete(sdata,0,0)
    df=pd.DataFrame(sdata,columns=minclass.columns)
```

```
    df['Response']=val_class
    return df


def smotev2(minclass,val_class,ndg,k):
    ## Tomar dos valores aleatorios
    re,c=minclass.shape
    sdata=np.zeros((1,c))
    #print('antes',sdata.shape)
    for aa in range(ndg):
        sel_rnd=np.random.choice(re,1,replace=False)
        dlist=[]

        for b in range(re):
            dlist.append(e_dis_abs(minclass.iloc[sel_rnd[0]],minclass.iloc[b]))

        dlist2=sorted(dlist)
        #print(dlist2[1:k+1])
        seleccion=np.random.choice(dlist2[1:k+1],1)
        val=dlist.index(seleccion)
        #print(val[0])
        sdata=np.vstack((sdata,(minclass.iloc[sel_rnd[0]]+(np.random.
↪uniform(0,1)*
                                                           (minclass.
↪iloc[val]-minclass.iloc[sel_rnd[0]])))))
        dlist=[]
        #print(aa)
    sdata=np.delete(sdata,0,0)
    #print(sdata.shape)
    df=pd.DataFrame(sdata,columns=minclass.columns)
    df['Response']=val_class
    return df
```

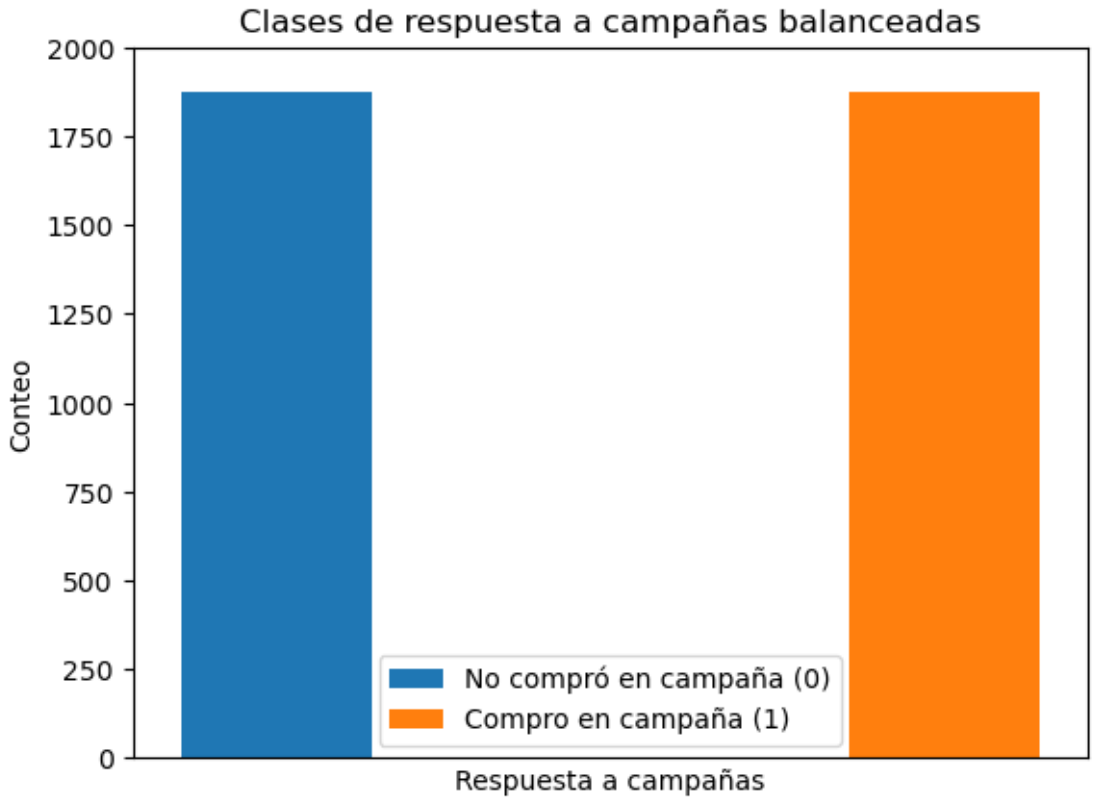
[15]:
```
min_class=data_p1[data_p1['Response']==1]
max_class=data_p1[data_p1['Response']==0]
```

[16]:
```
#dsint=pd.DataFrame(smotev1(min_class,1,10))
import time
t2=time.time()
dsint=smotev1(min_class,min_class['Response'][0],n_ds)
t1=time.time()
balclass=pd.concat([data_p1,dsint],ignore_index=True)
print(balclass.shape)
print('tiempo de ejecución:',t1-t2)
```

```
(3744, 7)
tiempo de ejecución: 0.8290863037109375
```

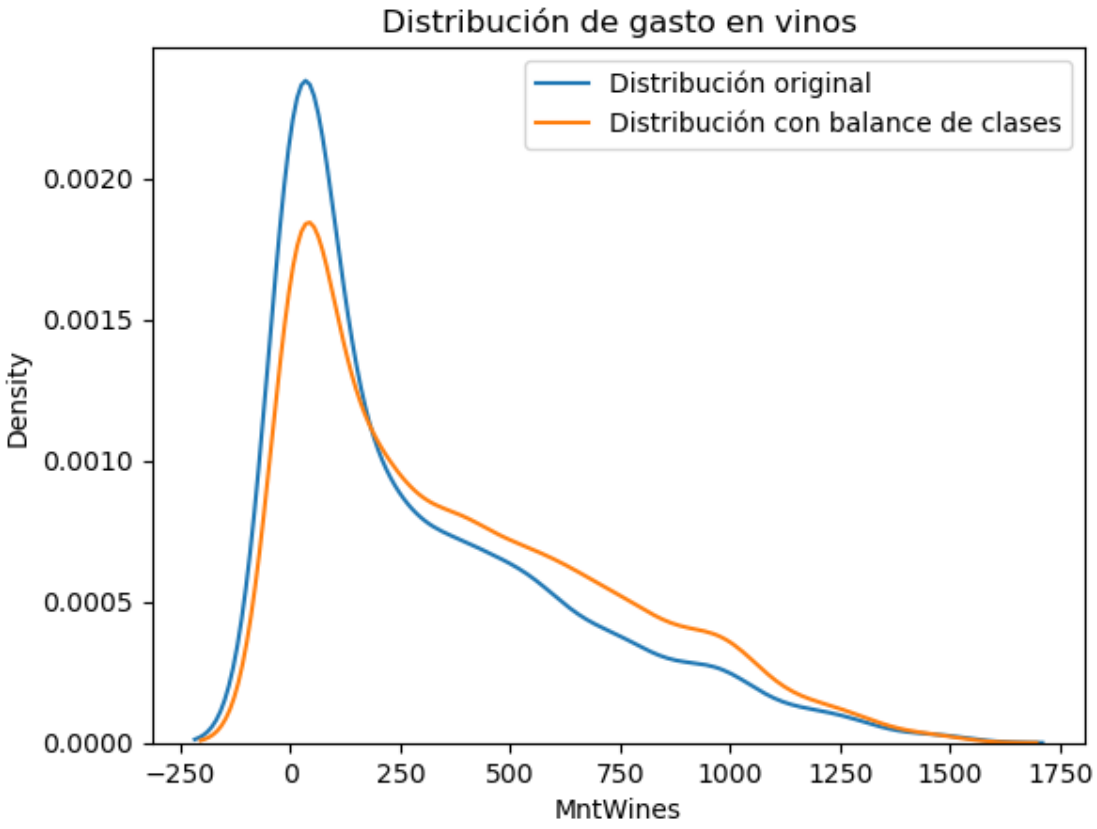
[17]:
```
ren,col=balclass.shape
ncc=[]
cc=[]
for dato in range(ren):
    if balclass['Response'][dato] == 0:
        ncc.append(0)
    else:
        cc.append(1)
col=['No compró en campaña (0)','Compro en campaña (1)']
#plt.figure(figsize=(6,3))
plt.hist([ncc,cc],bins=2,label=col)
plt.xlabel('Respuesta a campañas')
plt.ylabel('Conteo')
plt.axis([0,1,0,2000])
plt.tick_params(bottom=False,labelbottom=False)
plt.legend()
plt.title('Clases de respuesta a campañas balanceadas') # modificar si es␣
↪posible la escala de los valores
```

[17]: Text(0.5, 1.0, 'Clases de respuesta a campañas balanceadas')



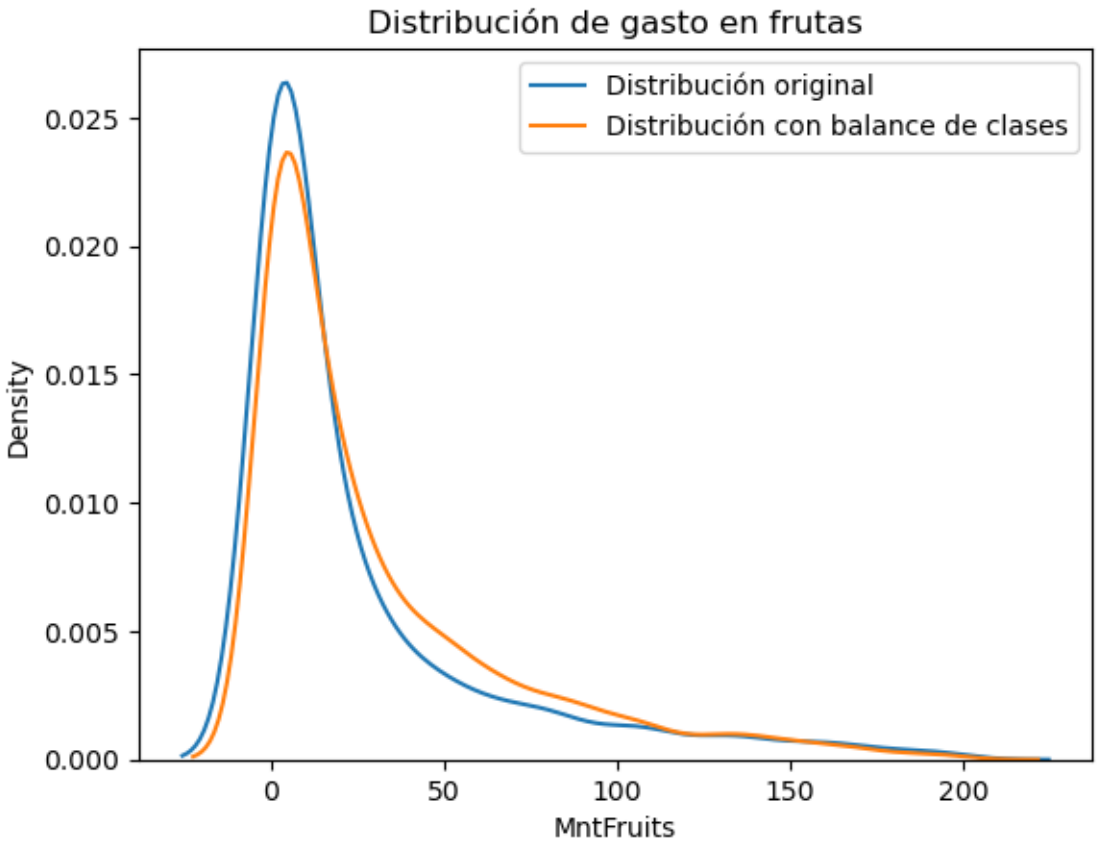
[18]:
```
sns.kdeplot(database['MntWines'])
sns.kdeplot(balclass['MntWines'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en vinos')
```

[18]: Text(0.5, 1.0, 'Distribución de gasto en vinos')



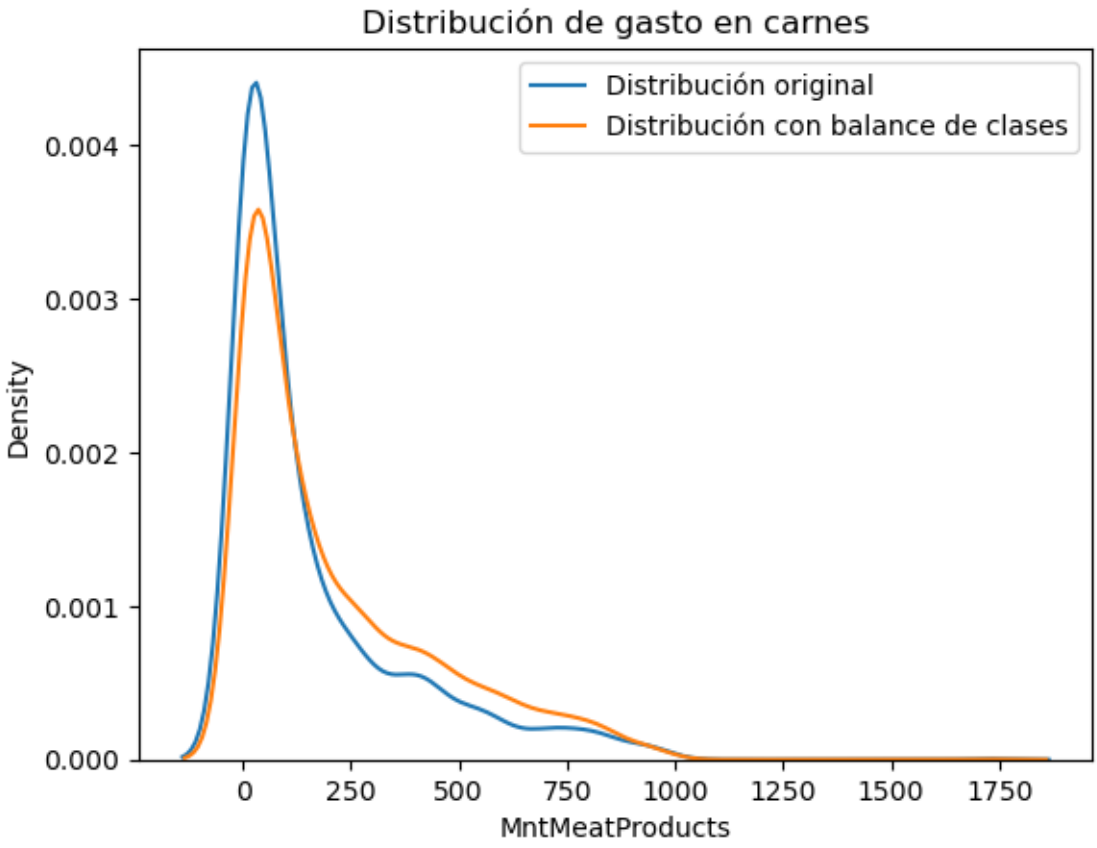
[19]:
```
sns.kdeplot(database['MntFruits'])
sns.kdeplot(balclass['MntFruits'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en frutas')
```

[19]: Text(0.5, 1.0, 'Distribución de gasto en frutas')



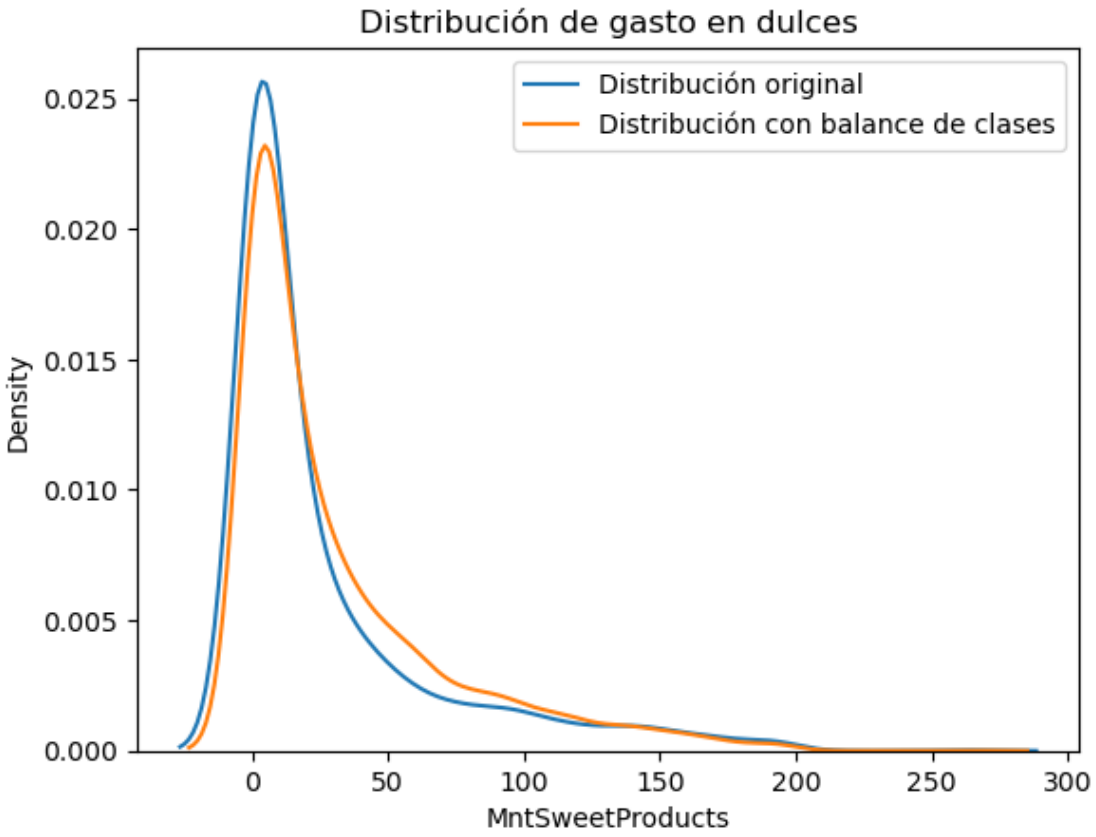
[20]:
```
sns.kdeplot(database['MntMeatProducts'])
sns.kdeplot(balclass['MntMeatProducts'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en carnes')
```

[20]:
```
Text(0.5, 1.0, 'Distribución de gasto en carnes')
```



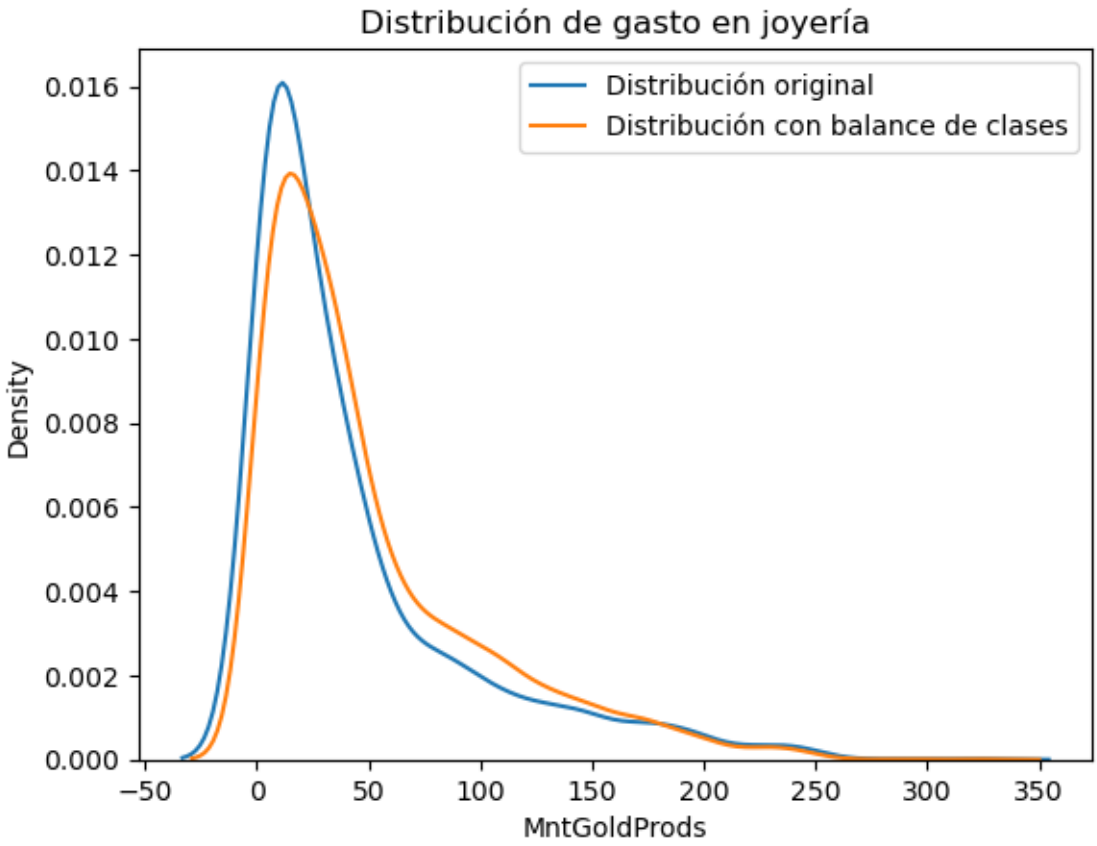
[21]:
```
sns.kdeplot(database['MntSweetProducts'])
sns.kdeplot(balclass['MntSweetProducts'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en dulces')
```

[21]:
```
Text(0.5, 1.0, 'Distribución de gasto en dulces')
```



[22]:
```
sns.kdeplot(database['MntGoldProds'])
sns.kdeplot(balclass['MntGoldProds'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en joyería')
```

[22]:
```
Text(0.5, 1.0, 'Distribución de gasto en joyería')
```



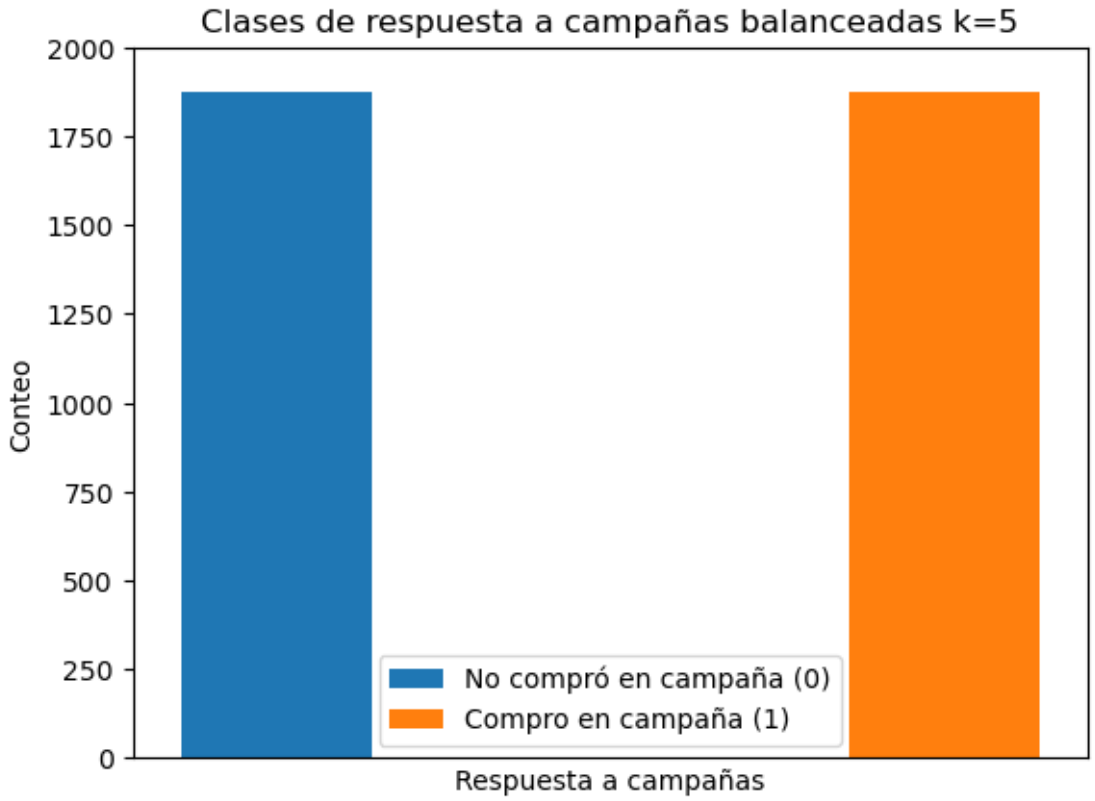
[23]:
```
data_p2=data_p1.copy()
t2=time.time()
dsint2=smotev2(min_class,min_class['Response'][0],n_ds,5) ## con 3 y 5
t1=time.time()
balclass2=pd.concat([data_p2,dsint2],ignore_index=True)

print('tiempo de ejecución:',t1-t2)
```

```
tiempo de ejecución: 91.38653922080994
```

[24]:
```
ren,col=balclass2.shape
ncc=[]
cc=[]
for dato in range(ren):
    if balclass['Response'][dato] == 0:
        ncc.append(0)
    else:
        cc.append(1)
col=['No compró en campaña (0)','Compro en campaña (1)']
#plt.figure(figsize=(6,3))
```

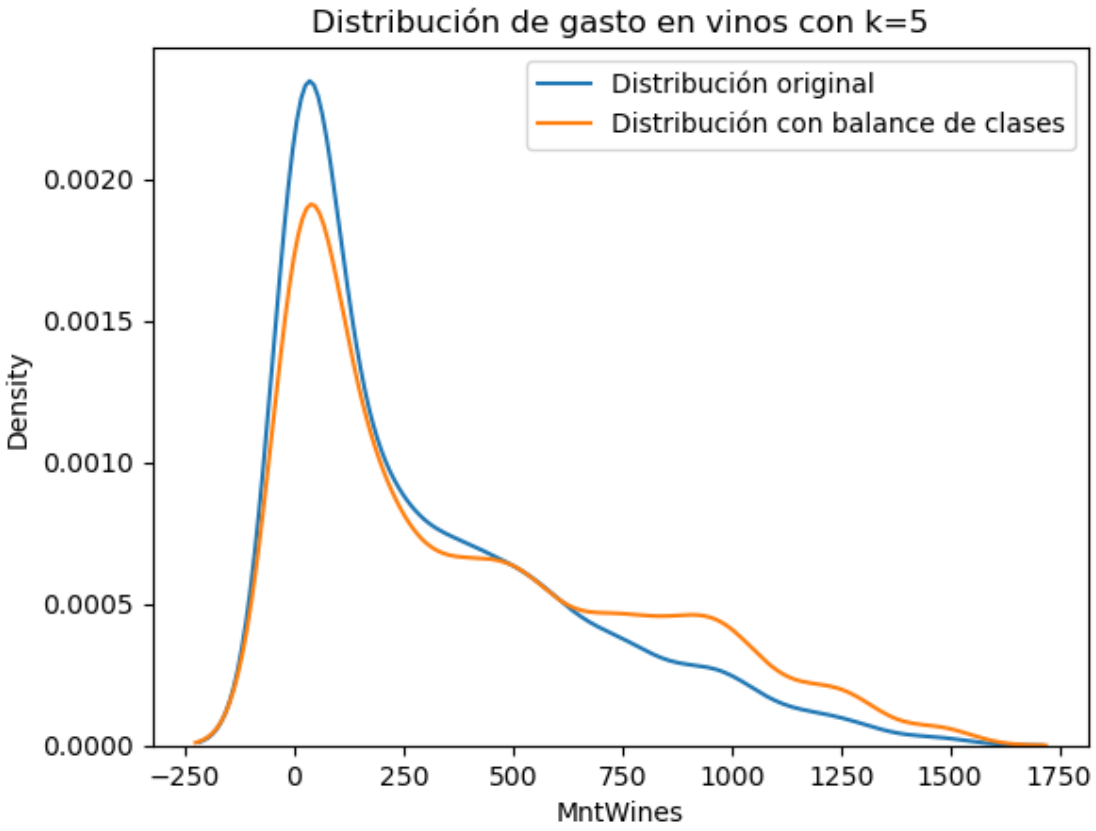
```
plt.hist([ncc,cc],bins=2,label=col)
plt.xlabel('Respuesta a campañas')
plt.ylabel('Conteo')
plt.axis([0,1,0,2000])
plt.tick_params(bottom=False,labelbottom=False)
plt.legend()
plt.title('Clases de respuesta a campañas balanceadas k=5') # modificar si es␣
↪posible la escala de los valores
```

[24]: `Text(0.5, 1.0, 'Clases de respuesta a campañas balanceadas k=5')`



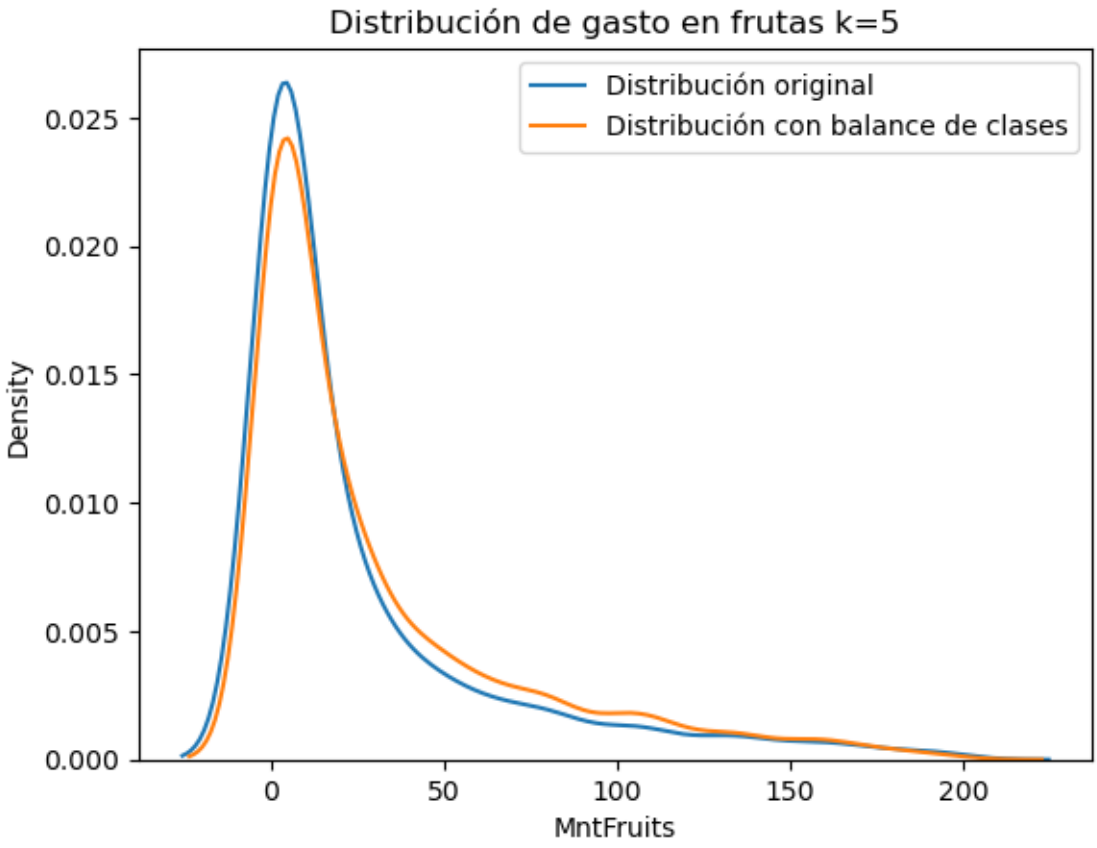
[25]:
```
sns.kdeplot(database['MntWines'])
sns.kdeplot(balclass2['MntWines'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en vinos con k=5')
```

[25]: `Text(0.5, 1.0, 'Distribución de gasto en vinos con k=5')`



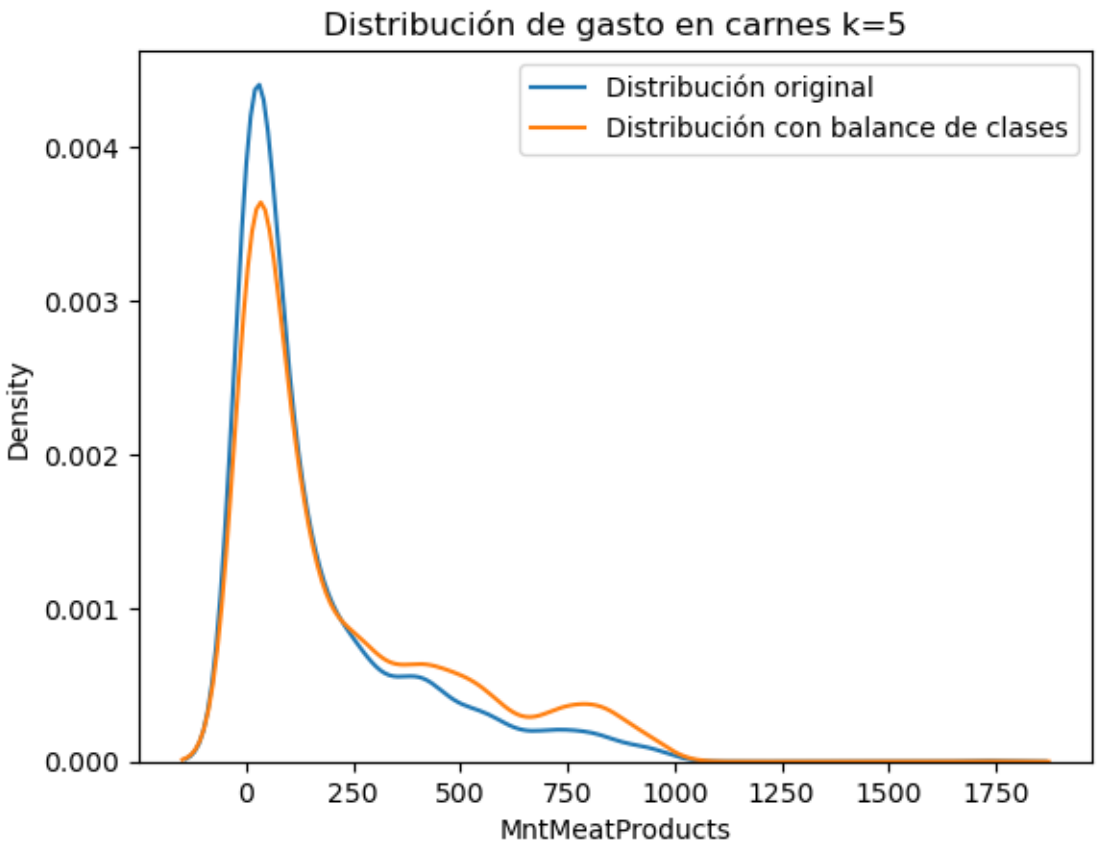
[26]:
```
sns.kdeplot(database['MntFruits'])
sns.kdeplot(balclass2['MntFruits'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en frutas k=5')
```

[26]: `Text(0.5, 1.0, 'Distribución de gasto en frutas k=5')`



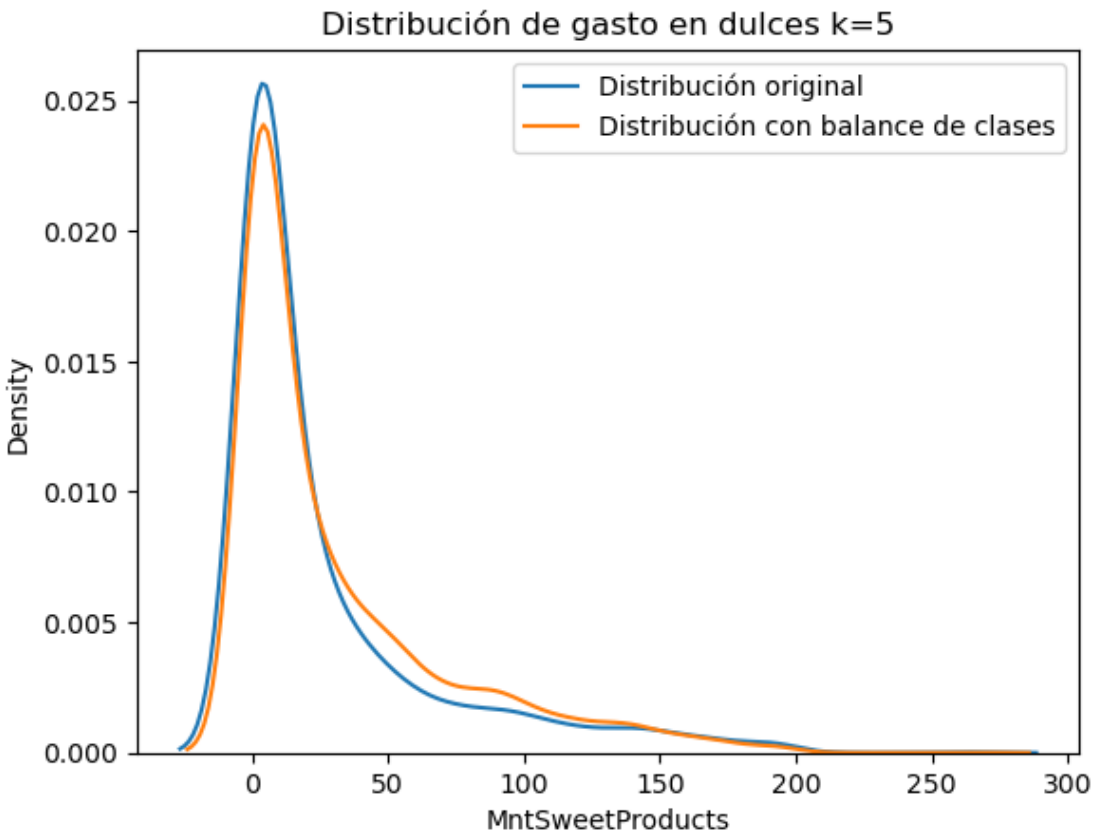
[27]:
```
sns.kdeplot(database['MntMeatProducts'])
sns.kdeplot(balclass2['MntMeatProducts'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en carnes k=5')
```

[27]:
```
Text(0.5, 1.0, 'Distribución de gasto en carnes k=5')
```



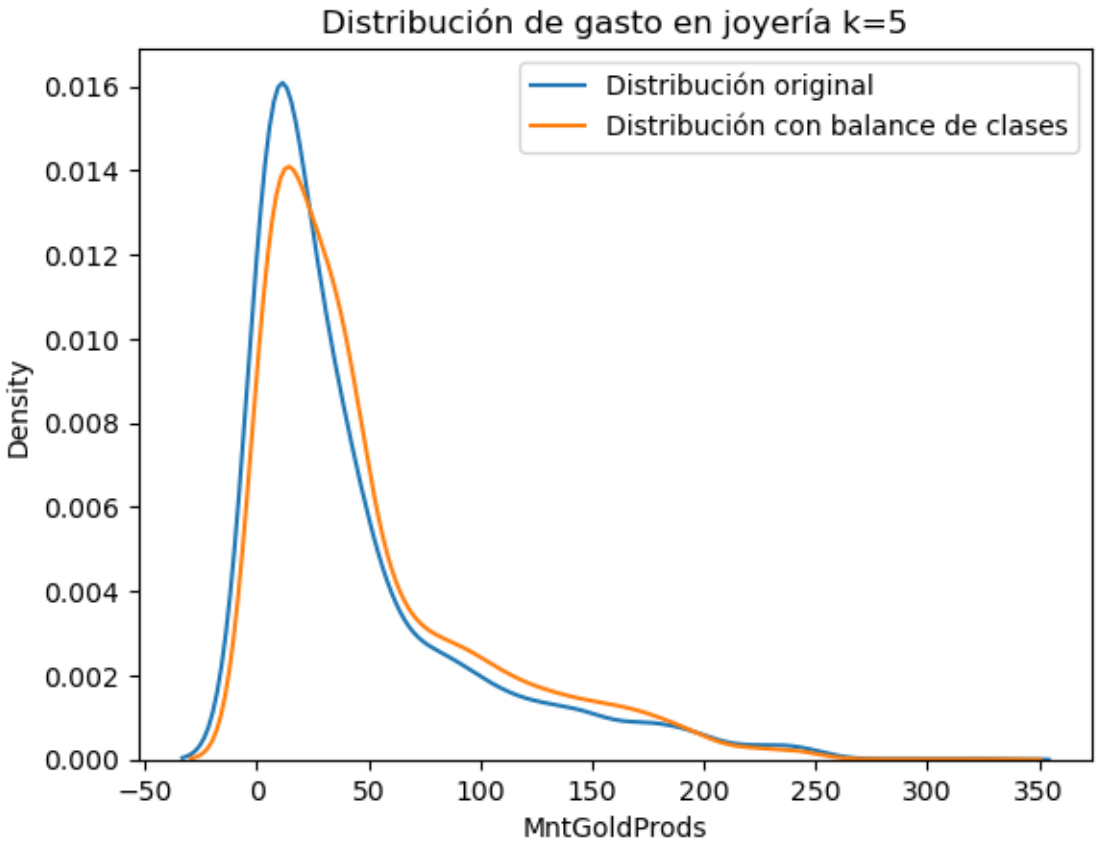
[28]:
```
sns.kdeplot(database['MntSweetProducts'])
sns.kdeplot(balclass2['MntSweetProducts'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en dulces k=5')
```

[28]:
```
Text(0.5, 1.0, 'Distribución de gasto en dulces k=5')
```



[29]:
```
sns.kdeplot(database['MntGoldProds'])
sns.kdeplot(balclass2['MntGoldProds'])
plt.legend(['Distribución original','Distribución con balance de clases'])
plt.title('Distribución de gasto en joyería k=5')
```

[29]:
```
Text(0.5, 1.0, 'Distribución de gasto en joyería k=5')
```


Distribución de gasto en joyería k=5
Distribución original
Distribución con balance de clases
Density
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
−50
0
50
100
150
200
250
300
350
MntGoldProds

[ ]: