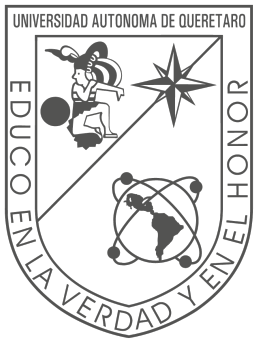
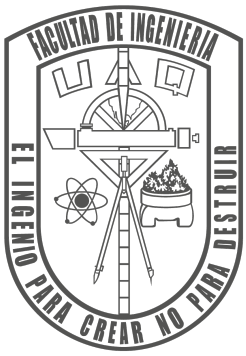
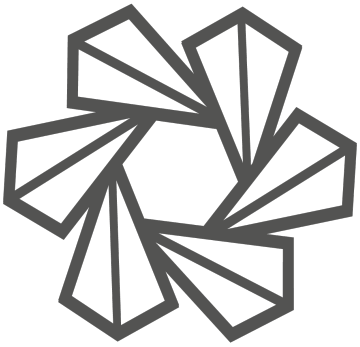
Universidad Autónoma de Querétaro

Facultad de Ingeniería
División de Investigación y Posgrado







Práctica 5

Clasificación Knn

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
2 de Junio de 2023

Índice

1. Objetivo

Esta práctica tiene como objetivo el de generar una clasificación Knn a un conjunto de datos etiquetados y con categorías bien definidas para poder realizar una predicción de la clasificación categórica de datos nuevos, aplicando conceptos anteriores que permitan obtener una mayor fiabilidad en el modelo construido.

2. Introducción

La práctica consiste en obtener una base de datos etiquetada en categorías y realizar un preprocesamiento de datos. Para posteriormente aplicar un algoritmo de Knn, realizando diferentes pruebas con diferentes valores de n vecinos cercanos y observar comportamientos, así como también obtener las métricas necesarias para evaluar el desempeño del modelo observado. Permitiendo obtener conclusiones de los resultados obtenidos.

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 utilizó el siguiente método [2, 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 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:

$$v_{norm} = \frac{1}{1 + e^{-\frac{v_i - \bar{v}}{\sigma_v}}} \tag{2}$$

3.2. Generación de datos sintéticos

El uso de métodos para generar datos sintéticos 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 [4]

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 1), 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:

$$S_1 = p_1 + (P(x)(p_2 - p_1)) \tag{3}$$

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

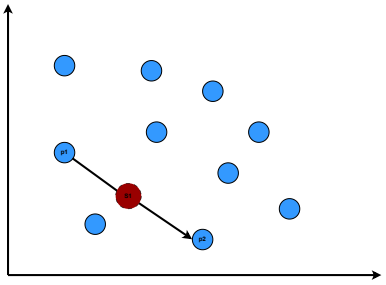


Figura 1: Algoritmo SMOTE para puntos aleatorios.

3.3. K-Folds (validación cruzada)

Este método consiste en crear k grupos aleatorios de igual tamaño para poder realizar una combinación de las precisiones y obtener un promedio. De este modo es posible observar si existen datos con los que aprenda mejor el modelo o no (véase Tabla 1).

Tabla 1: Grupos de K-folds

	Grupos				
Combinaciones	1_1	1_2	1_3	…	1_k
	2_1	2_2	2_3	…	2_k
	⋮	⋮	⋮	⋱	⋮
	C_1	C_2	C_3	…	C_k
	Grupo de prueba				
	Grupos de entrenamiento				

finalmente se aplica la media de los valores obtenidos en las diferentes métricas:

$$M(comb) = \frac{1}{c}\sum_{i=1}^{c} p(i) \tag{4}$$

3.4. Algoritmo Knn

El algoritmo de *Knn* es un algoritmo de clasificación categórica supervisado que permite realizar una predicción de las clases [5]. Para realizar este algoritmo es necesario tener una base de datos etiquetada. Esta base de datos será dividida segun sea conveniente en un periodo de entrenamiento y prueba, donde el entrenamiento será modelado en el conjunto de datos y estos serán la referencia para los datos de prueba [6]. Posteriormente se evaluará un punto de prueba y se obtendrán las distancias con respecto a todos los puntos de entrenamiento, agrupando los n más cercanos [7]. Para posteriormente elegir por medio de un voto mayoritario o plural el que más frecuencia de clase tenga, como se muestra en la Figura 2.

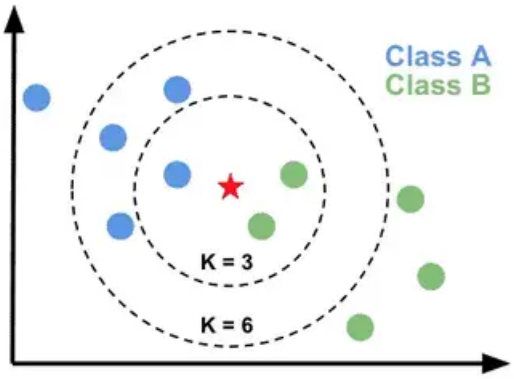


Figura 2: Knn visualización.

3.5. Métricas de evaluación

Para poder medir el desempeño del modelo *knn* se tiene que partir de una matriz de confusión, la cual nos permite tener una observabilidad del comportamiento del modelo y se puede describir como la Figura 3

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 3: Matriz de confusión para datos categóricos clasificados.

A partir de esta matriz podemos obtener las métricas:

- **Exactitud**: Explica el desempeño total del modelo en cuanto a sus aciertos obtenidos.

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

- **Precisión**: explica la precisión de las predicciones positivas.

$$Pr = \frac{TP}{TP + FP} \tag{6}$$

- **Sensibilidad (recall)**: Indica la cobertura de las muestras positivas.

$$Sens = \frac{TP}{TP + FN} \tag{7}$$

- **F1 score**: Calcula la media armónica de la precisión y sensibilidad (recall) de forma que se enfatiza el valor más bajo entre ambas.

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{8}$$

3.6. Coeficiente de correlación

Se obtiene mediante la covarianza del modelo de regresión con respecto a los valores reales. se define como CC y tiene un rango de $[-1, 1]$ donde 1 significa una correlación positiva perfecta, -1 una correlación negativa perfecta y cuando se tiende a 0 no existe correlación lineal.

$$CC = \frac{S_{PA}}{\sqrt{S_p S_A}} \tag{9}$$

donde:

$$\begin{aligned} S_{PA} &= \left(\frac{\sum_{i=1}^{n_d}(\hat{y}_i-\bar{\hat{y}})(y_i-\bar{y})}{n_d-1}\right) \\ S_P &= \frac{\sum_{i=1}^{n_d}(\hat{y}_i-\bar{\hat{y}})^2}{n_d-1} \\ S_A &= \frac{\sum_{i=1}^{n_d}(y_i-\bar{y})^2}{n_d-1} \end{aligned} \tag{10}$$

3.7. Análisis de Componentes Principales (PCA)

Para realizar el algoritmo de PCA se deben seguir los siguientes pasos:

1. Obtener los datos —en todas sus dimensiones
2. Restar la media. Para que PCA funcione de manera correcta, se requiere restar la media de cada dimensión.
3. Calcular la matriz de covarianza.
4. Calcular los eigenvalores y los eigenvectores de la matriz de covarianza.
5. Escoger los componentes y formar el vector de características. De la columna de los eigenvalores, el elemento con el número mayor es el componente principal del set de datos.
6. En general, una vez que los eigenvectores se encuentran en la matriz de covarianza, se ordenan de mayor a menor dando estos los componentes principales en orden.

4. Materiales y Métodos

4.1. Materiales

4.1.1. Base de datos

La base de datos elegida para esta práctica fue obtenida de kaggle realizada por la universidad China Chung-Hua [8] en donde se expone un conjunto de muestras obtenidas para realizar concreto y medir su resistencia. Se caracteriza este tipo de comportamientos como no lineales y altamente impredecibles debido a la diferencia en aspectos externos a la fabricación del concreto (repetibilidad, calidad variable de los materiales, etc.). La base de datos consiste en 9 atributos y 1030 instancias. En donde los atributos principalmente se dividen en 3 tipos:

- Materiales constructivos (medido en kg):
  1. Concreto
  2. Escoria de alto horno
  3. Ceniza
  4. Agua

5. Aditivo
6. grava
7. Arena

- Edad del concreto (1-365)
  1. Días de vida.
- Resistencia del concreto (M_{pa})
  1. Resistencia a la compresión. Esta clase será la salida (clusters) en la cual se clasifica en 4 según su resistencia para diferentes aplicaciones: Resistencia baja $H_{<30}$, Resistencia de uso convencional $H_{[30,45]}$, Alta resistencia $H_{[45,60]}$ y Ultra alta resistencia $H_{>60}$

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, además se agregará la librería de scikitlearn para comprobar resultados.

4.2. Metodología

La metodología consiste en la recopilación de las bases de datos, aplicar métodos de preprocesamiento de datos, realizar una segmentación de datos para realizar validación cruzada y segregación (80-20 %) para despúes evaluar el modelo con diferentes valores de los vecinos cercanos, los cuales serán calculados mediante otros métodos, para posteriormente mostrar los resultados de las métricas de evaluación (véase Figura 4).

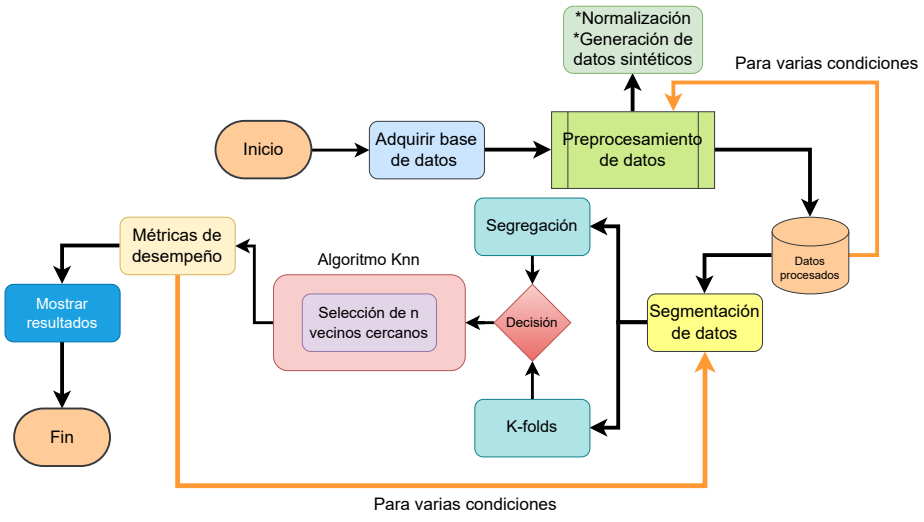


Figura 4: Metodología de la práctica.

5. Pseudocódigo

Algoritmo 1 Pseudocódigo regresión.

```
Inicio
 Class Knn(entrenamiento,prueba):
        Distancias
        Regresar clase
 Class métricas(resultado,prueba):
        Regresar (ex,pres,sens,f1)
 Class n_adecuado(kargs):
        Para p en x:
              Almacenar res[p] ← Knn,métricas
        Graficar res
 Class Preprocesamiento(datos):
         Regresar norm,Datos_sint.
 Class Segregación(%p,%e):
         Regresar dato_p,dato_e
 Class k-folds(datos,k):
      Regresar datos_k
Datos_n ← Preprocesamiento(datos)
t ← n_adecuados(Datos_n,0,100)
Datos ← segregación(80,20),k-folds([3-nn])
Para prueba en pruebas:
       Resultado[prueba]← Knn(Datos)
       Metrica[prueba] ← métricas(Resultado[prueba])
Fin
```

6. Resultados

6.1. Preprocesamiento y análisis de los datos

6.1.1. Distribución de los atributos y generalidades

Primeramente se realizó un análisis de las distribuciones de los atributos para observar valores faltantes, atípicos y generalidades de la base de datos. Se observan principalmente distribuciones normales y exponenciales, obteniendo la Figura 5, donde se observa que no hay muchos valores atípicos para la base de datos, por lo que no será necesario realizar un modelo especial para los datos atípicos o alguna otra técnica en el tratamiento de los mismos.

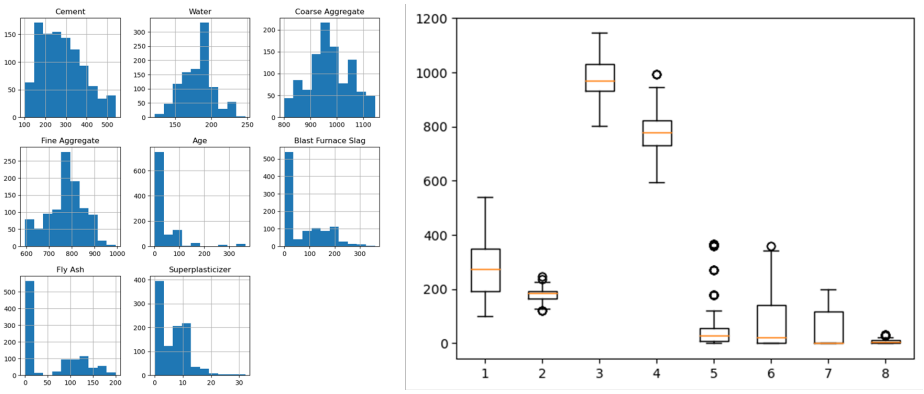


Figura 5: Distribuciones originales de la base de datos.

6.1.2. Balance de clases

Se observó un desbalance de clases, en donde se tuvo que aplicar el algoritmo de SMOTE aleatorio para poder realizar el balance de clases, en total se realizaron las siguientes cantidades de instancias para cada clase minoritaria con su respectiva nomenclatura de categoría como se muestra en la Tabla 2.

Tabla 2: Desbalance de clases.

Clase minoritaria	Desbalance de clase	Porcentaje de desbalance con respecto a la mayoritaria
1 ($H_{[30,45]}$)	36	9.04%
2 ($H_{[45,60]}$)	222	55.77%
3 (H_{60})	304	76.381%

Obteniendo un balance perfecto entre clases (Figura 7) comparado con la anterior distribución, obteniendo finalmente una cantidad de 1592 instancias.

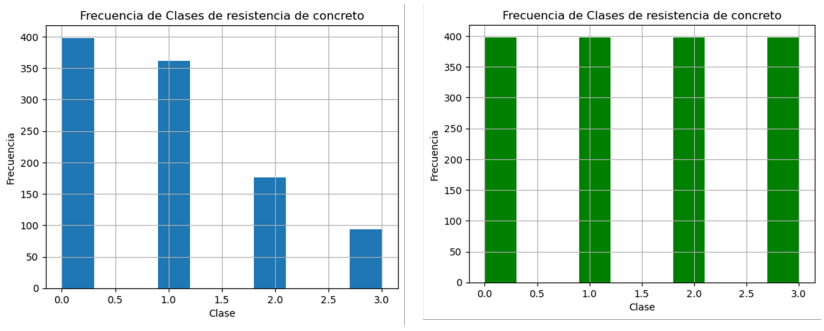


Figura 6: Balance de clases con generación de datos sintéticos.

6.1.3. Reducción de dimensionalidad

Una vez que se balancearon las clases, se realizó un analisis de componentes principales (PCA), obteniendo lo siguiente (véase Tabla 3).

Tabla 3: Resultado de reducción de dimensionalidad.

Atributo	Porcentaje	Atributo	Porcentaje
Cemento	31.18%	Aditivo	10.317%
Agua	20.32%	Ceniza	6.19%
Grava	18.36%	Arena	1.90%
Escoria	11.32%	Edad	0.33%

Por lo que se decidio realizar una reducción de dimensionalidad a 5 atributos de entrada, los cuales son: *Cemento, Agua, Grava, Escoria, Aditivo*. Pues estos acumulan el 91.55 % de la importancia de los datos, teniendo finalmente una base de datos de 1592 instancias y 6 atributos. Observando en la Figura 7 cómo se modificó la distribución con la generación de los datos sinteticos con SMOTE.

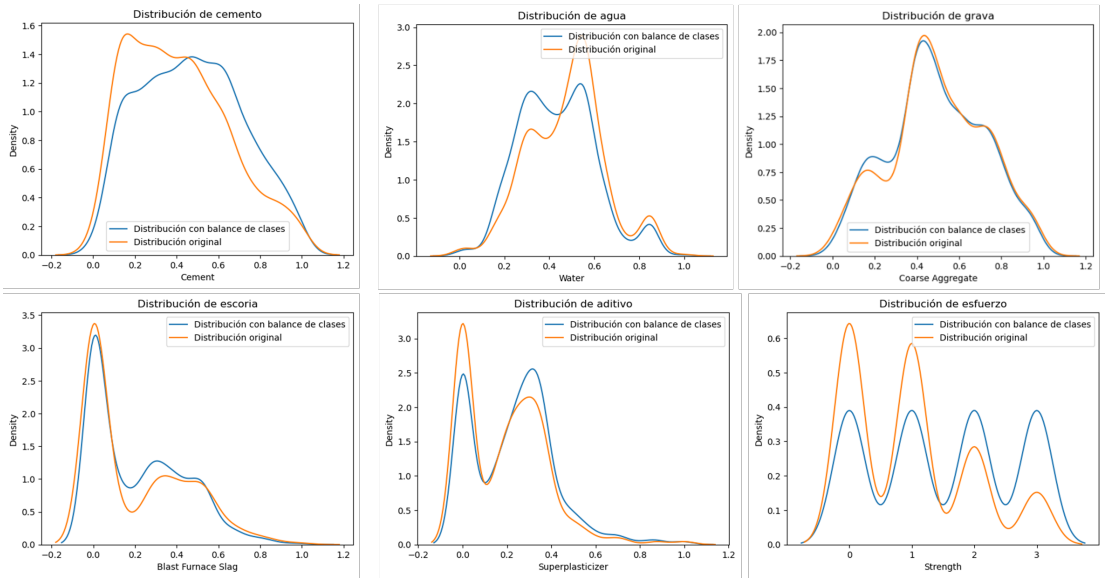


Figura 7: Gráficos de distribución de los atributos más significativos generados con SMOTE con respecto a los originales.

6.1.4. Normalización y correlación

Se realizó una normalización *min-max* en los atributos para poder tener todo en las mismas unidades (adimensional) pues es recomendable realizarlo con este algoritmo, y de esta manera poder tener una mejor comparación entre atributos. Todo esto con la intención de observar alguna relación implícita que pueda existir en la base de datos 8.

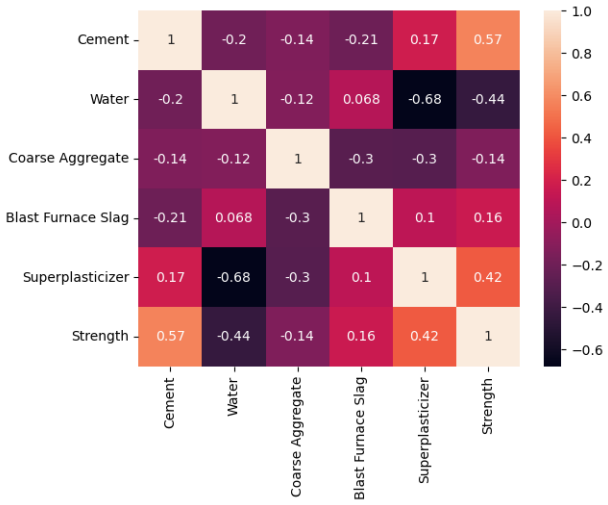


Figura 8: Matriz de correlación entre atributos.

Lo que comprueba la metadata de la base de datos, son altamente no lineales.

Finalmente se realizó un gráfico de caja para poder observar la dispersión e identificar valores atípicos (véase Figura 9).

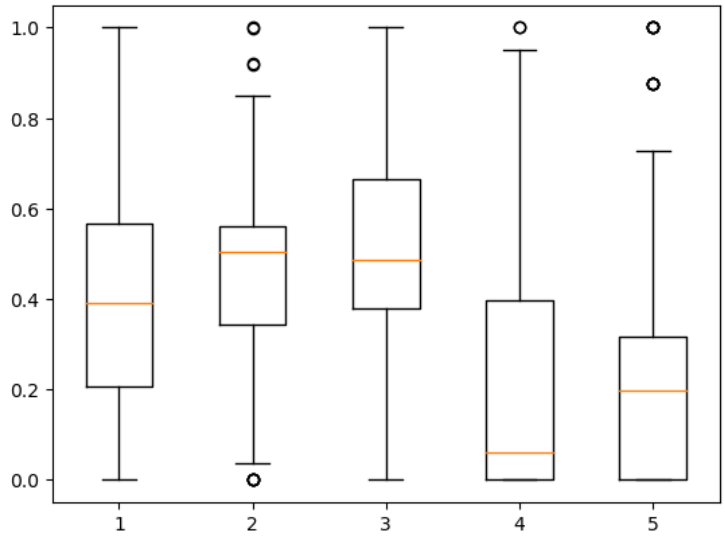


Figura 9: diagrama de caja de los atributos reducidos y normalizados.

donde:

1. *Cemento*
2. *Agua*
3. *Grava*

4. *Escoria*

5. *Aditivo*

Se observa que ya no hay muchos datos atípicos debido a los datos sintéticos que permitieron crear nuevas instancias en esos rangos.

6.2. Algoritmo knn

6.2.1. Calculo de vecinos adecuado

Para esta prueba se realizó un analisis de la exactitud del modelo segmentado en un 80 % para entrenamiento y un 20 % para pruebas. Además de realizar una prueba de K-folds con el fin de observar un rendimiento medio del modelo.

Además de que se observará el rendimiento con el método del codo distintos datos, como fueron: **datos originales**, **datos completos sin PCA** (con datos sintéticos) y **datos completos con PCA y normalizados**.

Como se observa en la Figura 10, la precisión de los datos originales son poco fiables y el algoritmo no tiene una precisión muy estable y a la baja. Por lo que no es sencillo obtener la cantidad de vecinos que permitan generalizar el modelo.

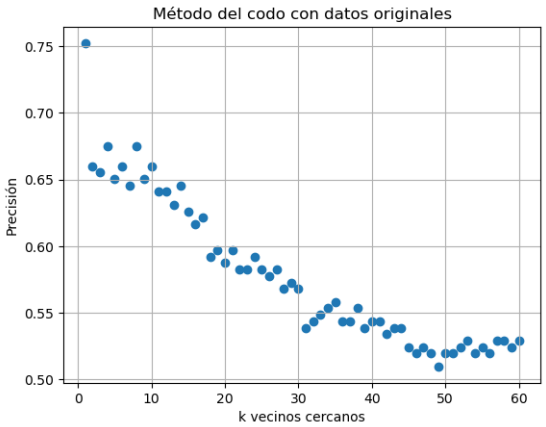


Figura 10: Precisión de los datos originales.

De igual manera en los datos completos normalizados y sin PCA, tuvieron un desempeño descendente (Figura 11). Por lo que tampoco fue posible dar con certeza una cantidad de vecinos que diera una mejor descripción del modelo.

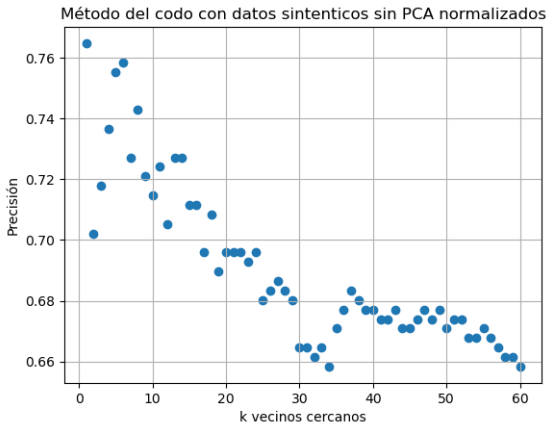


Figura 11: Precisión de los datos completos sin PCA y normalizados.

Por otro lado al aplicar el método del codo con los datos sintéticos con PCA y normalizados (véase Figura 12), se obtuvo un comportamiento más estable y con muchos mejores resultados. Obteniendo que puede realizarse un buen modelo con

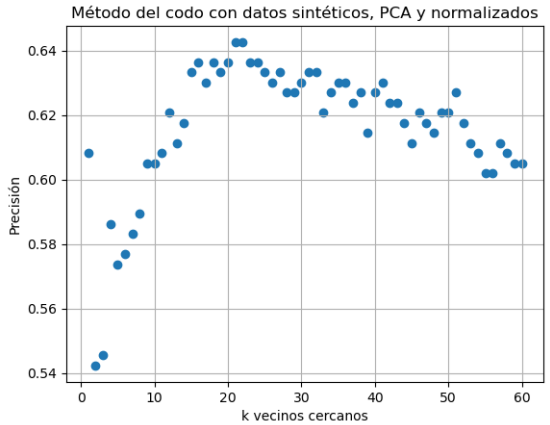


Figura 12: Precisión de los datos completos con PCA y normalizados.

Por lo que por la regla del codo se debe encontrar el valor de vecinos adecuado entre 17 y 22 aproximadamente. Una vez hecho esto, se realizará k-folds con $17 \leq k_n \leq 22$.

Una vez hecho esto, se procede a realizar pruebas de segregación 80 % entrenamiento (train) y 20 % pruebas (test), por lo que se obtuvo el resultado de las siguientes métricas en la Tabla 4.

Tabla 4: Resultado de reducción de dimensionalidad.

Número de vecinos	Exactitud	Precisión	Sensibilidad	F1 Score
17	0.63	0.61	0.62	0.6
18	0.64	0.62	0.64	0.61
19	0.63	0.61	0.63	0.61
20	0.64	0.62	0.64	0.62
21	0.64	0.62	0.64	0.62
22	0.64	0.63	0.64	0.62

Como se observa con la segregación 80-20, se tiene una estabilidad en el rango dado, lo que comprueba que dicho rango es adecuado para poder generalizar el modelo en ese caso. Cubriendo de buena manera los datos.

Finalmente se realizaron pruebas de validación cruzada para conocer si es que hay sesgos de información, es decir, que pueda haber información con la que aprenda mejor que otra en la Tabla 5.

Tabla 5: Resultado de reducción de dimensionalidad.

Numero de k-folds	Número de vecinos k	Exactitud	Precisión	Sensibilidad	F1 Score
3	17	0.3801	0.5526	0.3801	0.4042
5	17	0.3807	0.5517	0.3807	0.4075
7	17	0.3725	0.5419	0.3725	0.396
9	17	0.4346	0.6526	0.4346	0.46
3	19	0.4321	0.652	0.4321	0.4578
5	19	0.439	0.6517	0.439	0.4624
7	19	0.5422	0.6473	0.5422	0.5585
9	19	0.5347	0.6518	0.5347	0.5506
3	21	0.5428	0.6666	0.5428	0.5621
5	21	0.5372	0.6624	0.5372	0.558
7	21	0.5234	0.6449	0.5234	0.5392
9	21	0.5246	0.6751	0.5246	0.5417

Se observa que a pesar de que se encuentra la cantidad de vecinos en el rango de exactitud por la regla del codo, este cambia con respecto al numero de divisiones de la base de datos, esto debido a que al haber mas *folds* el conjunto de entrenamiento será mayor y por lo tanto tendrá más información para aprender y menos que predecir. Sin embargo, se observa que en el número de vecinos $k = 21$ se tiene un decremento conforme se incrementan los folds, suponiendo un sobre entrenamiento, lo cual no es deseado, por lo que con esta prueba, se observa que es **mejor trabajar con el rango de 17 a 19 vecinos** para obtener los resultados más fiables.

7. Conclusiones

La presente práctica mostro la aplicación ded un modelo Knn (k nearest neighbor) en donde se tiene un conjunto de datos de entrenamiento que son directamente parte del modelo y se miden las distancias del punto de prueba con respecto a todos los datos de entrenamiento, para obtener los k

puntos de entrenamiento menores en distancia y a partir de ello generar un voto plural para este problema multi clase para poder clasificar la dureza del concreto según su construcción y obtener la categoría constructiva en la que deberá utilizarse. Se realizaron balances de clases mediante la generación de datos sintéticos, incrementando en un 50 % de la base de datos. También realizó una normalización de tipo *minmax* y una reducción de dimensionalidad, lo que reprensento en un mejor comportamiento del modelo y por lo tanto, se obtuvo una mejor generalización del problema analizando las métricas y métodos para poder entender los datos de una mejor manera.

Finalmente se pudo concluir que el algoritmo de knn fue bastante sencillo de aplicar, sin embargo, computacionalmente es más costoso que otros algoritmos. A pesar de ello, es bastante útil para cuando se trabaja con datos etiquetados, el uso de datos sintéticos ayuda a balancear las clase eliminando muchos datos atípicos con ayuda del método SMOTE, el cual asegura que los datos se encontrarán dentro de las clases y hacer un aprendizaje más homogéneo y sin sesgos hacia alguna clase. Por la naturaleza del modelo, es fácil reentrenarlo pues únicamente se deberá actualizar el conjunto de datos de entrenamiento. A pesar de que se pudo observar una clara no linealidad entre atributos, fue posible obtener un modelo que pudo clasificar con una precisión y exactitud relativamente buena, puesto que si se tomara el peor de los casos, la exactitud debería ser de un 25 % como minino si es que siempre obtuviera la misma salida. Fue posible observar un aprendizaje del modelo para poder realizar las clasificaciones con cierta certeza, por lo que se puede decir que tal vez no sea la mejor técnica para clasificar este conjunto de datos, pero puede ser un punto de partida bastante interesante y avanzado.

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] 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).

[5] "Descubra el algoritmo knn : un algoritmo de aprendizaje supervisado." https://datascientest.com/es/que-es-el-algoritmo-knn. (Accessed on 06/01/2023).

[6] J. Chouinard, "k-nearest neighbors (knn) in python." https://www.jcchouinard.com/k-nearest-neighbors/, May 2022. (Accessed on 05/29/2023).

[7] M. REDA, "Knn & precision and recall." https://www.kaggle.com/code/mahmoudreda55/knn-precision-and-recall, 2021. (Accessed on 06/01/2023).

[8] I.-C. Yeh, "Modeling of strength of high-performance concrete using artificial neural networks."." https://www.researchgate.net/publication/222447231_Modeling_of_Strength_of_High-Performance_Concrete_Using_Artificial_Neural_Networks_Cement_and_Concrete_research_2812_1797-1808, 1998. (Accessed on 05/27/2023).

Practica 5 KNN Aldo Cervantes

June 2, 2023

1 Algoritmo KNN

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

```
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}"
```

1.1 Descripción de la base de datos

About Dataset Context Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate.

Content Data Characteristics:

The actual concrete compressive strength (MPa) for a given mixture under a specific age (days) was determined from laboratory. Data is in raw form (not scaled).

Summary Statistics:

Number of instances (observations): 1030 Number of Attributes: 9 Attribute breakdown: 8 quantitative input variables, and 1 quantitative output variable Missing Attribute Values: None

Variable Information:

Given is the variable name, variable type, the measurement unit and a brief description. The concrete compressive strength is the regression problem. The order of this listing corresponds to the order of numerals along the rows of the database.

- Name – Data Type – Measurement – Description

1. Cement (component 1) – quantitative – kg in a m3 mixture – Input Variable
2. Blast Furnace Slag (component 2) – quantitative – kg in a m3 mixture – Input Variable
3. Fly Ash (component 3) – quantitative – kg in a m3 mixture – Input Variable
4. Water (component 4) – quantitative – kg in a m3 mixture – Input Variable
5. Superplasticizer (component 5) – quantitative – kg in a m3 mixture – Input Variable
6. Coarse Aggregate (component 6) – quantitative – kg in a m3 mixture – Input Variable

7. Fine Aggregate (component 7) – quantitative – kg in a m3 mixture – Input Variable
8. Age – quantitative – Day (1~365) – Input Variable
9. Concrete compressive strength – quantitative – MPa – Output Variable

[2]:
```
data=pd.read_csv('concrete_data.csv')
data.columns
```

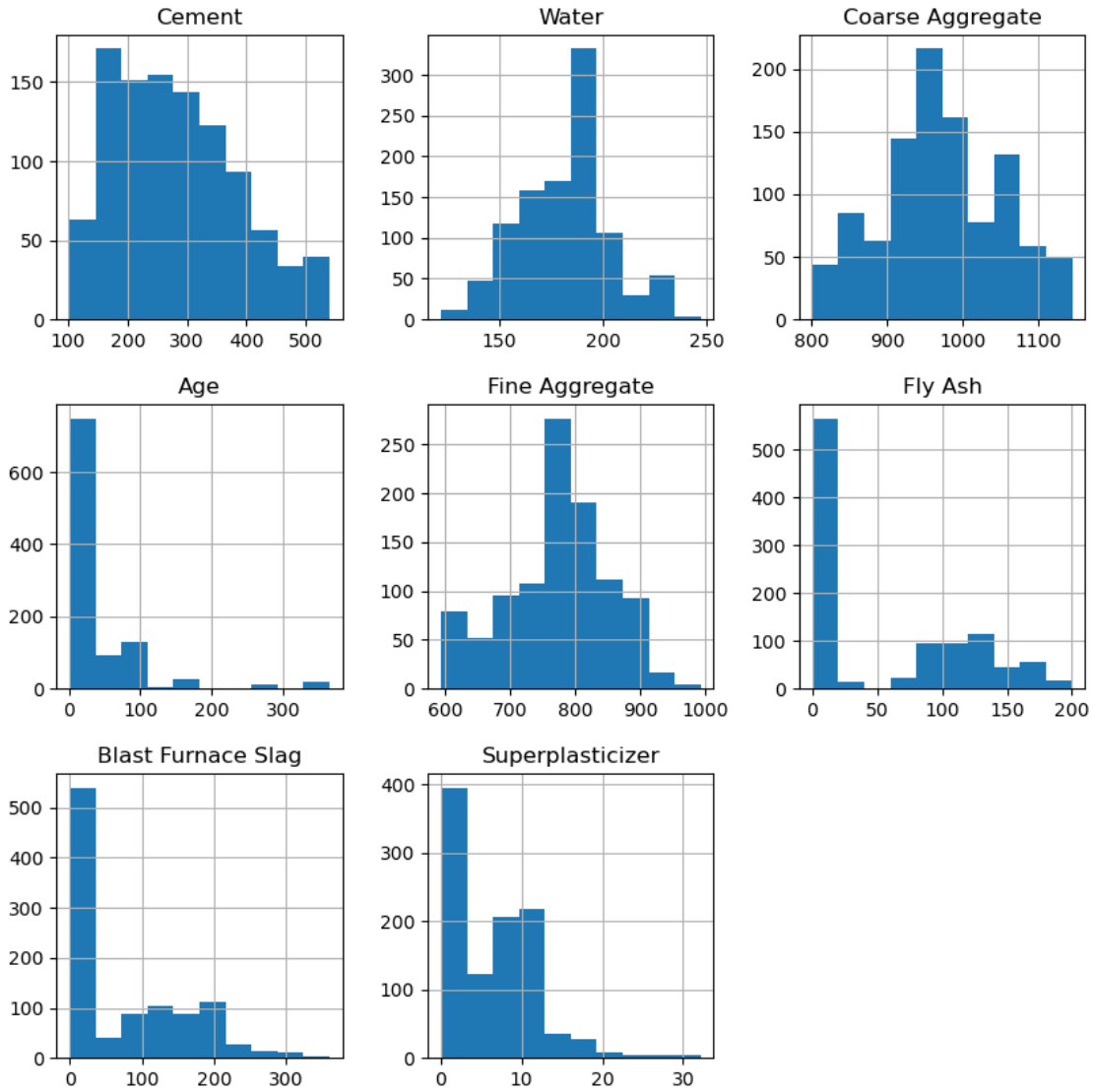
[2]:
```
Index(['Cement', 'Blast Furnace Slag', 'Fly Ash', 'Water', 'Superplasticizer',
       'Coarse Aggregate', 'Fine Aggregate', 'Age', 'Strength'],
      dtype='object')
```

[3]:
```
#data=data[data['Age']<=28]
data=data[['Cement','Water','Coarse Aggregate','Age','Fine Aggregate','Fly␣
↪Ash','Blast Furnace Slag','Superplasticizer','Strength']]
```

[4]:
```
fig = plt.figure(figsize = (10,10))
ax = fig.gca()
data[[columna for columna in data.columns if columna != 'Strength']].hist(ax=ax)
```

```
C:\Users\aldoa\AppData\Local\Temp\ipykernel_34316\3595810033.py:3: UserWarning:
To output multiple subplots, the figure containing the passed axes is being
cleared.
  data[[columna for columna in data.columns if columna !=
'Strength']].hist(ax=ax)
```

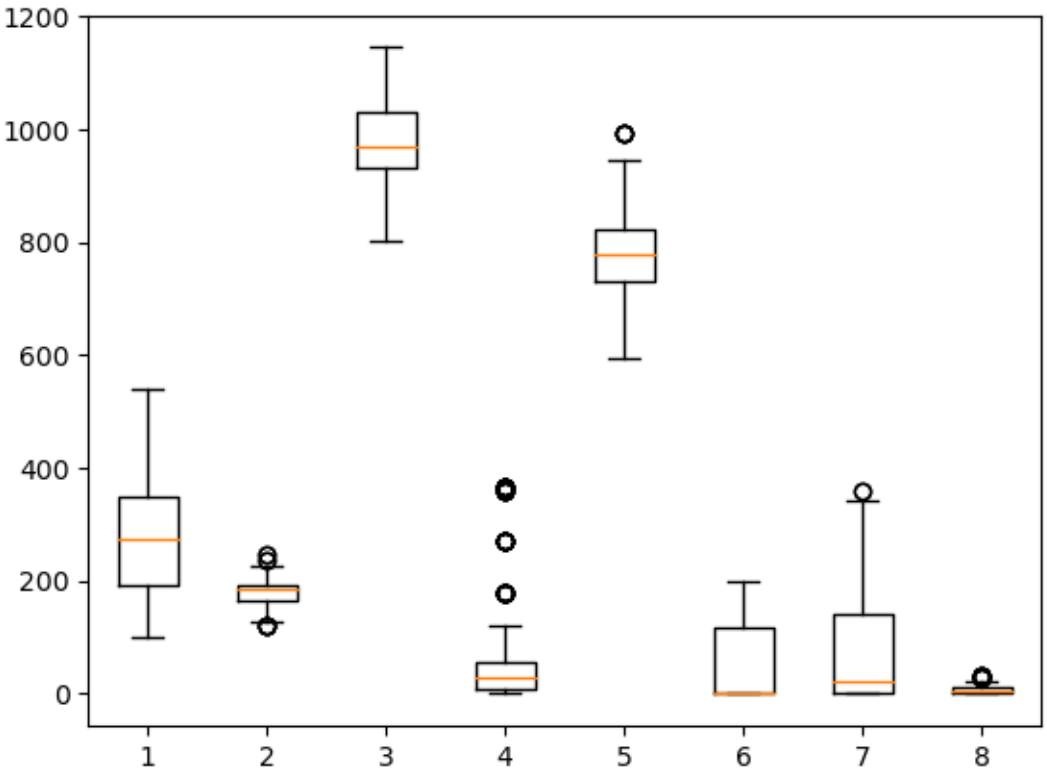
[4]:
```
array([[<AxesSubplot:title={'center':'Cement'}>,
        <AxesSubplot:title={'center':'Water'}>,
        <AxesSubplot:title={'center':'Coarse Aggregate'}>],
       [<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Fine Aggregate'}>,
        <AxesSubplot:title={'center':'Fly Ash'}>],
       [<AxesSubplot:title={'center':'Blast Furnace Slag'}>,
        <AxesSubplot:title={'center':'Superplasticizer'}>,
        <AxesSubplot:>]], dtype=object)
```



[5]:
```
plt.boxplot(data[[columna for columna in data.columns if columna != 'Strength']])
```

[5]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x22647ca60a0>,
  <matplotlib.lines.Line2D at 0x22647ca6370>,
  <matplotlib.lines.Line2D at 0x22647cb44c0>,
  <matplotlib.lines.Line2D at 0x22647cb4790>,
  <matplotlib.lines.Line2D at 0x22647cc28b0>,
  <matplotlib.lines.Line2D at 0x22647cc2b80>,
  <matplotlib.lines.Line2D at 0x22647cceca0>,
  <matplotlib.lines.Line2D at 0x22647ccef70>,
  <matplotlib.lines.Line2D at 0x22647cea0d0>,
  <matplotlib.lines.Line2D at 0x22647cea3a0>,
  <matplotlib.lines.Line2D at 0x22647cf44c0>,
```

```
 <matplotlib.lines.Line2D at 0x22647cf4790>,
 <matplotlib.lines.Line2D at 0x22647d018b0>,
 <matplotlib.lines.Line2D at 0x22647d01b80>,
 <matplotlib.lines.Line2D at 0x22647d0fca0>,
 <matplotlib.lines.Line2D at 0x22647d0ff70>],
'caps': [<matplotlib.lines.Line2D at 0x22647ca6640>,
 <matplotlib.lines.Line2D at 0x22647ca6910>,
 <matplotlib.lines.Line2D at 0x22647cb4a60>,
 <matplotlib.lines.Line2D at 0x22647cb4d30>,
 <matplotlib.lines.Line2D at 0x22647cc2e50>,
 <matplotlib.lines.Line2D at 0x22647cce160>,
 <matplotlib.lines.Line2D at 0x22647cda280>,
 <matplotlib.lines.Line2D at 0x22647cda550>,
 <matplotlib.lines.Line2D at 0x22647cea670>,
 <matplotlib.lines.Line2D at 0x22647cea940>,
 <matplotlib.lines.Line2D at 0x22647cf4a60>,
 <matplotlib.lines.Line2D at 0x22647cf4d30>,
 <matplotlib.lines.Line2D at 0x22647d01e50>,
 <matplotlib.lines.Line2D at 0x22647d0f160>,
 <matplotlib.lines.Line2D at 0x22647d1b280>,
 <matplotlib.lines.Line2D at 0x22647d1b550>],
'boxes': [<matplotlib.lines.Line2D at 0x22647c8fd90>,
 <matplotlib.lines.Line2D at 0x22647cb41f0>,
 <matplotlib.lines.Line2D at 0x22647cc25e0>,
 <matplotlib.lines.Line2D at 0x22647cce9d0>,
 <matplotlib.lines.Line2D at 0x22647cdadc0>,
 <matplotlib.lines.Line2D at 0x22647cf41f0>,
 <matplotlib.lines.Line2D at 0x22647d015e0>,
 <matplotlib.lines.Line2D at 0x22647d0f9d0>],
'medians': [<matplotlib.lines.Line2D at 0x22647ca6be0>,
 <matplotlib.lines.Line2D at 0x22647cc2040>,
 <matplotlib.lines.Line2D at 0x22647cce430>,
 <matplotlib.lines.Line2D at 0x22647cda820>,
 <matplotlib.lines.Line2D at 0x22647ceac10>,
 <matplotlib.lines.Line2D at 0x22647d01040>,
 <matplotlib.lines.Line2D at 0x22647d0f430>,
 <matplotlib.lines.Line2D at 0x22647d1b820>],
'fliers': [<matplotlib.lines.Line2D at 0x22647ca6eb0>,
 <matplotlib.lines.Line2D at 0x22647cc2310>,
 <matplotlib.lines.Line2D at 0x22647cce700>,
 <matplotlib.lines.Line2D at 0x22647cdaaf0>,
 <matplotlib.lines.Line2D at 0x22647ceaee0>,
 <matplotlib.lines.Line2D at 0x22647d01310>,
 <matplotlib.lines.Line2D at 0x22647d0f700>,
 <matplotlib.lines.Line2D at 0x22647d1baf0>],
'means': []}
```



[6]:
```
data['Strength'].describe()
```

[6]:
```
count    1030.000000
mean       35.817961
std        16.705742
min         2.330000
25%        23.710000
50%        34.445000
75%        46.135000
max        82.600000
Name: Strength, dtype: float64
```

1.2 Nueva nomenclatura.

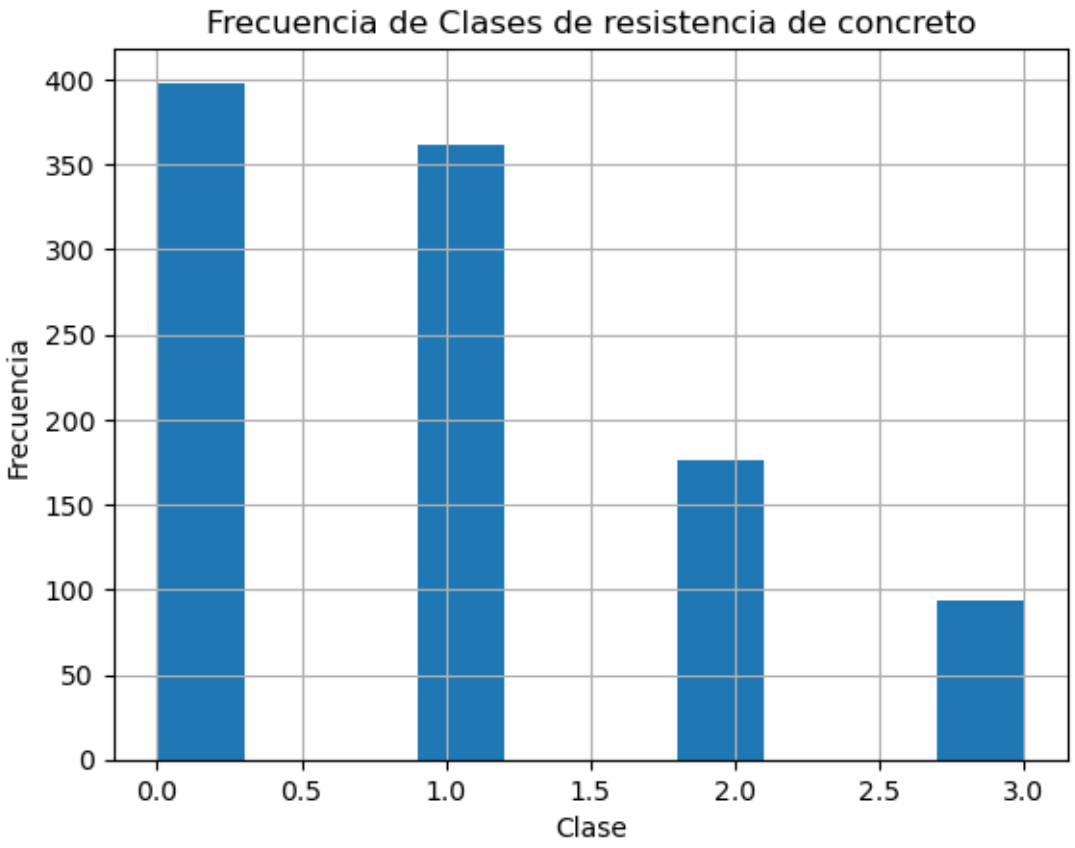
Se realizó una categorización de los concretos según su uso como se observa en PSI construcción y se encontrarón principalmente los siguientes tipos:

- Concreto pobre o de baja resistencia: $H_{<30}$ $H_x \leq 306\ kg/cm^2$
- Concreto de uso convencional:$H_{[30,45]}$ $306 > H_x \leq 458\ kg/cm^2$
- Concreto de alta resistencia: $H_{[45,60]}$, $458 > H_x \leq 611\ kg/cm^2$
- Concreto de ultra alta resistencia: $H_{>60}$, $H_x > 611\ kg/cm^2$

[7]:
```
condiciones = [
    data['Strength'] <= 30,
    (data['Strength'] > 30) & (data['Strength'] <= 45),
    (data['Strength'] > 45) & (data['Strength'] <= 60),
    data['Strength'] > 60
]
valores = [0, 1, 2, 3]

data['Strength']=np.select(condiciones,valores)
```

[8]:
```
data['Strength'].hist()
valores=data['Strength'].value_counts()
plt.title('Frecuencia de Clases de resistencia de concreto')
plt.xlabel('Clase')
plt.ylabel('Frecuencia')
for aa in range(len(valores)-1):
    temp=valores[0]-valores[aa+1]
    print('Cantidad de desbalance con clase ',temp)
    porct=(100*temp)/valores[0]
    print('porcentaje de desbalance ',porct)
```

```
Cantidad de desbalance con clase  36
porcentaje de desbalance  9.045226130653266
Cantidad de desbalance con clase  222
porcentaje de desbalance  55.778894472361806
Cantidad de desbalance con clase  304
porcentaje de desbalance  76.38190954773869
```



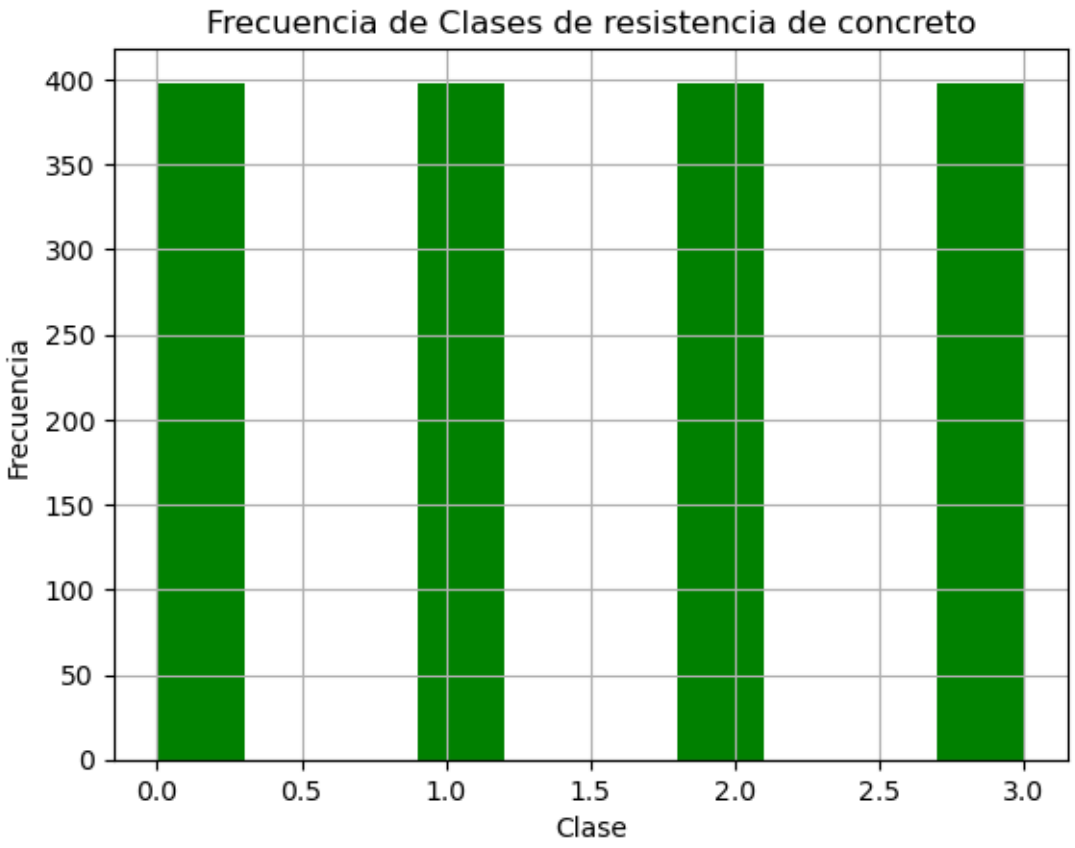
[9]:
```
data_eq=pd.DataFrame(data)
data3=pd.DataFrame(data)
data_raw=pd.DataFrame(data)
def euc_dis(x1,x2):
    return np.sqrt((x1-x2)**2)

def e_dis_abs(x1,x2):
    #r,c=x1.shape
    x3=(x1-x2)**2
    x4=np.sqrt(sum(x3))
    return x4


def smotev1(minclass,val_class,ndg):
    ## Tomar dos valores aleatorios
    np.random.seed(734289) ###################### 220
    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['Strength']=val_class
    return df
```

[10]:
```
max_class=data_eq[data_eq['Strength']==0]
min_class1=data_eq[data_eq['Strength']==1]
min_class2=data_eq[data_eq['Strength']==2]
min_class3=data_eq[data_eq['Strength']==3]
minclases=[min_class1,min_class2,min_class3]
bal_class=[]
ndf=[36,222,304]
for aa in range(3):
    bal_class.append(smotev1(minclases[aa],aa+1,ndf[aa]))
    data_eq=pd.concat([data_eq,bal_class[-1]], ignore_index=True)
#print(bal_class[0].shape)
```

[11]:
```
data_eq['Strength'].hist(color=['green'])
plt.title('Frecuencia de Clases de resistencia de concreto')
plt.xlabel('Clase')
plt.ylabel('Frecuencia')
print(data_eq.shape)
```

```
(1592, 9)
```



[12]:
```
data_eq.isnull().sum()
print('No hay valores faltantes')
```

```
No hay valores faltantes
```

2 Realiza analisis de componentes principales con y sin generación de datos sintenticos.

[13]:
```
def covarianza(X,Y):
    x_mean = X.mean()
    y_mean = Y.mean()
    n = len(X)
    cov = (((X-x_mean)*(Y-y_mean)).sum())/(n-1)
    return cov

def matriz_cov(data):
    atributos = data.columns
    n = len(atributos)
    m = np.zeros((n,n))
```

```
    for i in range(n):
        for j in range(n):
            X = data[atributos[i]]
            Y = data[atributos[j]]
            m[i][j] = covarianza(X,Y)
    return m

def pca1(data): # Ya tiene que estar normalizado
    atributos = list(data.columns)
    n = len(atributos)
    for a in atributos:
        data[a]=data[a]-data[a].mean()
    #m_cov=matriz_cov(data)
    m_cov=np.cov(np.transpose(data))
    #sns.heatmap(m_cov,annot=True)
    eival,eivec=np.linalg.eig(m_cov)
    p=(eival/eival.sum())*100
    return p,eivec

def ordenar(porcentajes,columnas):
    n=len(columnas)
    p2=porcentajes.copy()
    porcentaje=np.sort(porcentajes)[::-1]
    dicti={}
    for a in range(n):
        ub=int(np.where(porcentaje[a]==p2)[0])
        dicti[columnas[ub]]=porcentaje[a]
    return dicti
```

[14]:
```
#### Normalización

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

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

[15]:
```
#data=data[data['Age']<28]


print(data_eq.shape)
data2=pd.DataFrame(data_eq)
data_eq=norm_min_max(data_eq[[columna for columna in data_eq.columns if columna !
 ↪= 'Strength']],0,1)
porcentaje,eivec=pca1(data_eq[[columna for columna in data_eq.columns if columna␣
 ↪!= 'Strength']])
por=ordenar(porcentaje,[columna for columna in data_eq.columns if columna !=␣
 ↪'Strength'])
print(por)

print(data.shape)
```

```
(1592, 9)
{'Cement': 31.185943733989003, 'Water': 20.382246119394278, 'Coarse Aggregate':
18.360425124724383, 'Blast Furnace Slag': 11.324978732869177,
'Superplasticizer': 10.317974274027831, 'Fly Ash': 6.196758033453855, 'Fine
Aggregate': 1.9010783933969773, 'Age': 0.330595588144499}
(1030, 9)
```

2.1 5 Componentes principales, suman 91.55%

{‘Cement’: 31.185943733989003, ‘Water’: 20.382246119394278, ‘Coarse Aggregate’: 18.360425124724383, ‘Blast Furnace Slag’: 11.324978732869177, ‘Superplasticizer’: 10.317974274027831, ‘Fly Ash’: 6.196758033453855, ‘Fine Aggregate’: 1.9010783933969773, ‘Age’: 0.330595588144499}

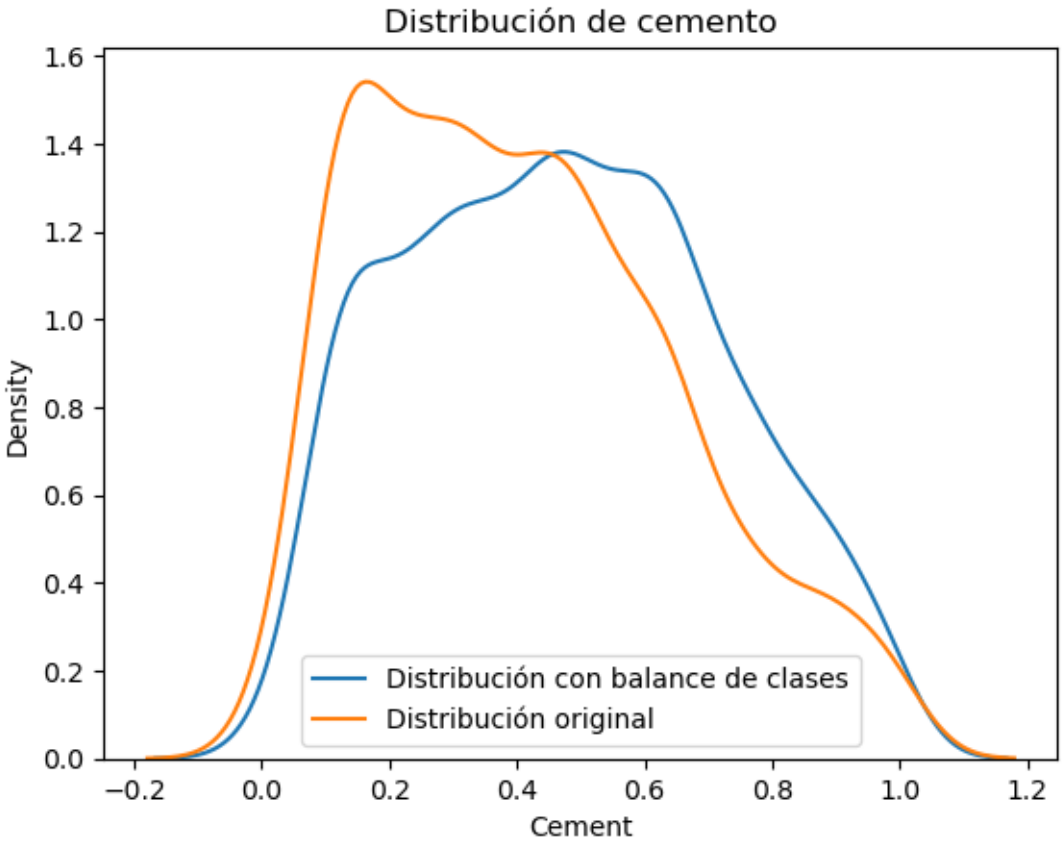
[16]:
```
#data.shape
datan_pca=data_eq[['Cement','Water','Coarse Aggregate', 'Blast Furnace Slag',␣
 ↪'Superplasticizer']]
datan_pca=pd.concat([datan_pca,data2['Strength']],axis=1)

print(datan_pca.shape)
print(data.shape)
```

```
(1592, 6)
(1030, 9)
```

[17]:
```
data=norm_min_max(data[[columna for columna in data_eq.columns if columna !=␣
↪'Strength']],0,1)
```

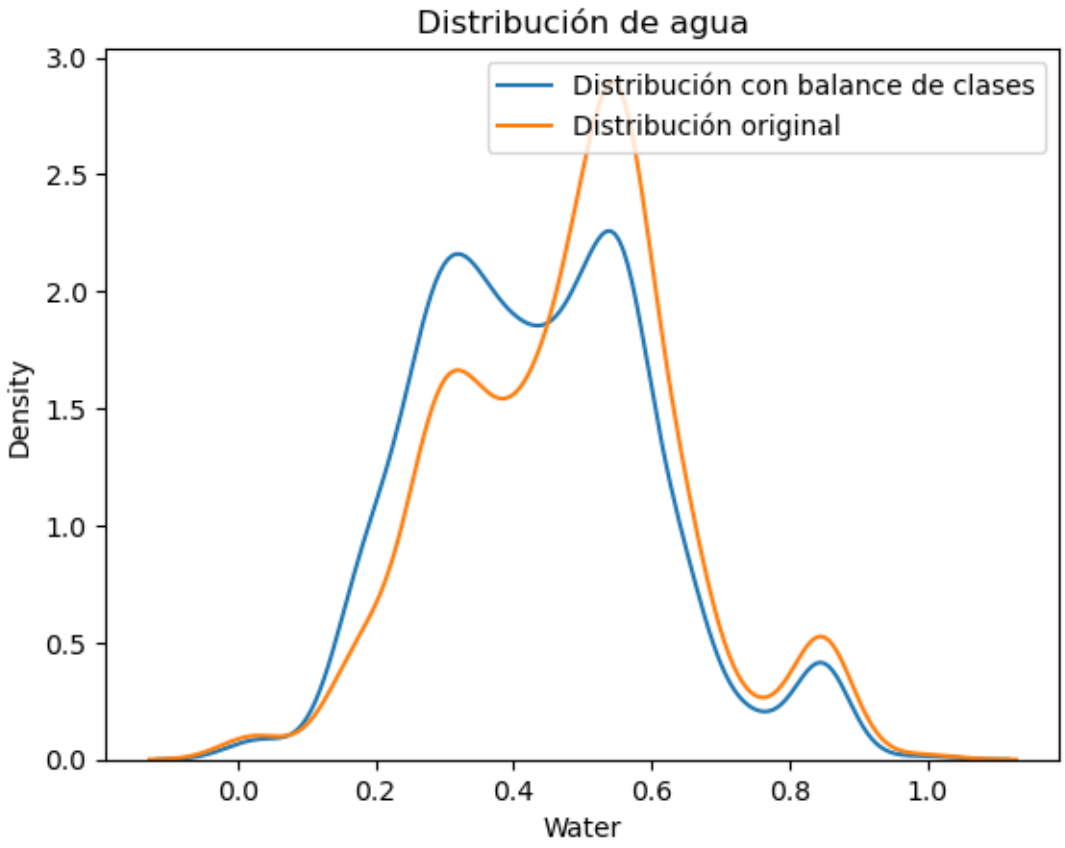
[18]:
```
sns.kdeplot(datan_pca['Cement'])
sns.kdeplot(data['Cement'])
plt.legend(['Distribución con balance de clases','Distribución original'])
plt.title('Distribución de cemento')
```

[18]: Text(0.5, 1.0, 'Distribución de cemento')



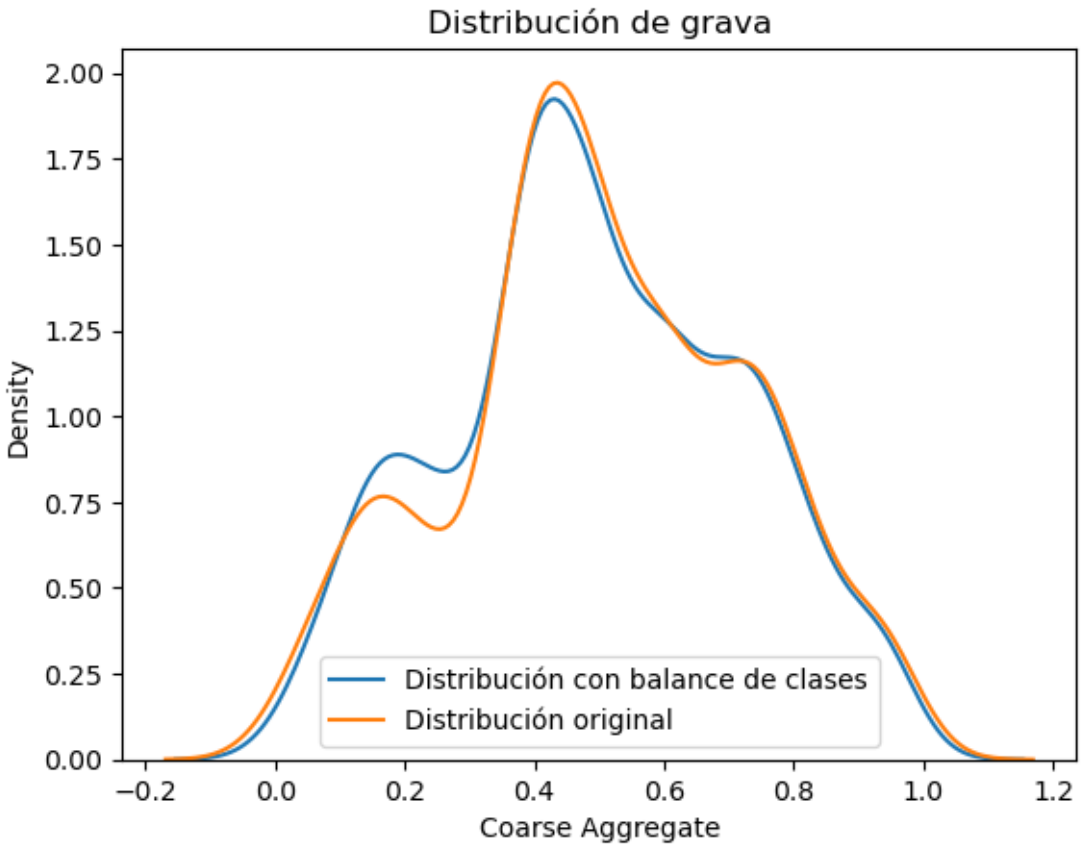
[19]:
```
sns.kdeplot(datan_pca['Water'])
sns.kdeplot(data['Water'])
plt.legend(['Distribución con balance de clases','Distribución original'])
plt.title('Distribución de agua')
```

[19]: Text(0.5, 1.0, 'Distribución de agua')



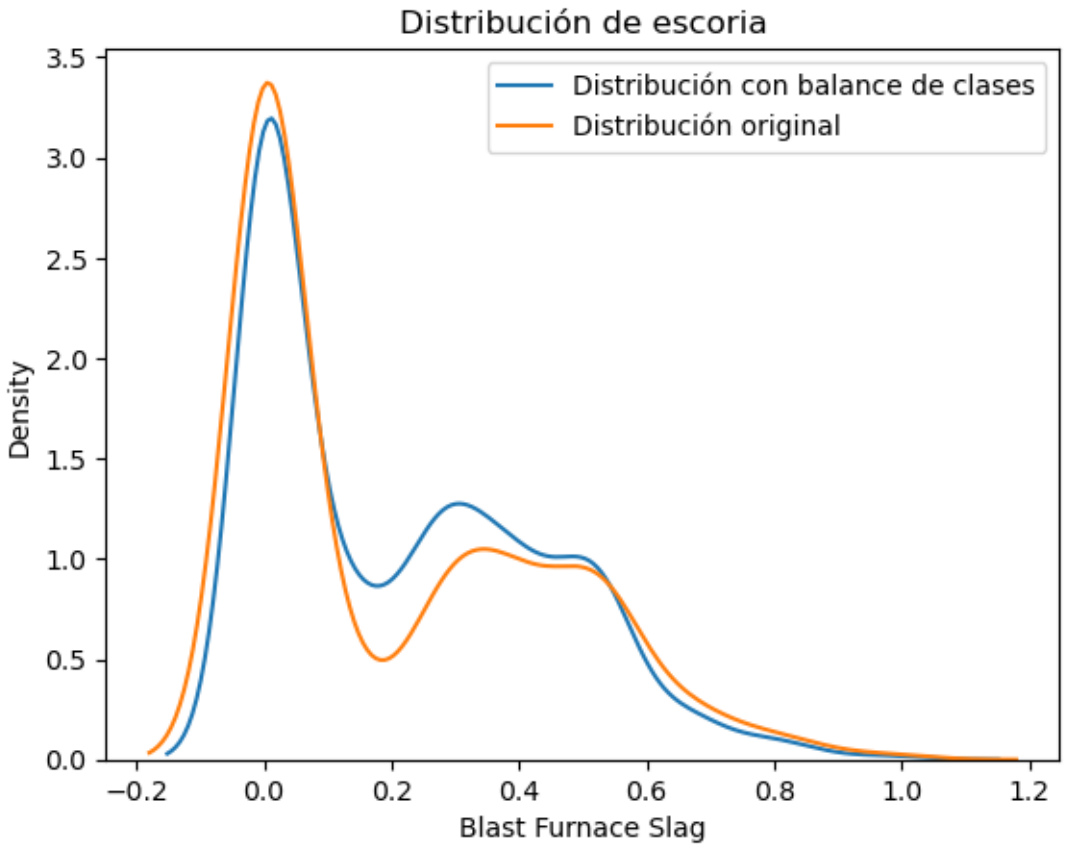
[20]:
```
sns.kdeplot(datan_pca['Coarse Aggregate'])
sns.kdeplot(data['Coarse Aggregate'])
plt.legend(['Distribución con balance de clases','Distribución original'])
plt.title('Distribución de grava')
```

[20]: Text(0.5, 1.0, 'Distribución de grava')



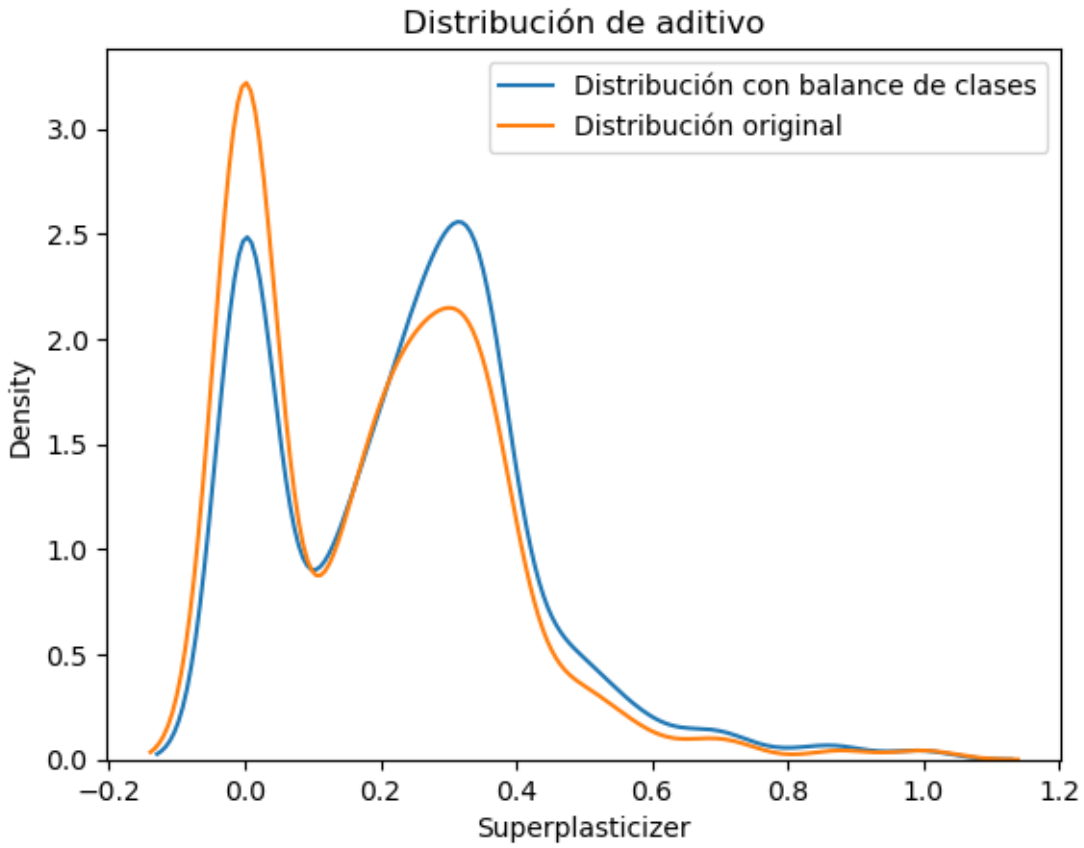
[21]:
```
sns.kdeplot(datan_pca['Blast Furnace Slag'])
sns.kdeplot(data['Blast Furnace Slag'])
plt.legend(['Distribución con balance de clases','Distribución original'])
plt.title('Distribución de escoria')
```

[21]:
```
Text(0.5, 1.0, 'Distribución de escoria')
```



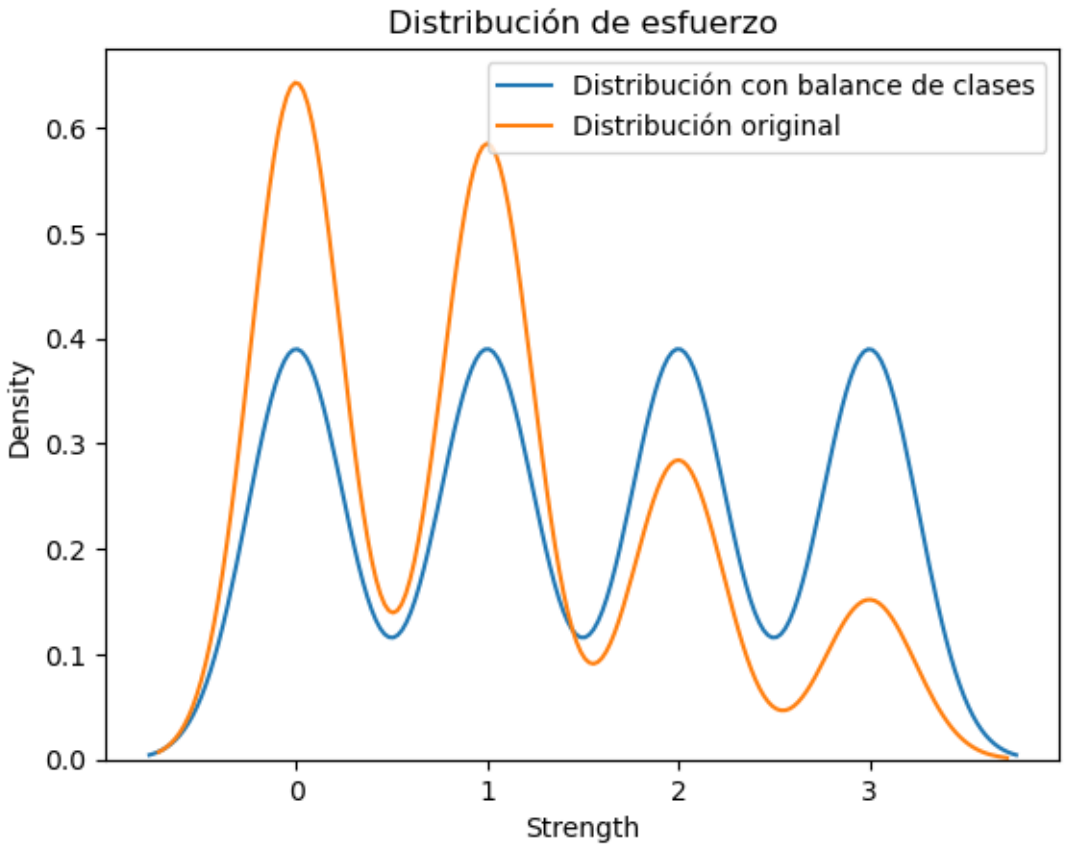
[22]:
```
sns.kdeplot(datan_pca['Superplasticizer'])
sns.kdeplot(data['Superplasticizer'])
plt.legend(['Distribución con balance de clases','Distribución original'])
plt.title('Distribución de aditivo')
```

[22]:
```
Text(0.5, 1.0, 'Distribución de aditivo')
```



[23]:
```
sns.kdeplot(datan_pca['Strength'])
#data3=pd.read_csv('concrete_data.csv')
sns.kdeplot(data3['Strength'])
plt.legend(['Distribución con balance de clases','Distribución original'])
plt.title('Distribución de esfuerzo')
```

[23]:
```
Text(0.5, 1.0, 'Distribución de esfuerzo')
```



[24]:
```
datan_pca.describe()
```

[24]:

	Cement	Water	Coarse Aggregate	Blast Furnace Slag	\
count	1592.000000	1592.000000	1592.000000	1592.000000	
mean	0.472944	0.436815	0.493001	0.216953	
std	0.241549	0.169346	0.223704	0.220972	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.271233	0.305493	0.364525	0.000000	
50%	0.467208	0.428914	0.482558	0.153657	
75%	0.649252	0.560703	0.661047	0.386917	
max	1.000000	1.000000	1.000000	1.000000	

	Superplasticizer	Strength
count	1592.000000	1592.000000
mean	0.232342	1.500000
std	0.188279	1.118385
min	0.000000	0.000000
25%	0.000000	0.750000
50%	0.248447	1.500000
75%	0.347826	2.250000

```
max            1.000000     3.000000
```

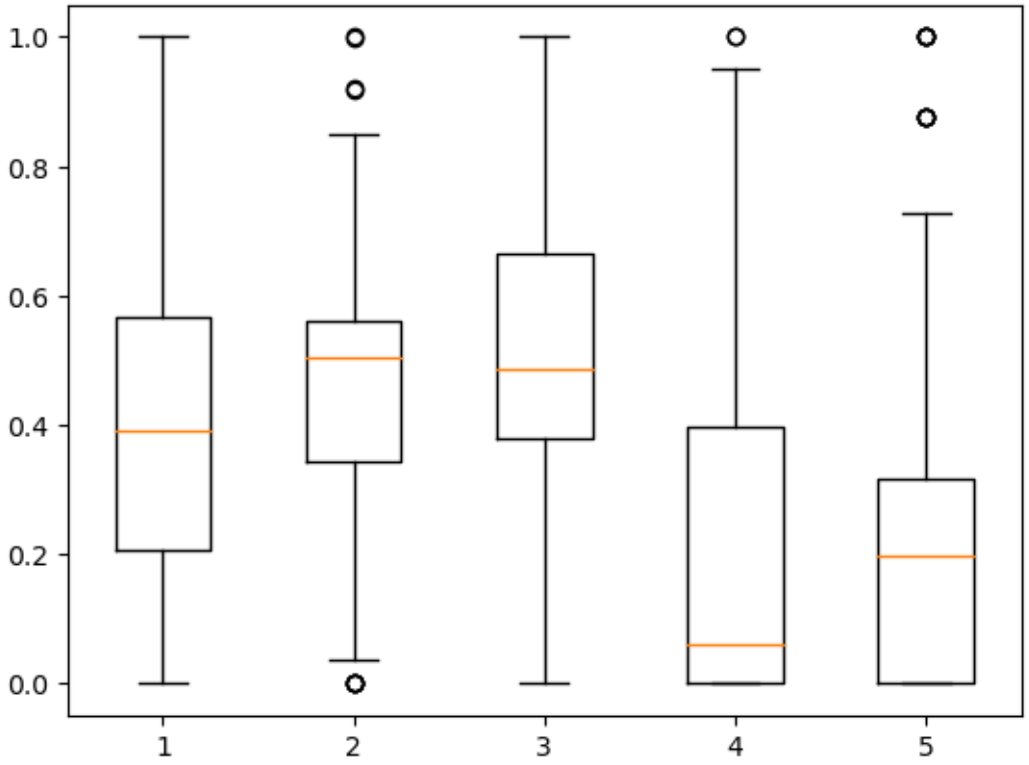
[25]:
```
data3[['Cement','Water','Coarse Aggregate', 'Blast Furnace Slag',␣
 ↪'Superplasticizer','Strength']].describe()
```

[25]:
```
            Cement        Water  Coarse Aggregate  Blast Furnace Slag  \
count  1030.000000  1030.000000       1030.000000         1030.000000
mean    281.167864   181.567282        972.918932           73.895825
std     104.506364    21.354219         77.753954           86.279342
min     102.000000   121.800000        801.000000            0.000000
25%     192.375000   164.900000        932.000000            0.000000
50%     272.900000   185.000000        968.000000           22.000000
75%     350.000000   192.000000       1029.400000          142.950000
max     540.000000   247.000000       1145.000000          359.400000

       Superplasticizer     Strength
count       1030.000000  1030.000000
mean           6.204660     0.966990
std            5.973841     0.960279
min            0.000000     0.000000
25%            0.000000     0.000000
50%            6.400000     1.000000
75%           10.200000     2.000000
max           32.200000     3.000000
```

[26]:
```
plt.boxplot(data[[columna for columna in datan_pca.columns if columna !=␣
 ↪'Strength']])
```

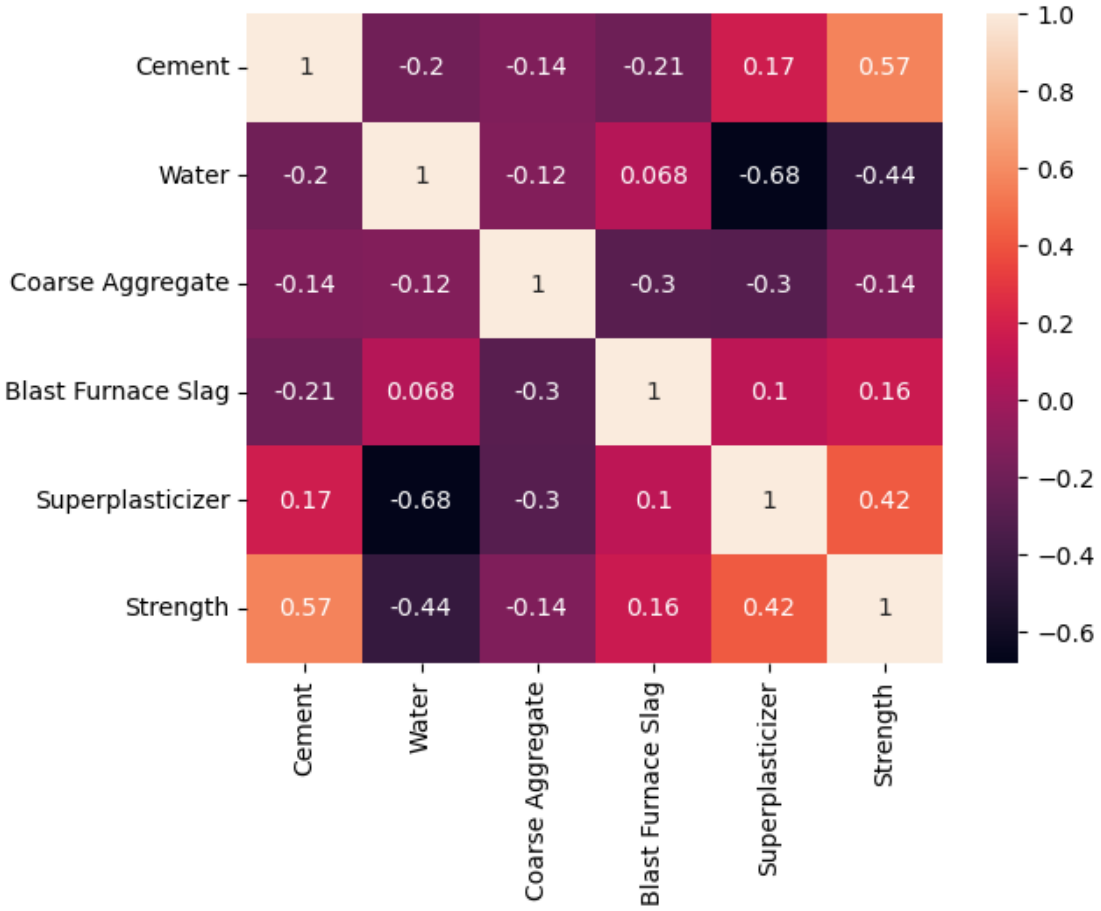
[26]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x22649dc6490>,
  <matplotlib.lines.Line2D at 0x22649dc6760>,
  <matplotlib.lines.Line2D at 0x22649dd1880>,
  <matplotlib.lines.Line2D at 0x22649dd1b50>,
  <matplotlib.lines.Line2D at 0x22649ddfc70>,
  <matplotlib.lines.Line2D at 0x22649ddff40>,
  <matplotlib.lines.Line2D at 0x22649df80d0>,
  <matplotlib.lines.Line2D at 0x22649df83a0>,
  <matplotlib.lines.Line2D at 0x22649e064c0>,
  <matplotlib.lines.Line2D at 0x22649e06790>],
 'caps': [<matplotlib.lines.Line2D at 0x22649dc6a30>,
  <matplotlib.lines.Line2D at 0x22649dc6d00>,
  <matplotlib.lines.Line2D at 0x22649dd1e20>,
  <matplotlib.lines.Line2D at 0x22649ddf130>,
  <matplotlib.lines.Line2D at 0x22649dec250>,
  <matplotlib.lines.Line2D at 0x22649dec520>,
  <matplotlib.lines.Line2D at 0x22649df8670>,
  <matplotlib.lines.Line2D at 0x22649df8940>,
  <matplotlib.lines.Line2D at 0x22649e06a60>,
```

```
 <matplotlib.lines.Line2D at 0x22649e06d30>],
'boxes': [<matplotlib.lines.Line2D at 0x22649dc62e0>,
 <matplotlib.lines.Line2D at 0x22649dd15b0>,
 <matplotlib.lines.Line2D at 0x22649ddf9a0>,
 <matplotlib.lines.Line2D at 0x22649decd90>,
 <matplotlib.lines.Line2D at 0x22649e061f0>],
'medians': [<matplotlib.lines.Line2D at 0x22649dc6fd0>,
 <matplotlib.lines.Line2D at 0x22649ddf400>,
 <matplotlib.lines.Line2D at 0x22649dec7f0>,
 <matplotlib.lines.Line2D at 0x22649df8c10>,
 <matplotlib.lines.Line2D at 0x22649e13040>],
'fliers': [<matplotlib.lines.Line2D at 0x22649dd12e0>,
 <matplotlib.lines.Line2D at 0x22649ddf6d0>,
 <matplotlib.lines.Line2D at 0x22649decac0>,
 <matplotlib.lines.Line2D at 0x22649df8ee0>,
 <matplotlib.lines.Line2D at 0x22649e13310>],
'means': []}
```



[27]:
```
### Correlacion entre valores
sns.heatmap(datan_pca[:-1].corr(),annot=True)
```

[27]: <AxesSubplot:>



[28]:
```
# entrenamiento con 80/20
np.random.seed(4567)
entrenamiento80=datan_pca.sample(n=int(datan_pca.shape[0]*0.8))
prueba20=datan_pca[~datan_pca.index.isin(entrenamiento80.index)] # [~datan_pca.
↪index.isin(entrenamiento80.index)]
```

[29]:
```
entrenamiento80.shape
#entrenamiento_target=entrenamiento80.pop('Strength')
```

[29]:
```
(1273, 6)
```

[30]:
```
entrenamiento80.tail()
```

[30]:

	Cement	Water	Coarse Aggregate	Blast Furnace Slag	\
1569	0.874165	0.430926	0.806825	0.200503	
1405	0.998387	0.320837	0.737090	0.000206	
231	0.255023	0.478435	0.769767	0.272955	
1582	0.604761	0.149332	0.281088	0.155437	

```
37    0.526256  0.848243          0.380814            0.396494

      Superplasticizer  Strength
1569          0.081508         3
1405          0.078658         3
231           0.214286         2
1582          0.529827         3
37            0.000000         1
```

[31]:
```
prueba20.shape
#prueba_target=prueba20.pop('Strength')
```

[31]:
```
(319, 6)
```

[32]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score

def elbow_m(dist,n,columna):
    val_min=dist.nsmallest(n,columna)
    #index_min=val_min.index
    return np.sum(val_min)/n


def mod_knn(train,pr,n):
    rent,colt=train.shape
    renpr,colpr=pr.shape
    tr=pd.DataFrame(train)
    proof=pd.DataFrame(pr)
    prediccion=pd.DataFrame(np.zeros((renpr,1)),columns=['prediccion'])
    #train_target=tr.pop('Strength')
    #pr_target=proof.pop('Strength')
    e_met=[]
    problema=0
    dist_temp=pd.DataFrame(np.zeros((rent,1)),columns=['Dist'])
    for aa in range(renpr):
        for bb in range(rent):
            dist_temp.iloc[bb,0]=e_dis_abs(proof.iloc[aa,:-1],tr.iloc[bb,:-1])

        mindis=dist_temp.nsmallest(n,'Dist')
        indmin=mindis.index
        recuento=tr.iloc[indmin,-1].value_counts()
        cuenta_maxima = recuento.max()
        categorias_mas_frecuentes = recuento[recuento == cuenta_maxima].index.
↪tolist()
        if len(categorias_mas_frecuentes)>1:
            problema+=1
        #    cat=np.zeros((n,1))
```

```
        #    for xx in range(n): # vecinos cercanos que causan problema
        #        cat[tr.iloc[indmin[xx],-1],0]+=dist_temp.iloc[indmin[xx],0]
        #
        #    prediccion.iloc[aa,:]=np.argmin(cat)
        #else:
        prediccion.iloc[aa,:]=recuento.idxmax()
        """
        ### Ya funciono#################
        #e_met.append(elbow_m(dist_temp,n,'Dist'))
        mindis=dist_temp.nsmallest(n,'Dist')
        indmin=mindis.index
        recuento=tr.iloc[indmin,-1].value_counts()
        #recuento=mindis['Dist'].value_counts()
        prediccion.iloc[aa,:]=recuento.idxmax()
        """
    print(problema)
    return prediccion.astype(int)
```

[33]:
```
############################ Método del codo ###########
"""
import time

t1=time.time()
elbow=[]
n_range=[]
ot=[]
for o in range(30):
    resp=mod_knn(entrenamiento80,prueba20,o+1)
    elbow.append(accuracy_score(prueba20.iloc[:,-1],resp))
    ot.append(o+1)
t2=time.time()

plt.plot(ot,elbow)


#plt.plot(elbow)
#plt.scatter(6,elbow[6],color='red')
#plt.title('Método del codo ')
#plt.xlabel('n Vecinos')
#plt.ylabel('Distancia')
"""
```

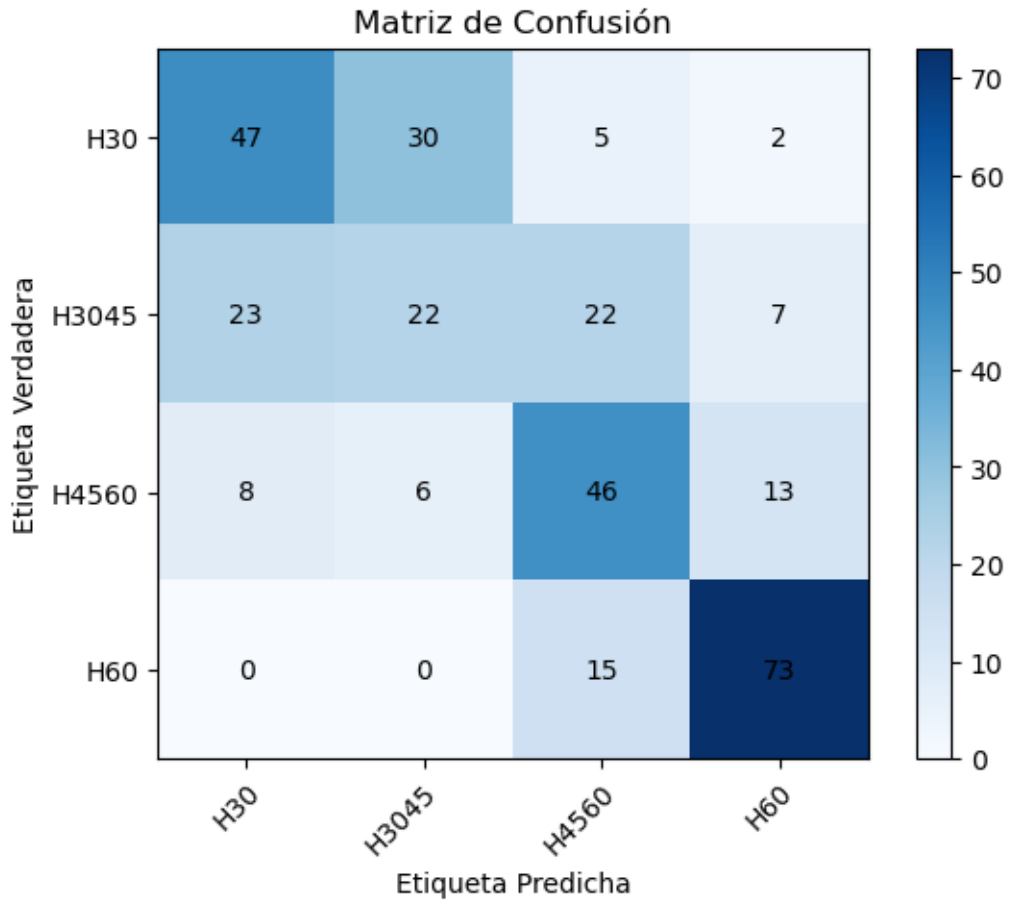
[33]:
```
"\nimport time\n\nt1=time.time()\nelbow=[]\nn_range=[]\not=[]\nfor o in
range(30):\n    resp=mod_knn(entrenamiento80,prueba20,o+1)\n
elbow.append(accuracy_score(prueba20.iloc[:,-1],resp))\n    ot.append(o+1)\nt2=t
ime.time()\n\nplt.plot(ot,elbow)\n\n\n#plt.plot(elbow)\n#plt.scatter(6,elbow[6],
color='red')\n#plt.title('Método del codo ')\n#plt.xlabel('n
```

```
Vecinos')\n#plt.ylabel('Distancia')\n"
```

[34]:
```
pred=mod_knn(entrenamiento80,prueba20,7)
```

```
8
```

[35]:
```
#"""
pred.astype(int)
matriz_confusion=confusion_matrix(prueba20.iloc[:,-1],pred)
precision=precision_score(prueba20.iloc[:,-1],pred,average=None)
exactitud=accuracy_score(prueba20.iloc[:,-1],pred)
sensibilidad=recall_score(prueba20.iloc[:,-1],pred,average=None)
f1=f1_score(prueba20.iloc[:,-1],pred,average=None)

labels=['H30','H3045','H4560','H60']
fig, ax = plt.subplots()
im = ax.imshow(matriz_confusion, cmap='Blues')
cbar = ax.figure.colorbar(im, ax=ax)
ax.set_xticks(np.arange(len(labels)))
ax.set_yticks(np.arange(len(labels)))
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)

plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

# Iterar sobre los datos y agregar el valor en cada celda
for i in range(len(labels)):
    for j in range(len(labels)):
        text = ax.text(j, i, matriz_confusion[i, j], ha="center", va="center",␣
 ↪color="black")

# Configurar título y etiquetas de los ejes
ax.set_title("Matriz de Confusión")
ax.set_xlabel("Etiqueta Predicha")
ax.set_ylabel("Etiqueta Verdadera")

# Mostrar la figura
plt.show()
print('exactitud: ',exactitud)
print('Precisión ind: ',precision)
print('precisión total: ', np.mean(precision))
print('sensibilidad ind (recall): ',sensibilidad)
print('Sensibilidad total: ', np.mean(sensibilidad))
print('f1 score individual: ',f1)
print('f1 score total: ', np.mean(f1))
#"""
```




```
exactitud:  0.5893416927899686
Precisión ind:  [0.6025641  0.37931034 0.52272727 0.76842105]
precisión total:  0.568255693187635
sensibilidad ind (recall):  [0.55952381 0.2972973  0.63013699 0.82954545]
Sensibilidad total:  0.5791258869169829
f1 score individual:  [0.58024691 0.33333333 0.57142857 0.79781421]
f1 score total:  0.5707057564981062
```

[36]:
```
entrenamiento80.tail()
```

[36]:

	Cement	Water	Coarse Aggregate	Blast Furnace Slag	\
1569	0.874165	0.430926	0.806825	0.200503	
1405	0.998387	0.320837	0.737090	0.000206	
231	0.255023	0.478435	0.769767	0.272955	
1582	0.604761	0.149332	0.281088	0.155437	
37	0.526256	0.848243	0.380814	0.396494	

	Superplasticizer	Strength
1569	0.081508	3

```
1405        0.078658        3
231         0.214286        2
1582        0.529827        3
37          0.000000        1
```

[37]:
```
#pred.value_counts()
```

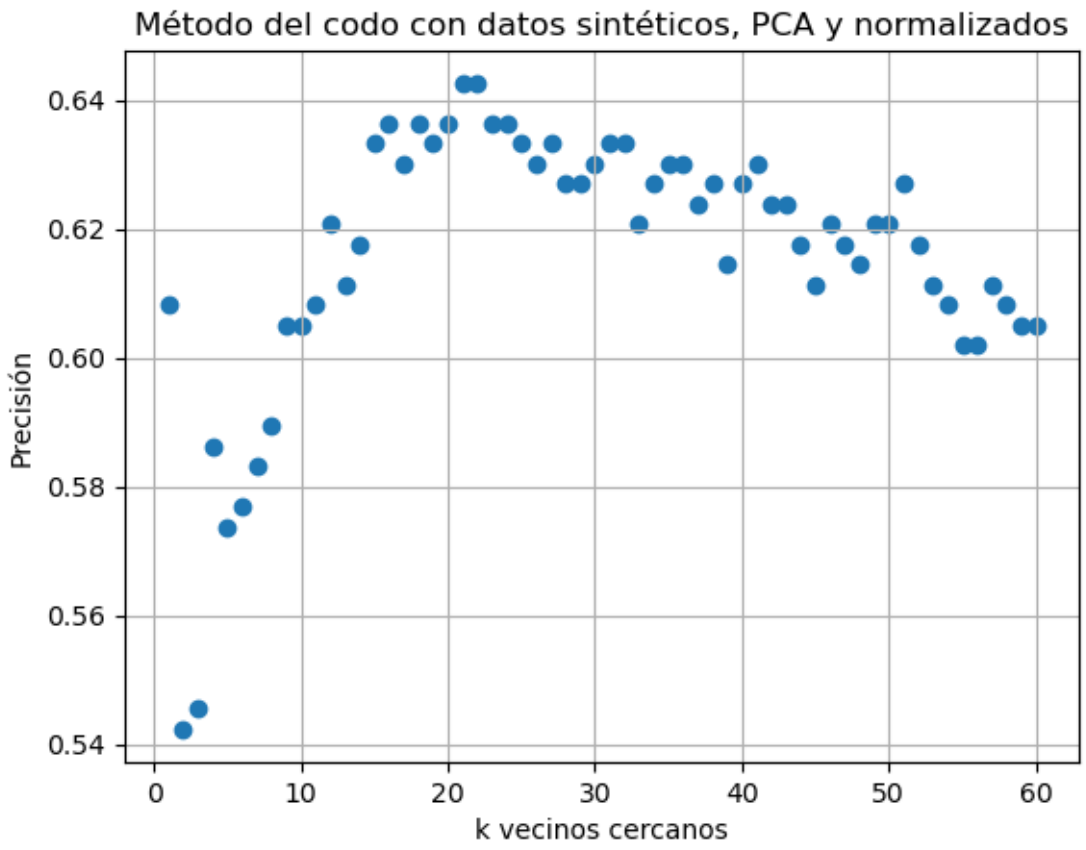
[38]:
```
# K-folds
k_folds=7
bins=np.array_split(datan_pca,k_folds)
#print(bins[0].shape)
```

[39]:
```
########################################## K-folds␣
 ↪#################################
#"""
res=[]
pres=[]
sens=[]
exac=[]
f1sc=[]

for f in range(k_folds):
    excluded_group=bins[f]
    tr_group=[grupo for j, grupo in enumerate(bins) if j!=f]
    concatenated_data=pd.concat(tr_group,axis=0)
    res.append(mod_knn(concatenated_data,excluded_group,6))
    pres.append(precision_score(excluded_group.iloc[:,-1],res[-1],average=None))
    exac.append(accuracy_score(excluded_group.iloc[:,-1],res[-1]))
    sens.append(recall_score(excluded_group.iloc[:
 ↪,-1],res[-1],average=None,zero_division=1))
    f1sc.append(f1_score(excluded_group.iloc[:
 ↪,-1],res[-1],average=None,zero_division=1))

print('exactitud en cada prueba: ',exac)
print('precisión promedio en cada clase en todas los kfolds: ', np.
 ↪mean(pres,axis=0))
print('Precisión total: ',np.mean(pres))
print('sensibilidad (recall) promedio en cada clase en todos los kfold: ',np.
 ↪mean(sens,axis=0))
print('Sensibilidad total: ', np.mean(sensibilidad))
print('f1 score promedio en cada clase de kfolds: ',np.mean(f1,axis=0))
print('f1 score total: ', np.mean(f1))
#"""
```

```
21
54
56
```

```
71
39
26
43
exactitud en cada prueba:  [0.5131578947368421, 0.40350877192982454,
0.4780701754385965, 0.6475770925110133, 0.6211453744493393, 0.6079295154185022,
0.5991189427312775]
precisión promedio en cada clase en todas los kfolds:  [0.43598696 0.38249706
0.47310148 0.5214824 ]
Precisión total:  0.4532669742574633
sensibilidad (recall) promedio en cada clase en todos los kfold:  [0.71546319
0.51972696 0.63777969 0.8191382 ]
Sensibilidad total:  0.5791258869169829
f1 score promedio en cada clase de kfolds:  0.5707057564981062
f1 score total:  0.5707057564981062
```

[40]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
score_acc=[]
kk=[]
#knn_sk=KNeighborsClassifier(n_neighbors=11)
#res=knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
#print(knn_sk.predict(prueba20.iloc[:,:-1]))
#"""
for k in range(60):
    knn_sk=KNeighborsClassifier(n_neighbors=k+1)
    knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
    score_acc.append(knn_sk.score(prueba20.iloc[:,:-1],prueba20.iloc[:,-1]))
    kk.append(k+1)
#"""
plt.scatter(kk,score_acc)
plt.xlabel('k vecinos cercanos')
plt.ylabel('Precisión')
plt.title('Método del codo con datos sintéticos, PCA y normalizados')
plt.grid()
```



[41]:
```
for ll in range(17,23):
    knn_sk=KNeighborsClassifier(n_neighbors=ll)
    knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
    kn=knn_sk.predict(prueba20.iloc[:,:-1])
    metric=classification_report(prueba20.iloc[:,-1],kn)
    print('prueba con k=',ll)
    print(metric)
```

```
prueba con k= 17
              precision    recall  f1-score   support

           0       0.62      0.67      0.64        84
           1       0.51      0.31      0.39        74
           2       0.56      0.58      0.57        73
           3       0.73      0.91      0.81        88

    accuracy                           0.63       319
   macro avg       0.61      0.62      0.60       319
weighted avg       0.61      0.63      0.61       319
```

```
prueba con k= 18
              precision    recall  f1-score   support

           0       0.61      0.69      0.65        84
           1       0.51      0.27      0.35        74
           2       0.57      0.59      0.58        73
           3       0.75      0.93      0.83        88

    accuracy                           0.64       319
   macro avg       0.61      0.62      0.60       319
weighted avg       0.62      0.64      0.61       319

prueba con k= 19
              precision    recall  f1-score   support

           0       0.62      0.69      0.66        84
           1       0.49      0.28      0.36        74
           2       0.57      0.58      0.57        73
           3       0.74      0.92      0.82        88

    accuracy                           0.63       319
   macro avg       0.61      0.62      0.60       319
weighted avg       0.61      0.63      0.61       319

prueba con k= 20
              precision    recall  f1-score   support

           0       0.62      0.70      0.66        84
           1       0.50      0.28      0.36        74
           2       0.58      0.58      0.58        73
           3       0.74      0.92      0.82        88

    accuracy                           0.64       319
   macro avg       0.61      0.62      0.60       319
weighted avg       0.62      0.64      0.62       319

prueba con k= 21
              precision    recall  f1-score   support

           0       0.63      0.71      0.67        84
           1       0.53      0.28      0.37        74
           2       0.57      0.59      0.58        73
           3       0.74      0.92      0.82        88

    accuracy                           0.64       319
   macro avg       0.62      0.63      0.61       319
weighted avg       0.62      0.64      0.62       319
```

```
prueba con k= 22
              precision    recall  f1-score   support

           0       0.63      0.75      0.68        84
           1       0.53      0.24      0.33        74
           2       0.57      0.63      0.60        73
           3       0.75      0.89      0.81        88

    accuracy                           0.64       319
   macro avg       0.62      0.63      0.61       319
weighted avg       0.63      0.64      0.62       319
```

[42]:
```
## K-folds
neigh=[17,19,21]
folds=[3,5,7,9]

kfoldacc=np.zeros((len(folds),len(neigh)))
kfoldpres=np.zeros((len(folds),len(neigh)))
kfoldrecall=np.zeros((len(folds),len(neigh)))
kfoldf1=np.zeros((len(folds),len(neigh)))

kfa=[]
kfp=[]
kfr=[]
kff1=[]
#kfoldmet=np.zeros()

for aa in range(len(folds)):
    np.random.seed(2345)
    bins=np.array_split(datan_pca,folds[aa])
    for bb in range(len(neigh)):
        kfoldacc=np.zeros((folds[aa],1))
        kfoldpres=np.zeros((folds[aa],1))
        kfoldrecall=np.zeros((folds[aa],1))
        kfoldf1=np.zeros((folds[aa],1))
        for f in range(folds[aa]):
            excluded_group=bins[f]
            #print('ex',excluded_group.shape)
            tr_group=[grupo for j, grupo in enumerate(bins) if j!=f]
            concatenated_data=pd.concat(tr_group,axis=0)
            #print('conc',concatenated_data.shape)
            knn_sk=KNeighborsClassifier(n_neighbors=neigh[bb])
            knn_sk.fit(concatenated_data.iloc[:,:-1],concatenated_data.iloc[:
 ↪,-1])
            kn=knn_sk.predict(excluded_group.iloc[:,:-1])
```

```
            metric=classification_report(excluded_group.iloc[:
↪,-1],kn,output_dict=True,zero_division=1)
            #print('metrica',metric)
            kfoldacc[f,]=metric['accuracy']
            kfoldpres[f,]=metric['weighted avg']['precision']
            kfoldrecall[f,]=metric['weighted avg']['recall']
            kfoldf1[f,]=metric['weighted avg']['f1-score']
        #print(f)
        kfa.append(np.mean(kfoldacc))
        kfp.append(np.mean(kfoldpres))
        kfr.append(np.mean(kfoldrecall))
        kff1.append(np.mean(kfoldf1))
        #kfoldmet[]

#knn_sk=KNeighborsClassifier(n_neighbors=17)
#knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
#kn=knn_sk.predict(prueba20.iloc[:,:-1])
#metric=classification_report(prueba20.iloc[:,-1],kn,output_dict=True)
#metric['weighted avg']#['precision']
```

[43]:
```
kff1
```

[43]:
```
[0.40416710372679177,
 0.4074588729178048,
 0.3960401109981903,
 0.4600006742847783,
 0.45776936952580866,
 0.4624234250779655,
 0.5585201404168109,
 0.5505953365152834,
 0.5621453257795391,
 0.5579528321369582,
 0.539150333415906,
 0.5417196706585305]
```

[44]:
```
n_bins=3
bins=np.array_split(datan_pca,n_bins)
for f in range(n_bins):
    excluded_group=bins[f]
    print('ex',excluded_group.shape)
    tr_group=[grupo for j, grupo in enumerate(bins) if j!=f]
    concatenated_data=pd.concat(tr_group,axis=0)
    print('conc',concatenated_data.shape)
    knn_sk=KNeighborsClassifier(n_neighbors=17)
    knn_sk.fit(concatenated_data.iloc[:,:-1],concatenated_data.iloc[:,-1])
    kn=knn_sk.predict(excluded_group.iloc[:,:-1])
```

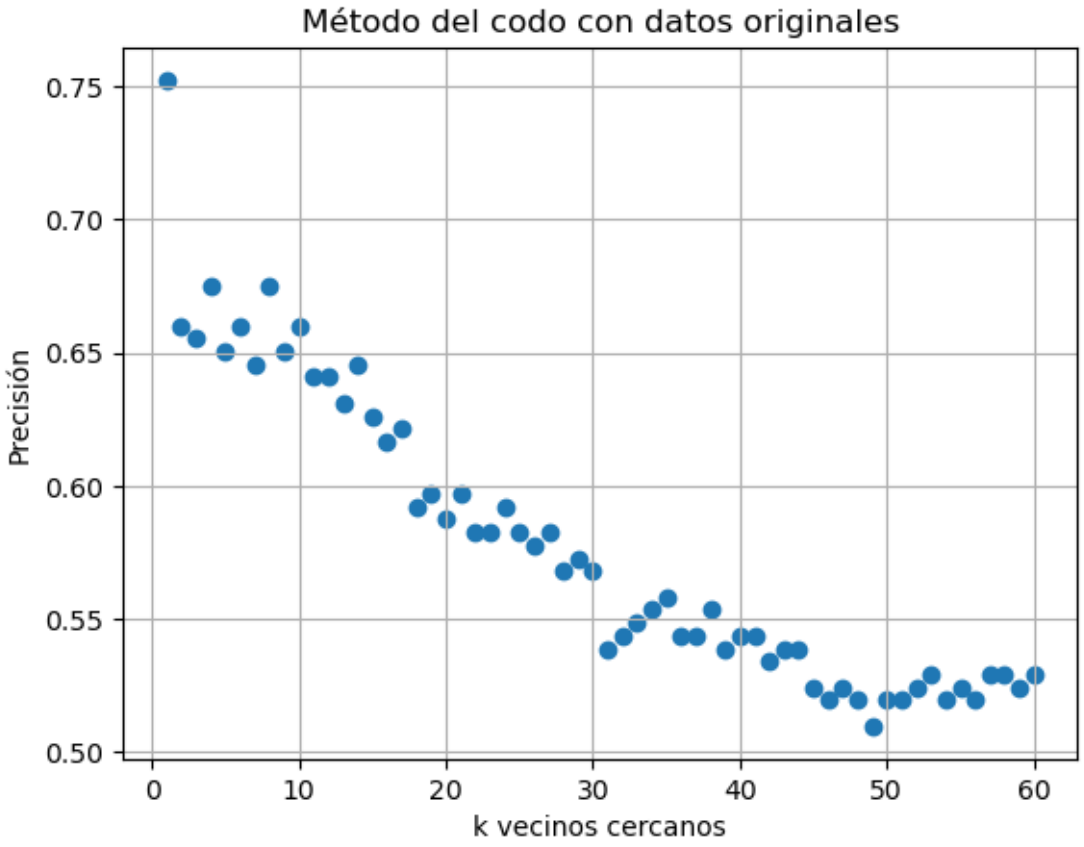
```
    metric=classification_report(excluded_group.iloc[:
↪,-1],kn,output_dict=True,zero_division=1)
    print(metric)
```

```
ex (531, 6)
conc (1061, 6)
{'0': {'precision': 0.4625, 'recall': 0.2740740740740741, 'f1-score':
0.34418604651162793, 'support': 135}, '1': {'precision': 0.5, 'recall':
0.07386363636363637, 'f1-score': 0.12871287128712872, 'support': 176}, '2':
{'precision': 0.3208955223880597, 'recall': 0.6277372262773723, 'f1-score':
0.42469135802469127, 'support': 137}, '3': {'precision': 0.47770700636942676,
'recall': 0.9036144578313253, 'f1-score': 0.625, 'support': 83}, 'accuracy':
0.3973634651600753, 'macro avg': {'precision': 0.4402756321893716, 'recall':
0.469822348636602, 'f1-score': 0.380647568955862, 'support': 531}, 'weighted
avg': {'precision': 0.44077187965315745, 'recall': 0.3973634651600753,
'f1-score': 0.33743182236344094, 'support': 531}}
ex (531, 6)
conc (1061, 6)
{'0': {'precision': 0.728, 'recall': 0.34600760456273766, 'f1-score':
0.46907216494845366, 'support': 263}, '1': {'precision': 0.27472527472527475,
'recall': 0.11467889908256881, 'f1-score': 0.16181229773462785, 'support': 218},
'2': {'precision': 0.08365019011406843, 'recall': 0.5641025641025641,
'f1-score': 0.14569536423841056, 'support': 39}, '3': {'precision':
0.21153846153846154, 'recall': 1.0, 'f1-score': 0.34920634920634924, 'support':
11}, 'accuracy': 0.2806026365348399, 'macro avg': {'precision':
0.3244784815944512, 'recall': 0.5061972669369676, 'f1-score':
0.28144654403196034, 'support': 531}, 'weighted avg': {'precision':
0.4838858575922442, 'recall': 0.2806026365348399, 'f1-score':
0.31669387821875716, 'support': 531}}
ex (530, 6)
conc (1062, 6)
{'0': {'precision': 0.0, 'recall': 1.0, 'f1-score': 0.0, 'support': 0}, '1':
{'precision': 0.013986013986013986, 'recall': 0.5, 'f1-score':
0.0272108843537415, 'support': 4}, '2': {'precision': 0.49732620320855614,
'recall': 0.4189189189189189, 'f1-score': 0.4547677261613692, 'support': 222},
'3': {'precision': 0.9146341463414634, 'recall': 0.4934210526315789, 'f1-score':
0.6410256410256411, 'support': 304}, 'accuracy': 0.46226415094339623, 'macro
avg': {'precision': 0.35648659088400836, 'recall': 0.6030849928876244,
'f1-score': 0.2807510628851879, 'support': 530}, 'weighted avg': {'precision':
0.7330398899170724, 'recall': 0.46226415094339623, 'f1-score':
0.5583756105981771, 'support': 530}}
```

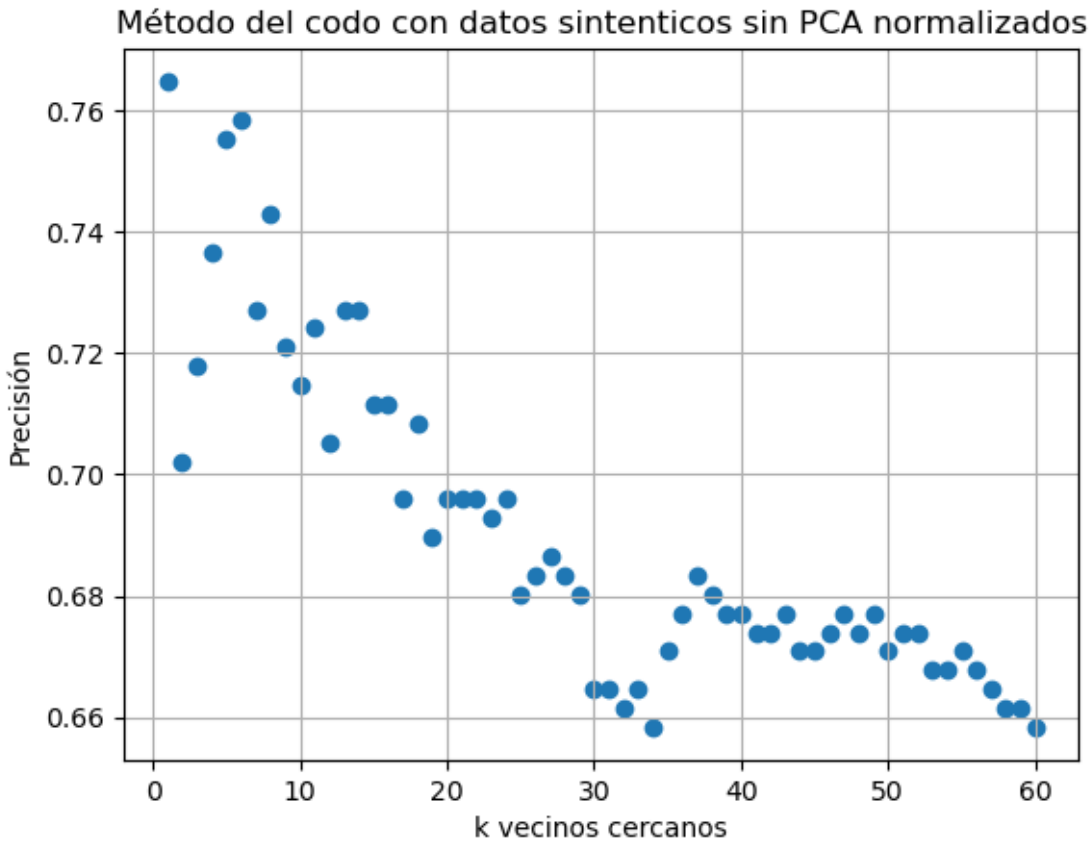
[ ]:

[45]:
```
# Datos originales
np.random.seed(4567)
entrenamiento80=data3.sample(n=int(data3.shape[0]*0.8))
```

```
prueba20=data3[~data3.index.isin(entrenamiento80.index)]
#data3.shape
score_acc=[]
kk=[]
#knn_sk=KNeighborsClassifier(n_neighbors=11)
#res=knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
#print(knn_sk.predict(prueba20.iloc[:,:-1]))
#"""
for k in range(60):
    knn_sk=KNeighborsClassifier(n_neighbors=k+1)
    knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
    score_acc.append(knn_sk.score(prueba20.iloc[:,:-1],prueba20.iloc[:,-1]))
    kk.append(k+1)
#"""
plt.scatter(kk,score_acc)
plt.xlabel('k vecinos cercanos')
plt.ylabel('Precisión')
plt.title('Método del codo con datos originales')
plt.grid()
```



[46]:
```
#data_eq.shape
np.random.seed(4567)
data_eq=pd.concat([data_eq,data2['Strength']],axis=1)
entrenamiento80=data_eq.sample(n=int(data_eq.shape[0]*0.8))
prueba20=data_eq[~data_eq.index.isin(entrenamiento80.index)]
#data3.shape
score_acc=[]
kk=[]
#knn_sk=KNeighborsClassifier(n_neighbors=11)
#res=knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
#print(knn_sk.predict(prueba20.iloc[:,:-1]))
#"""
for k in range(60):
    knn_sk=KNeighborsClassifier(n_neighbors=k+1)
    knn_sk.fit(entrenamiento80.iloc[:,:-1],entrenamiento80.iloc[:,-1])
    score_acc.append(knn_sk.score(prueba20.iloc[:,:-1],prueba20.iloc[:,-1]))
    kk.append(k+1)
#"""
plt.scatter(kk,score_acc)
plt.xlabel('k vecinos cercanos')
plt.ylabel('Precisión')
plt.title('Método del codo con datos sintenticos sin PCA normalizados')
plt.grid()
```



[ ]:

[47]:
```
data_eq.head()
```

[47]:

	Cement	Water	Coarse Aggregate	Age	Fine Aggregate	Fly Ash	\
0	1.000000	0.321086	0.694767	0.074176	0.205720	0.0	
1	1.000000	0.321086	0.738372	0.074176	0.205720	0.0	
2	0.526256	0.848243	0.380814	0.739011	0.000000	0.0	
3	0.526256	0.848243	0.380814	1.000000	0.000000	0.0	
4	0.220548	0.560703	0.515698	0.986264	0.580783	0.0	

	Blast Furnace Slag	Superplasticizer	Strength
0	0.000000	0.07764	3
1	0.000000	0.07764	3
2	0.396494	0.00000	1
3	0.396494	0.00000	1
4	0.368392	0.00000	1

[ ]:

[ ]: