INFORME PARA LA TAREA N°2

Nombre: Juan Torres **Cédula**: E-8-160393 **Grupo**: 1AN213

Materia: Introducción a la Ciencia de Datos **Profesor**: Juan Montenegro

1. Regresión Lineal

La **regresión lineal** es un modelo lineal utilizado en el análisis de datos para realizar predicciones en base a valores conocidos (características) que guardan cierta relación con nuestro objetivo. La ecuación de un modelo lineal, para un número n de características es la siguiente:

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[n] \times x[n] + b$$

En este modelo, los coeficientes (w) nos indican el peso asignado a cada característica (x). Son indicadores de su importancia para el resultado (y).

El objetivo de un modelo lineal es optimizar el peso (b), mediante la siguiente ecuación de coste:

$$\sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{p} (w_j \times x_{ij}) \right)^2$$

En esta ecuación, $\mathbf{y_i}$ es el **valor real**, $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ es el **valor predicho** a partir de nuestra ecuación lineal, **m** es el **número de filas** y **p** es el **número de características**.

2. Regularización

Al momento de entrenar y probar los modelos de regresión, pueden surgir dos problemas: **sobreajuste** y **subajuste**.

El **sobreajuste** ocurre cuando el modelo brinda predicciones precisas para los datos de entrenamiento, pero no para los datos de prueba o los datos nuevos.

Por otra parte, el **subajuste** se presenta cuando el modelo no puede capturar la relación entre las variables de entrada y salida con precisión, lo que genera una alta tasa de error tanto en el conjunto del entrenamiento como en los datos de prueba.

El sobreajuste tiende a darse cuando el modelo es muy complejo, y el subajuste se da cuando es muy simple.

Por medio de la **regularización** se busca evitar el sobreajuste de los datos, especialmente cuando existe una gran variabilidad entre el rendimiento de los datos de entrenamiento y los de prueba.

Hay varios métodos para reducir la complejidad de los modelos y así evitar el sobreajuste en los modelos lineales, por ejemplo: **los modelos de regresión de Lasso y Ridge**.

3. Regresión de Lasso

La Regresión de Lasso es un tipo de regresión lineal que utiliza la regularización L1, la cual se define como la suma de los valores absolutos de los coeficientes del modelo.

La principal característica de la norma L1 es su capacidad para reducir algunos coeficientes a cero. La regresión Lasso logra esto al introducir un **factor de penalización** llamado **alpha (α)**, lo que permite disminuir la complejidad de un modelo de machine learning.

Esta penalización añadida, se basa en la siguiente **función de costo**:

$$\text{Costo}_{\text{Lasso}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} {\beta_j}^2$$

Donde:

- y_i son los **valores reales** (**n** es el **número de observaciones**)
- $\hat{y}_i$ son los **valores predichos**
- β_j son los **coeficientes del modelo** (**p** es el **número de características**)
- α es el **factor de penalización**

Este enfoque ayuda a eliminar las características no relevantes, enfocándose en las más importantes y, de esta manera, evita el sobreajuste a los datos de entrenamiento.

4. Regresión de Ridge

Por otra parte, la **Regresión de Ridge**, también conocida como **regularización L2**, es otra técnica de regularización que penaliza la magnitud de los coeficientes del modelo. Pero, a diferencia de Lasso, Ridge no reduce los coeficientes a cero, sino que **los hace más pequeños**. Esto es muy útil para lidiar con la multicolinealidad, es decir, cuando los predictores están altamente correlacionados, y también ayuda a evitar el sobreajuste.

La regresión Ridge agrega una penalización basada en la suma de cuadrados de los coeficientes de la **función de costo**.

$$\text{Costo}_{\text{Ridge}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} {\beta_j}^2$$

Donde:

- y_i son los **valores reales** (**n** es el **número de observaciones**)
- $\hat{y}_i$ son los **valores predichos**
- β_j son los **coeficientes del modelo** (**p** es el **número de características**)
- α es un parámetro que **controla el grado de penalización**. Entre mayor sea, los coeficientes serán menores.

DESARROLLO

(a) Graficar los datos que descargaste como un gráfico de dispersión 3D. ¿Parece que los datos de entrenamiento se encuentran en un plano o en una curva?

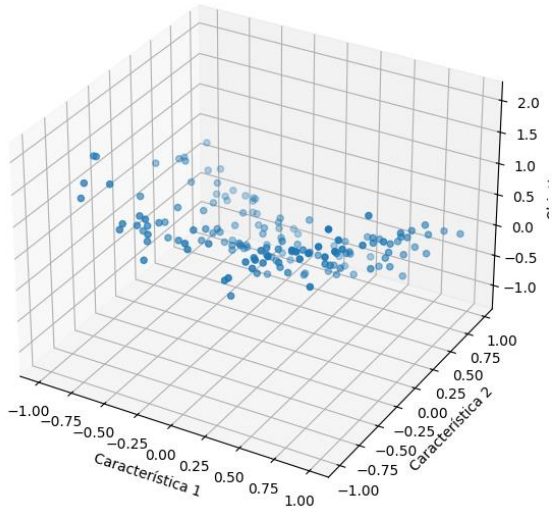


*Figura 1: Gráfico de Dispersión 3D para los **datos de entrenamiento.***

- ***Eje X**: **Característica 1***
- ***Eje y: Característica 2,***
- ***Eje Z: Variable Objetivo***

En este gráfico, los datos están representados por un conjunto de puntos que parecen seguir una tendencia lineal. A simple vista, pareciese ser que los datos de entrenamiento se encuentran sobre un plano, en dos dimensiones, particularmente, el plano XY.

(b) Agregar características polinómicas adicionales

Utilizando la función PolynomialFeatures de sklearn, se agregaron en total **21 nuevas características polinómicas** equivalentes a las combinaciones de las potencias de las dos características iniciales (X1 y X2)

```
Características polinómicas:
['1' 'X1' 'X2' 'X1^2' 'X1 X2' 'X2^2' 'X1^3' 'X1^2 X2' 'X1 X2^2' 'X2^3'
 'X1^4' 'X1^3 X2' 'X1^2 X2^2' 'X1 X2^3' 'X2^4' 'X1^5' 'X1^4 X2'
 'X1^3 X2^2' 'X1^2 X2^3' 'X1 X2^4' 'X2^5']
```

```
[6]: print(f"Original columns: {X.shape[1]}, New columns: {X_poly.shape[1]}")
     Original columns: 2, New columns: 21
```

Ahora entrena modelos de regresión Lasso con estas características polinómicas para un amplio rango de valores de C, por ejemplo, 1, 10, 1000

Para los modelos de regresión de Lasso, el valor C y el valor de *alpha (α)* son inversamente proporcionales (C=1/ *α*). Entonces, si C disminuye, α aumenta, y si C aumenta, α disminuye.

- **Para C=0.0001, 0.001, 0.01, 0.1 (α=10 000, 1000, 100, 10)**

Los resultados nos indicaron que para **valores altos y muy altos de α** (10 000, 1000, 100, 10), es decir, **valores muy bajos de C** (0.0001, 0.001, 0.01, 0.1), la mayoría de los coeficientes eran cero o muy cercanos a cero.

```
Resultados para C = 0.0001 (alpha = 10000.0):
1: 0.0
X1: 0.0
X2: -0.0
X1^2: 0.0
X1 X2: -0.0
X2^2: 0.0
X1^3: 0.0
X1^2 X2: -0.0
X1 X2^2: 0.0
X2^3: -0.0
X1^4: 0.0
X1^3 X2: -0.0
X1^2 X2^2: 0.0
X1 X2^3: -0.0
X2^4: 0.0
X1^5: 0.0
X1^4 X2: -0.0
X1^3 X2^2: 0.0
X1^2 X2^3: -0.0
X1 X2^4: 0.0
X2^5: -0.0
```

```
Resultados para C = 0.01 (alpha = 100.0):
1: 0.0
X1: 0.0
X2: -0.0
X1^2: 0.0
X1 X2: -0.0
X2^2: 0.0
X1^3: 0.0
X1^2 X2: -0.0
X1 X2^2: 0.0
X2^3: -0.0
X1^4: 0.0
X1^3 X2: -0.0
X1^2 X2^2: 0.0
X1 X2^3: -0.0
X2^4: 0.0
X1^5: 0.0
X1^4 X2: -0.0
X1^3 X2^2: 0.0
X1^2 X2^3: -0.0
X1 X2^4: 0.0
X2^5: -0.0
```

Cuando trabajamos con **valores altos de α,** la penalización sobre los coeficientes es **considerable**.

Como podemos observar en estos resultados, las restricciones impuestas son tan fuertes que todos los coeficientes se reducen a cero o a valores muy cercanos a cero.

Este es el comportamiento esperado al trabajar con la **regresión de Lasso**, sobre todo en situaciones donde se trabaja con muchas características, como en este caso, que trabajamos con las 21 variables polinómicas generadas.

Sin embargo, este resultado no nos conviene, ya que nuestro objetivo es poder identificar y descartar aquellas características que no sean significativas para el estudio, no eliminarlas todas.

Por lo tanto, vamos a ir **disminuyendo gradualmente el valor de α**, para observar el comportamiento de los coeficientes, y así mantener solo las variables más relevantes.

- **Para C=1 (α=1)**

Este comportamiento se repitió en cada ocasión, hasta que llegamos a **C=1**, **α=1**. Así que se tomó la decisión de usar estos valores como punto de partida para el análisis.

```
Resultados para C = 1 (alpha = 1.0):
1: 0.0
X1: 0.0
X2: -0.0
X1^2: 0.0
X1 X2: -0.0
X2^2: 0.0
X1^3: 0.0
X1^2 X2: -0.0
X1 X2^2: 0.0
X2^3: -0.0
X1^4: 0.0
X1^3 X2: -0.0
X1^2 X2^2: 0.0
X1 X2^3: -0.0
X2^4: 0.0
X1^5: 0.0
X1^4 X2: -0.0
X1^3 X2^2: 0.0
X1^2 X2^3: -0.0
X1 X2^4: 0.0
X2^5: -0.0
```

- **Para C=10 (α = 0.1)**

Un valor de C=10 nos da un valor más bajo para α, lo que significa que la penalización disminuye, y, por lo tanto, los coeficientes tendrán menor restricción.

Por eso vemos en nuestros resultados que, a pesar de que la mayoría de los coeficientes siguen siendo cero, para el caso de X2 ya obtuvimos un valor diferente. Esto sugiere que X2 podría ser una característica relevante para el modelo.

Además, demuestra que, al ir aumentando el valor de C, disminuyendo el valor de α, podremos identificar más fácilmente las características más significativas en nuestro estudio, y descartar aquellas que no lo son.

```
Resultados para C = 10 (alpha = 0.1):
1: 0.0
X1: 0.0
X2: -0.7592007341941692
X1^2: 0.0
X1 X2: -0.0
X2^2: 0.0
X1^3: 0.0
X1^2 X2: -0.0
X1 X2^2: 0.0
X2^3: -0.0
X1^4: 0.0
X1^3 X2: -0.0
X1^2 X2^2: 0.0
X1 X2^3: -0.0
X2^4: 0.0
X1^5: 0.0
X1^4 X2: -0.0
X1^3 X2^2: 0.0
X1^2 X2^3: -0.0
X1 X2^4: 0.0
X2^5: -0.0
```

- **Para C= 100 (α=0.01)**

Al disminuir la restricción sobre los coeficientes, vemos que ahora nos han presentado una nueva posible variable relevante para el modelo: X_1^2, manteniendo todavía a X_2.

```
Resultados para C = 100 (alpha = 0.01):
1: 0.0
X1: -0.0
X2: -0.9973566952239966
X1^2: 0.8985744508163679
X1 X2: 0.0
X2^2: 0.0
X1^3: -0.0
X1^2 X2: 0.0
X1 X2^2: -0.0
X2^3: -0.0
X1^4: 0.0
X1^3 X2: 0.0
X1^2 X2^2: 0.0
X1 X2^3: -0.0
X2^4: 0.0
X1^5: -0.0
X1^4 X2: 0.0
X1^3 X2^2: -0.0
X1^2 X2^3: 0.0
X1 X2^4: -0.0
X2^5: -0.0
```

El modelo Lasso, para C=10, nos arrojó solo un coeficiente, para X_2, lo cual indicaba un comportamiento lineal, pero para C=100, nos dio también un segundo valor para el coeficiente de una de las demuestra que a medida que aumentamos los valores de C, y que el modelo se vuelve más complejo, este se va transformando de un modelo lineal a uno cuadrático, es decir que representa una curva.

- **Para C=1000 (α = 0.001)**

Al aumentar el valor de C a 1000, el valor de α disminuyó a 0.001. Esto redujo las restricciones sobre los coeficientes casi al mínimo, lo cual podemos ver reflejado en los resultados.

Pues, ahora obtuvimos valores diferentes a cero para varios de nuestros coeficientes, particularmente de las características: X_1^2, $X_2 X_1^2$, $X_1 X_2^2$, X_1^4, $X_1^3 X_2$ y X_1^5. Lo que indicaría que el modelo las considera importantes en la predicción de la variable objetivo.

Sin embargo, debemos tener cuidado, pues al trabajar con tantas variables, podríamos caer en el **sobreajuste**.

```
Resultados para C = 1000 (alpha = 0.001):
1: 0.0
X1: 0.0
X2: -1.0649190700606501
X1^2: 0.9162330780025673
X1 X2: 0.0
X2^2: 0.0
X1^3: -0.0
X1^2 X2: 0.10758571861353783
X1 X2^2: -0.04586561716754906
X2^3: -0.0
X1^4: 0.09869387660350638
X1^3 X2: 0.016743699650863086
X1^2 X2^2: 0.0
X1 X2^3: -0.0
X2^4: 0.0
X1^5: -0.01464815019561762
X1^4 X2: 0.0
X1^3 X2^2: -0.0
X1^2 X2^3: -0.0
X1 X2^4: -0.0
X2^5: -0.0
```

Al estar trabajando con tantas variables, corremos el riesgo de que nuestro modelo caiga en el **sobreajuste**, es por eso por lo que al aplicar la regresión Lasso, probando con diferentes valores para el factor de penalización (α), buscamos identificar cuáles son las variables que en verdad son esenciales para nuestro modelo. Y así, trabajar con un modelo más simple.

Como vimos en los resultados, al iniciar con valores tan altos para α, todos los coeficientes se hacían cero, pero a medida que íbamos disminuyendo los valores, obteníamos nuevos coeficientes para algunas de las variables.

Entre más disminuíamos α, más coeficientes obteníamos, lo cual nos ayuda a tener una idea de cuáles son las variables más relevantes para nuestro modelo. Eso sí, hay que tener cuidado de no caer en el sobreajuste.

Esto se debe a que cuando α es muy alto, la penalización sobre los coeficientes es muy alta y por eso se hacen cero, pero al disminuir el valor de α, la penalización también disminuye y le da mayor libertad al modelo para ajustarse a los datos.

(c) Para cada uno de los modelos del apartado (b), genera predicciones para la variable objetivo. Genera estas predicciones en una cuadrícula de valores de características.

En el apartado b, generé varios modelos probando diferentes valores de C. Pero, decidí generar las predicciones para los modelos donde: C=1, C=10, C=100 y C=1000. Los modelos con valores bajos de C, no los tomaré en cuenta ya que me arrojaron los mismos resultados que el modelo donde C=1.

1) **Creación de la Cuadrícula**

Lo primero que hicimos fue crear un conjunto de 50 valores, entre -5 y 5 para cubrir los posibles valores de las características.

Luego, creamos una lista vacía (X_test) donde vamos a almacenar todas las combinaciones de los valores de las características.

Ahora, generamos dos bucles: El primero toma un valor i del conjunto de datos, y el segundo, toma un valor j.

Cada par de valores (i, j), los colocamos juntos en una lista [i, j] y los añadimos a X_test usando append().

Convertimos la lista X_test en un arreglo de numpy usando np.array(), para poder hacer predicciones usando los valores de la lista.

Las primeras cinco filas de la lista serían:

```
[[-5.         -5.        ]
 [-5.         -4.79591837]
 [-5.         -4.59183673]
 [-5.         -4.3877551 ]
 [-5.         -4.18367347]]
```

2) Características Polinómicas de la Cuadrícula

Lo siguiente fue generar los datos polinómicos para la cuadrícula, igual que como lo hicimos para las características X1 y X2. Aquí lo que se hizo fue usar la función PolynomialFeatures para crear características polinómicas a partir de combinaciones de X1 y X2 con potencias hasta de grado 5, y nos aseguramos de que las combinaciones fueran las mismas que generamos anteriormente.

```
     1   X1        X2  X1^2      X1 X2       X2^2   X1^3     X1^2 X2  \
0  1.0 -5.0 -5.000000  25.0  25.000000  25.000000 -125.0 -125.000000
1  1.0 -5.0 -4.795918  25.0  23.979592  23.000833 -125.0 -119.897959
2  1.0 -5.0 -4.591837  25.0  22.959184  21.084965 -125.0 -114.795918
3  1.0 -5.0 -4.387755  25.0  21.938776  19.252395 -125.0 -109.693878
4  1.0 -5.0 -4.183673  25.0  20.918367  17.503124 -125.0 -104.591837

      X1 X2^2        X2^3  ...     X1^3 X2   X1^2 X2^2     X1 X2^3  \
0 -125.000000 -125.000000  ...  625.000000  625.000000  625.000000
1 -115.004165 -110.310117  ...  599.489796  575.020825  551.550587
2 -105.424823  -96.818715  ...  573.979592  527.124115  484.093575
3  -96.261974  -84.474794  ...  548.469388  481.309871  422.373968
4  -87.515618  -73.227354  ...  522.959184  437.578092  366.136771

         X2^4    X1^5      X1^4 X2    X1^3 X2^2    X1^2 X2^3      X1 X2^4  \
0  625.000000 -3125.0 -3125.000000 -3125.000000 -3125.000000 -3125.000000
1  529.038318 -3125.0 -2997.448980 -2875.104123 -2757.752935 -2645.191590
2  444.575732 -3125.0 -2869.897959 -2635.620575 -2420.467875 -2222.878661
3  370.654707 -3125.0 -2742.346939 -2406.549354 -2111.869842 -1853.273535
4  306.359339 -3125.0 -2614.795918 -2187.890462 -1830.683856 -1531.796696

          X2^5
0 -3125.000000
1 -2537.224587
2 -2041.419178
3 -1626.342081
4 -1281.707440

[5 rows x 21 columns]
```

3) Predicciones

Cuando trabajamos el modelo Lasso, vimos que, para los valores bajos de C, los coeficientes eran cero. Y esto seguía ocurriendo para diferentes valores de C, hasta llegar a C=1, por lo que se decidió realizar las predicciones a partir de este valor. Es decir, se generaron predicciones para los modelos donde C=1, C=10, C=100 y C=1000.

Entre algunos de los resultados de las predicciones, tenemos los siguientes:

```
Predicciones para C = 1 (alpha = 1.0):
[0.27050008 0.27050008 0.27050008 0.27050008 0.27050008]

Predicciones para C = 10 (alpha = 0.1):
[4.11243731 3.95749838 3.80255946 3.64762053 3.49268161]

Predicciones para C = 100 (alpha = 0.01):
[27.49502112 27.29147894 27.08793675 26.88439457 26.68085239]

Predicciones para C = 1000 (alpha = 0.001):
[138.4618994  137.90787537 137.37295404 136.85713544 136.36041955]
```

4) Gráfica

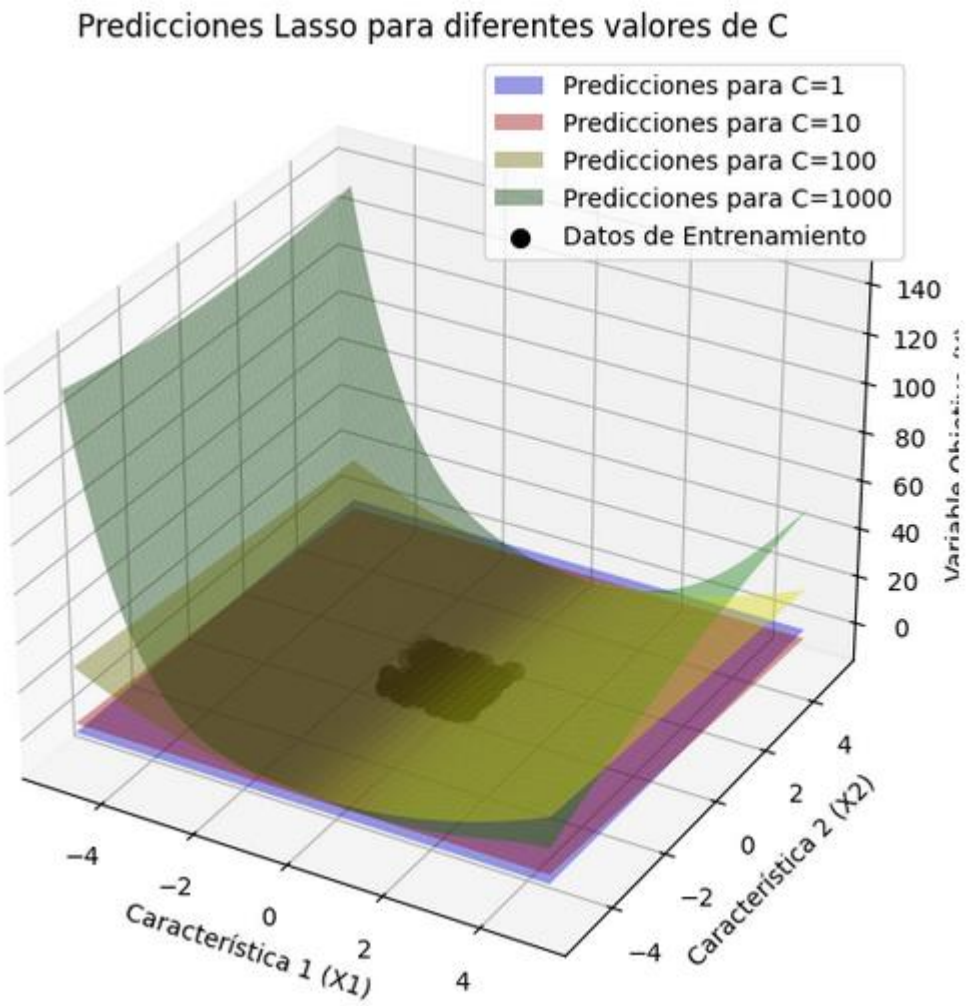


Figura 2**: Gráfico 3D con los **datos de entrenamiento** y las **predicciones** para los modelos de regresión Lasso donde **C=1**, **C=10**, **C=100** y **C=1000

Interpretación:

- **Predicciones**

A partir de las gráficas de las predicciones de los diferentes modelos, podemos ver el efecto que provoca el cambio de los valores de C.

Considero que lo que vemos en las gráficas complementa lo que observamos en el inciso (b). Cuando trabajamos con valores bajos de C, o valores altos de α, obtenemos un modelo más simple, debido a que Lasso penaliza fuertemente los coeficientes. Estos modelos tienen un comportamiento más plano y lineal.

Esto se ve reflejado en nuestra gráfica, pues para los valores más bajos de C, como C=1 y C=10, el gráfico de las predicciones toma la forma de un plano.

Y a medida que aumentamos el valor de C, o disminuimos α, entonces nuestro modelo se vuelve más complejo. Esto se observa en las gráficas, pues al representar las predicciones de los modelos donde C=100 y C=1000, estas van tomando una forma cada vez más curva, a medida que aumentamos los valores de C.

En resumen, podemos **observar gráficamente** cómo, al trabajar con Lasso, los valores más altos de C nos darán un modelo más complejo, y los valores más bajos, nos darán un modelo más simple y lineal.

- **Datos de Entrenamiento**

Cuando graficamos la primera vez los datos de entrenamiento en el inciso (a), vimos que estos estaban dispersos y parecían seguir una tendencia lineal. Ahora, cuando los graficamos junto a las predicciones de los modelos, se ven mucho más concentrados en el centro.

Esto puede ser debido a la diferencia de escalas, pues en la primera gráfica, nos enfocamos únicamente en los datos de entrenamiento, y su rango de valores. Pero, al graficarlos junto a las predicciones, las cuales tienen rangos más amplios y requieren de una mayor escala, puede parecer que los datos de entrenamiento estén "agrupados" o menos dispersos.

(d) ¿Qué es el sobreajuste y el subajuste? Usando los datos de parámetros del apartado (b) y la visualización del apartado (c), explica cómo se puede usar el parámetro de penalización C para gestionar el equilibrio entre el subajuste y el sobreajuste de los datos.

El **sobreajuste** es cuando un modelo de machine learning brinda predicciones muy precisas, prácticamente exactas, para los **datos de entrenamiento**, pero no para los **datos de prueba** o **datos nuevos**.

Por otro lado, el **subajuste** se da cuando un modelo de machine learning no puede capturar bien, o de forma precisa, la relación entre las variables de entrada y de salida, ni la variabilidad de los datos, dejando de lado importantes tendencias y patrones.

Además, presenta un alto índice de errores al trabajar con los datos de entrenamiento, los datos de prueba y los datos nuevos.

En el modelo de regresión Lasso, el **parámetro de penalización C**, por medio de su inverso **alpha (α)**, juega un papel crucial al momento de gestionar el equilibrio entre subajuste y sobreajuste de los datos.

Tanto en las gráficas de las predicciones del apartado (c), como en los coeficientes generados por la regresión Lasso para los diferentes valores de C en el apatado (b), podemos ver que la complejidad del modelo varía dependiendo de los cambios en el parámetro C.

Con un valor bajo de C, como en el caso de **C=1** o **C=10**, la regularización del modelo es alta (**α es alto**), y esto podemos verlo en la gran cantidad de coeficientes reducidos a cero, lo cual simplifica el modelo. Como resultado, también obtenemos una gráfica de predicción casi plana. Sin embargo, esta simplificación podría provocar un **subajuste**.

A medida que aumentamos el valor de C, por ejemplo, cuando **C=100** o **C=1000**, la penalización disminuye (**α es bajo**), permitiendo que los coeficientes tengan valores más grandes, lo cual nos da un modelo más complejo. En ambos casos, la gráfica de las predicciones toma una forma curva. Entre más alto el valor de C, más pronunciada es la curva. Sin embargo, un modelo muy complejo, conlleva un riesgo de **sobreajuste**.

Es posible lograr un equilibrio entre el subajuste y el sobreajuste de los datos al encontrar el valor adecuado para C. Esto se puede lograr por medio de la **validación cruzada**, donde probamos diferentes valores de C para así encontrar el indicado.

(e) Repite los apartados (b)-(c) para un modelo de regresión Ridge. Este usa una penalización L2 en lugar de una penalización L1 en la función de costo. Compara el impacto en los parámetros del modelo al cambiar C en la regresión Lasso y en la regresión Ridge.

1) Generación de Características Polinómicas

Primero, se generaron las mismas 21 características polinómicas con las que trabajamos el modelo de regresión de Lasso, usando Polynomial Features, para así poder apreciar mejor el impacto de cambiar al modelo de Ridge.

```
Características polinómicas:
['1' 'X1' 'X2' 'X1^2' 'X1 X2' 'X2^2' 'X1^3' 'X1^2 X2' 'X1 X2^2' 'X2^3'
 'X1^4' 'X1^3 X2' 'X1^2 X2^2' 'X1 X2^3' 'X2^4' 'X1^5' 'X1^4 X2'
 'X1^3 X2^2' 'X1^2 X2^3' 'X1 X2^4' 'X2^5']

Original columns: 2, New columns: 21
```

2) Entrenamiento de los Modelos

Al igual que el modelo de regresión de Lasso, el modelo de Ridge trabaja con un parámetro **α**, que **controla el grado de penalización**. Este parámetro es el inverso del valor C (C=1/ α). Por lo tanto, se espera que entre mayor sea α, o menor sea C, los coeficientes serán menores.

Resultados:

```
Resultados para C = 0.0001 (alpha = 10000.0):
1: 0.0
X1: 0.0012136440994787512
X2: -0.007261148544714049
X1^2: 0.001908607582519028
X1 X2: -0.00018033923798465082
X2^2: 0.0003604544234104459
X1^3: 0.0009512416785914265
X1^2 X2: -0.0023170600910598594
X1 X2^2: 0.0007793308304207714
X2^3: -0.004534495121057838
X1^4: 0.0016570631807680336
X1^3 X2: -0.0004579947900212403
X1^2 X2^2: 0.0009025232989467479
X1 X2^3: -0.00017429489047369245
X2^4: 0.0003745896750441139
X1^5: 0.0007930985884880564
X1^4 X2: -0.00139213921324121
X1^3 X2^2: 0.0005933330805546962
X1^2 X2^3: -0.0015176151415810866
X1 X2^4: 0.0006579702348537915
X2^5: -0.003326316906697218
```

```
Resultados para C = 0.001 (alpha = 1000.0):
1: 0.0
X1: 0.009611403424257663
X2: -0.06502516452594617
X1^2: 0.018258663112487177
X1 X2: -0.001651861655808666
X2^2: 0.0032706621406082183
X1^3: 0.00768285070148735
X1^2 X2: -0.02018983341895002
X1 X2^2: 0.006174668329971944
X2^3: -0.04021743021180574
X1^4: 0.015843466392949195
X1^3 X2: -0.004188528143864221
X1^2 X2^2: 0.008518259713878238
X1 X2^3: -0.001581457415212726
X2^4: 0.003374731561232028
X1^5: 0.006462530864941264
X1^4 X2: -0.011980191234859735
X1^3 X2^2: 0.004798880987697002
X1^2 X2^3: -0.013123747974155883
X1 X2^4: 0.005293549461050783
X2^5: -0.02934440455075745
```

```
Resultados para C = 0.01 (alpha = 100.0):
1: 0.0
X1: 0.020418568731012197
X2: -0.33099070696732924
X1^2: 0.13464164506402507
X1 X2: -0.006043862418147583
X2^2: 0.014892151299014098
X1^3: 0.019413130543675404
X1^2 X2: -0.08234860185934784
X1 X2^2: 0.008679044015055112
X2^3: -0.1886967438187266
X1^4: 0.11645773296251333
X1^3 X2: -0.021506275707590082
X1^2 X2^2: 0.057393142365896016
X1 X2^3: -0.006853801474338563
X2^4: 0.015228883366028807
X1^5: 0.017656754357910684
X1^4 X2: -0.043619941765878294
X1^3 X2^2: 0.0099492716385197
X1^2 X2^3: -0.049605596820939075
X1 X2^4: 0.009216301875803952
X2^5: -0.13126503514546461
```

```
Resultados para C = 0.1 (alpha = 10.0):
1: 0.0
X1: 0.012136717347756548
X2: -0.6958497641602677
X1^2: 0.4186787315902653
X1 X2: 0.01758425791349774
X2^2: -0.003241670681927677
X1^3: 0.001967147524414086
X1^2 X2: -0.059555366707802974
X1 X2^2: -0.04931293032521749
X2^3: -0.25497260016602363
X1^4: 0.3479460633242717
X1^3 X2: -0.009237041968940318
X1^2 X2^2: 0.14554694742918925
X1 X2^3: -0.005417335714864393
X2^4: 0.006561374468039278
X1^5: 0.006872882058316325
X1^4 X2: -0.003873805422217819
X1^3 X2^2: -0.018736845270238514
X1^2 X2^3: -0.008262236966240491
X1 X2^4: -0.03567981084885416
X2^5: -0.12147273978845637
```

```
Resultados para C = 1 (alpha = 1.0):
1: 0.0
X1: 0.04151925069416773
X2: -0.9766420105173507
X1^2: 0.6166962022569543
X1 X2: 0.006625969366431846
X2^2: -0.025538333290221198
X1^3: -0.045573305040478025
X1^2 X2: 0.07013819621793448
X1 X2^2: -0.11113564091639043
X2^3: -0.13220033063764747
X1^4: 0.36656444947613104
X1^3 X2: 0.09747557773706254
X1^2 X2^2: 0.13454816816707646
X1 X2^3: -0.06211861233917442
X2^4: -0.0008502974354951371
X1^5: -0.00030064324915689056
X1^4 X2: 0.03897675109765461
X1^3 X2^2: 0.02189622975644265
X1^2 X2^3: -0.0494930293313087
X1 X2^4: -0.02448379947860631
X2^5: 0.0305489123247204
```

```
Resultados para C = 10 (alpha = 0.1):
1: 0.0
X1: 0.09776615645224267
X2: -1.12816079356421
X1^2: 0.7289354448912896
X1 X2: -0.019990024514404296
X2^2: -0.02494827905057212
X1^3: -0.17400654515338881
X1^2 X2: 0.35997518105044973
X1 X2^2: -0.2663398901893836
X2^3: 0.04962227542242888
X1^4: 0.2830606719481492
X1^3 X2: 0.1639016901151386
X1^2 X2^2: 0.11205982665880818
X1 X2^3: -0.08742043885231401
X2^4: -0.007578433298005727
X1^5: 0.07061843093355336
X1^4 X2: -0.024528552350262845
X1^3 X2^2: 0.12335127695167941
X1^2 X2^3: -0.36480741529365435
X1 X2^4: 0.07594550959566629
X2^5: 0.04622943388873059
```

```
Resultados para C = 100 (alpha = 0.01):
1: 0.0
X1: 0.1253878694848992
X2: -1.1895277271175564
X1^2: 0.7291941546591141
X1 X2: -0.019178745398427267
X2^2: -0.031450822007400826
X1^3: -0.24818398553871107
X1^2 X2: 0.5410744541636359
X1 X2^2: -0.3513422680083947
X2^3: 0.16462425580515216
X1^4: 0.2817195810440601
X1^3 X2: 0.16680928760581012
X1^2 X2^2: 0.12183787435546266
X1 X2^3: -0.08531216292499333
X2^4: -0.003964790518577138
X1^5: 0.13029472167784711
X1^4 X2: -0.11178787274060559
X1^3 X2^2: 0.1367331748914702
X1^2 X2^3: -0.5199920118858639
X1 X2^4: 0.15388466276717294
X2^5: -0.0001173486476393901
```

```
Resultados para C = 1000 (alpha = 0.001):
1: 0.0
X1: 0.12907256174226928
X2: -1.1991814253852489
X1^2: 0.727024015971327
X1 X2: -0.01851426642469072
X2^2: -0.033025809031168855
X1^3: -0.25804405676179965
X1^2 X2: 0.5715461929514433
X1 X2^2: -0.3639368247765604
X2^3: 0.18435629396108624
X1^4: 0.28338985004617795
X1^3 X2: 0.16638139380275377
X1^2 X2^2: 0.12432546618862807
X1 X2^3: -0.08446273799982518
X2^4: -0.002807156283188721
X1^5: 0.13892132560438208
X1^4 X2: -0.12901317260967973
X1^3 X2^2: 0.13599226256891675
X1^2 X2^3: -0.5431823900871334
X1 X2^4: 0.16703853161443788
X2^5: -0.009886841241321428
```

Interpretación:

Recordemos que cuando trabajamos con el modelo de Lasso, para valores bajos de C, o valores altos de alpha, todos los coeficientes eran reducidos a cero. Debido a que esos modelos presentaban una alta penalización.

Pero, a medida que aumentábamos el valor de C, o disminuíamos alpha, íbamos obteniendo valores diferentes a cero para cada vez más coeficientes.

Sin embargo, ahora vemos que, al trabajar con el modelo de regresión de Ridge, obtuvimos resultados diferentes.

En primer lugar, en este caso prácticamente ninguno de los valores de los coeficientes fue cero, para ningún valor de C, fuera alto o bajo.

También vemos que los valores de los coeficientes aumentan a medida que el valor de C aumenta, o que alpha disminuye. Esto se nota sobre todo al cambiar de C=0.1 a C=1, y luego a C=10, lo que nos lleva a concluir que esto ocurre para valores bajos de C o altos de alpha.

Sin embargo, cuando le asignamos valores altos a C, o bajos a alpha, los resultados de los coeficientes mostraron menor variabilidad, es decir ya no variaban como antes, ni aumentaban, ni disminuían de forma significativa.

Además, en todo momento, los coeficientes presentaron valores bastante bajos. Todo esto se debe a varias razones:

La **penalización alpha de Ridge** reduce los valores de los coeficientes. A medida que aumenta, los coeficientes van a tender hacia cero, pero sin llegar a ser cero, como cuando trabajamos con Lasso.

Por medio de esta penalización, Ridge logra **controlar la varianza** que se da como producto de la multicolinealidad, la cual se da cuando hay una alta correlación entre las variables. Esto también nos ayuda a **evitar el sobreajuste**, pero **introduce un sesgo** en las estimaciones de los coeficientes.

En conclusión, podemos decir que el modelo de Ridge **aplica penalización** sobre los coeficientes, por eso al principio vemos que a medida que **reducimos dicha penalización**, aumentando los valores de C, los valores de **los coeficientes aumentan**.

Pero, a diferencia de Lasso, Ridge **mantiene controlados los valores de los coeficientes** y evita que crezcan mucho, lo cual es muy útil cuando trabajamos con datos que presentan **multicolinealidad**, y nos ayuda a **evitar el sobreajuste**.

Este control sobre los valores de los coeficientes se da por medio de la **regularización L2** utilizada por el modelo Ridge. Esta penaliza el cuadrado de los valores de los coeficientes, lo cual tiende a disminuir el valor absoluto de todos los coeficientes de manera uniforme. Esto ayuda a prevenir que el modelo sea excesivamente influenciado por una o más características que tengan una varianza particularmente alta.

3) Predicciones usando los modelos de Ridge

Para poder realizar las predicciones, tuvimos que generar una cuadrícula de valores de características, con los mismos parámetros que la que creamos para las predicciones de Lasso, y también le generaron características polinómicas. Ya con esto listo se procedió con las predicciones.

Cuando trabajamos con Lasso, realizamos las predicciones únicamente para los modelos donde C=1, C=10, C=100 y C=1000, debido a que valores más bajos de C no tenían un efecto significativo sobre los coeficientes del modelo en comparación con C=1, así que usamos este como punto de partida.

Ahora que trabajamos con Ridge, vemos que los valores bajos sí tienen un efecto sobre los coeficientes, pero para mantener la consistencia, de manera que podamos hacer una mejor comparación entre Ridge y Lasso, se realizarán las predicciones para los modelos de Ridge con los mismos valores que Lasso.

Se espera que esto no afecte nuestros resultados, pues ya vimos que la tendencia de aumento en los valores de los coeficientes se dio con los valores bajos hasta llegar a C=10, por lo que todavía podríamos ver el impacto de este auge en los coeficientes, en las predicciones de estos modelos que vamos a utilizar.

```
Predicciones para C = 1 (alpha = 1.0):
[328.20825979 319.26662838 310.2788149  301.41782984 292.83254507]

Predicciones para C = 10 (alpha = 0.1):
[544.87584552 499.31924178 454.46211764 410.75473009 368.58702403]

Predicciones para C = 100 (alpha = 0.01):
[969.06510345 869.58466431 775.06695798 685.61838267 601.31325111]

Predicciones para C = 1000 (alpha = 0.001):
[1060.21537224  950.43819492  846.47355561  748.35111736  656.07510583]
```

4) Gráfica

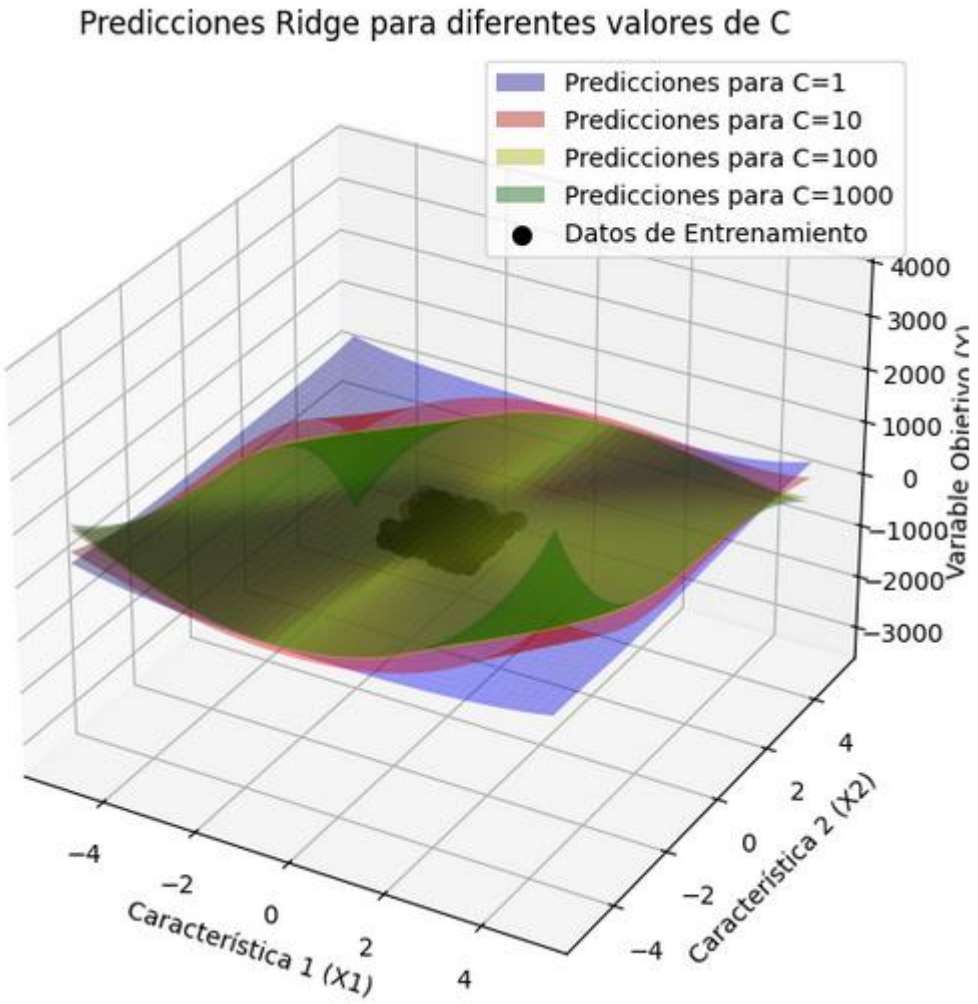


Figura 3: *Gráfico 3D con los* ***datos de entrenamiento*** *y las* ***predicciones*** *para los modelos de regresión Ridge donde* ***C=1****,* ***C=10****,* ***C=100*** *y* ***C=1000***

Interpretación:

- **Predicciones**

En estas gráficas podemos ver representado el control que ejerce el modelo de Ridge sobre los coeficientes. Cuando trabajamos con Lasso, vimos que cuando los valores de los coeficientes eran cero, obteníamos una gráfica totalmente plana, y a medida que la penalización iba disminuyendo, la complejidad del modelo aumentaba y esto se veía representado en como la gráfica iba tomando una forma curva a medida que el valor de C aumentaba, y α disminuía.

A pesar de que Ridge no lleva los valores de los coeficientes a cero, para valores altos de α, o bajos de C, como sí lo hacía Lasso, los resultados obtenidos para los coeficientes del modelo nos demuestran que: para valores bajos de C, obtenemos valores bajos para los coeficientes, los cuales también se representan como una gráfica relativamente plana, no totalmente porque los valores de los coeficientes no son cero.

Al igual que sucede con el modelo de Lasso, a medida que estos valores aumentan, la gráfica va tomando una forma más curva, representando una mayor complejidad en el modelo.

Sin embargo, vimos que Ridge, a diferencia de Lasso, mantiene controlados los coeficientes del modelo para evitar que crezcan demasiado, llegando un punto donde, sin importar que tan baja sea la penalización o que tan alto sea el valor de C, los valores de los coeficientes se mantienen en un rango constante, y esto en la gráfica podemos verlo representado de la siguiente manera:

Cuando las gráficas empiezan a tomar la forma de la curva, esa tiene una forma **convexa**, pero, a medida que aumentamos los coeficientes, estas inician con una forma **cóncava**, pero llega un punto donde empiezan a tomar una forma a cóncava, como si se estuvieran doblando.

Considero que esta es la representación gráfica del control que ejerce Ridge sobre los coeficientes, al no dejarlos crecer demasiado después de cierto punto, lo cual se ve reflejado en las predicciones.

- **Datos de Entrenamiento**

La representación de los datos de entrenamiento en la gráfica para Ridge, es la misma que obtuvimos al trabajar con Lasso. Debido a la diferencia de escalas, los datos de entrenamiento ya no parecen seguir una tendencia lineal, sino que los vemos mucho más concentrados en el centro.

(ii) Usando el modelo Lasso con características polinómicas del apartado (i), ahora utilizarás validación cruzada para seleccionar C.

- **Validación Cruzada**

Es una técnica que se utiliza para **evaluar modelos de machine learning**. Consiste en dividir los datos de entrada disponibles en subconjuntos o iteraciones, y luego entrenar varios modelos de machine learning en esos subconjuntos.

En una validación cruzada de k iteraciones, dividimos nuestros datos en k subconjuntos.

Entrenamos un modelo de todos los subconjuntos, menos uno (k-1). Luego, evaluamos el modelo en el subconjunto que no se utilizó para el entrenamiento. Este proceso se repite k veces, con un subconjunto diferente reservado para la evaluación, y excluido del entrenamiento, cada vez. El siguiente diagrama lo explica:

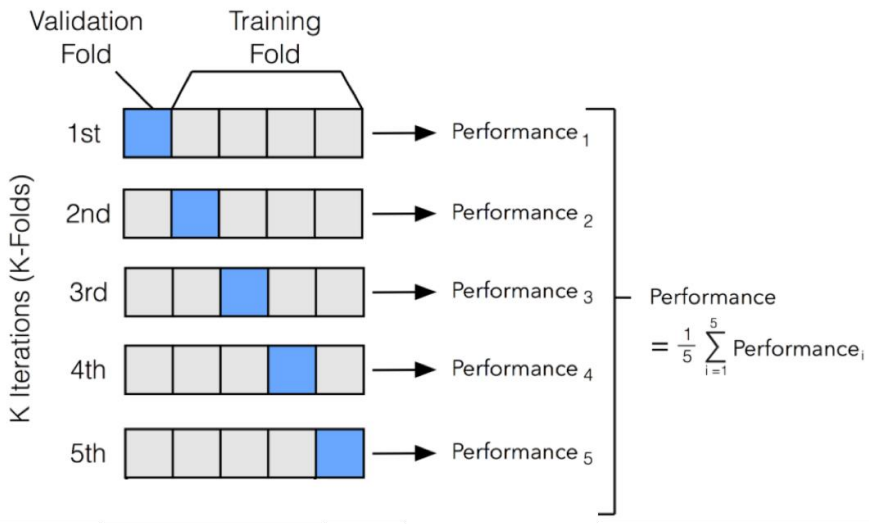


***Figura 4**: Diagrama que representa la validación de K-iteraciones, donde K=5.*

a) Usa validación cruzada de 5 particiones para graficar el promedio y la desviación estándar del error de predicción frente a C. Usa la función errorbar de matplotlib para esto. Necesitarás elegir el rango de valores de C para graficar, justifica tu elección.

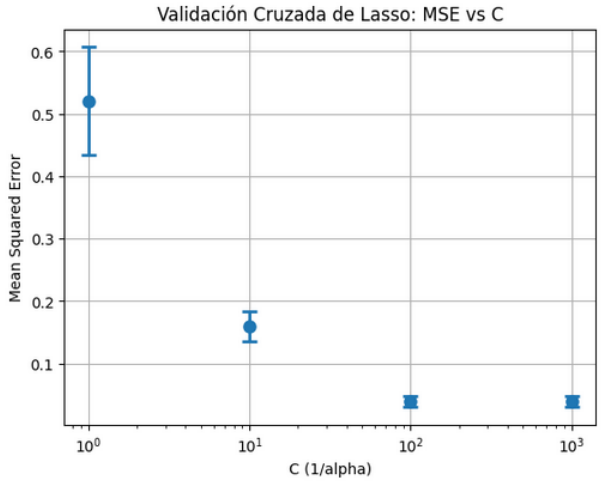
Para realizar la **validación cruzada**, para el **modelo Lasso**, elegí el siguiente rango de valores de C: C=1, C=10, C=100 y C=1000. Esta elección la realicé debido a que estos valores cubren un amplio espectro de valores que me permitirá observar y analizar el rendimiento del modelo desde una regularización alta, hasta una muy baja.

Por otro lado, usar potencias de 10 (10^0, 10^1, 10^2 y 10^3) es una práctica común en la ciencia de datos, ya que nos permite ajustar más fácilmente los modelos y obtener una visión más clara de los efectos de los cambios que realicemos, en este caso, de C. Tal y como lo vemos en la gráfica que presento en el próximo punto.

Además, estos valores los usé al entrenar el modelo de Lasso, por lo que me pareció buena idea mantener los valores.

b) Basándote en los datos de la validación cruzada, ¿qué valor de C recomendarías usar aquí? Es importante que expliques las razones de tu elección.

Luego de aplicar la validación cruzada de 5 particiones para graficar el promedio y la derivación estándar del error de predicción frente a cada valor de C, obtuve los siguientes resultados:




```
Resultados detallados de la validación cruzada para cada C:
C = 1: MSE medio = 0.5200, Desviación estándar = 0.0867
C = 10: MSE medio = 0.1594, Desviación estándar = 0.0246
C = 100: MSE medio = 0.0393, Desviación estándar = 0.0089
C = 1000: MSE medio = 0.0394, Desviación estándar = 0.0080
El valor óptimo de alpha es: 0.0100, correspondiente a C = 100.0000
```

***Figura 5**: Gráfica de Validación Cruzada de Lasso: MSE vs C*

En esta gráfica podemos ver representado el cambio en la desviación estándar (representada por las líneas) y en el MSE promedio (representado por los puntos) a medida que variamos el valor de C. Como podemos ver, tanto la desviación estándar, como el MSE promedio van disminuyendo, a medida que aumentamos el valor de C.

Basándome en estos datos, recomendaría seleccionar el **valor de C=100**, de entre el rango de valores proporcionado, debido a las siguientes razones:

El valor de **C=100** presenta el **valor promedio más bajo del error de predicción** (**MSE medio = 0.0393**), lo cual nos indica que, con este valor de C, el modelo predice **con mayor precisión** que con los demás.

Por otro lado, **la desviación estándar del MSE**, para **C=100**, **es bastante baja**, con un valor de **0.0089**, lo cual nos indica que el modelo es estable y nos brinda resultados consistentes para cada uno de los subconjuntos de datos.

Además, ambos resultados (**el MSE promedio** y la **desviación estándar**), nos dicen que este modelo (**C=100**) es capaz de generalizar muy bien, con un error de predicción y una variabilidad muy bajos.

c) Repite los apartados (a)-(b) para un modelo de regresión Ridge.

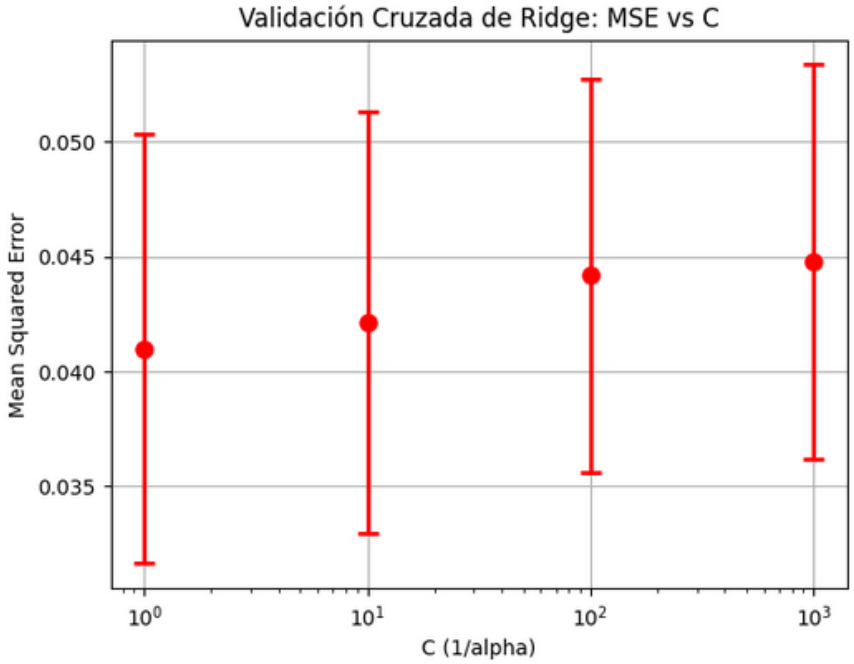
1) **Rango de Valores para C**

Para realizar la **validación cruzada**, para el **modelo Ridge**, elegí **el mismo rango de valores de C** que elegí para la **validación cruzada del modelo Lasso** (**C=1, C=10, C=100 y C=1000**).

Esto lo hice por las mismas razones por las que elegí este rango para la validación cruzada del modelo Lasso en primer lugar. Pero también, para poder comparar mejor mis resultados, ya que de esta manera realizo la validación cruzada bajo las mismas condiciones para ambos modelos.

2) **Análisis de Resultados y Valor de C recomendado**

Luego de aplicar la validación cruzada de 5 particiones para graficar el promedio y la derivación estándar del error de predicción frente a cada valor de C, obtuve los siguientes resultados:




```
Resultados detallados de la validación cruzada para cada C:
C = 1: MSE medio = 0.0410, Desviación estándar = 0.0093
C = 10: MSE medio = 0.0421, Desviación estándar = 0.0092
C = 100: MSE medio = 0.0442, Desviación estándar = 0.0086
C = 1000: MSE medio = 0.0448, Desviación estándar = 0.0086
El valor óptimo de alpha es: 1.0000, correspondiente a C = 1.0000
```

En esta gráfica podemos ver representado el cambio en la desviación estándar (representada por las líneas) y en el MSE promedio (representado por los puntos) a medida que variamos el valor de C. Como podemos ver, tanto la desviación estándar, como el MSE promedio van disminuyendo, a medida que aumentamos el valor de C.

Sin embargo, vemos que **el mismo efecto que tuvo el modelo Ridge** sobre las **predicciones**, y los **coeficientes**, se refleja sobre el **MSE medio** y la **desviación estándar**, pues después de cierto punto, sin importar qué tan alto es el valor de C, la variación en los resultados es **mínima**.

Esto gracias a que Ridge, al mantener controlados los coeficientes, y por ende las predicciones, después de cierto punto, también mantiene controladas su precisión y variabilidad.

Basándome en estos datos, recomendaría seleccionar el **valor de C=1**, de entre el rango de valores proporcionado, debido a las siguientes razones:

El valor de **C=1** presenta el **valor promedio más bajo del error de predicción** (**MSE medio = 0.0410**), lo cual nos indica que, con este valor de C, el modelo predice **con mayor precisión** que con los demás.

Por otro lado, **la desviación estándar del MSE**, para **C=100**, **es bastante baja**, con un valor de **0.0093**, lo cual nos indica que el modelo es estable y nos brinda resultados consistentes para cada uno de los subconjuntos de datos.

Además, ambos resultados (**el MSE promedio** y la **desviación estándar**), nos dicen que este modelo (**C=1**) es capaz de generalizar muy bien, con un error de predicción y una variabilidad muy bajos.

CONCLUSIONES

Se trabajó con el mismo conjunto de datos para ambos modelos, y se establecieron las mismas condiciones en ambos, específicamente, al entrenarlos con los mismos valores de penalización, para poder comparar sus resultados.

La regresión Lasso, por su tendencia a llevar los valores de los coeficientes a cero, nos dio modelos cada vez más complejos a medida que aumentábamos la penalización. Sin embargo, corríamos el riesgo de caer en el sobreajuste si lo aumentábamos mucho. Por medio de la validación cruzada encontramos el valor más adecuado, dentro del rango establecido, para trabajar con estos datos, el cual fue un valor alto (C=100) pero que no nos hacía caer en el sobreajuste, entregándonos resultados precisos, con variabilidad baja, lo cual pudimos confirmar gracias a la validación cruzada.

Hicimos lo mismo con el modelo de Ridge, y este lo que hizo fue que mantuvo controlados los coeficientes, las predicciones, su variabilidad y precisión, incluso para altos valores de C. Por medio de la validación cruzada, se determinó que el valor idea para trabajar con este modelo, con este conjunto de datos en cuestión, sería C=1.

Luego de trabajar ambos modelos, llegué a la siguiente conclusión: El modelo de regresión de Lasso puede ser muy útil, cuando tenemos a nuestra disposición muchas variables, pero queremos determinar y seleccionar solo aquellas que son realmente relevantes para el estudio.

Por otra parte, el modelo de regresión de Ridge puede ser de gran ayuda, cuando trabajamos con muchas variables, y todas son importantes para el estudio, pero no queremos caer en el sobreajuste al trabajar un modelo tan complejo.

En resumen, Ridge puede ser muy útil cuando trabajamos modelos complejos, y Lasso, cuando queremos simplificar un modelo.

Para este caso en particular, considero que, fue mejor trabajar con el modelo Ridge, pues estábamos manejando veintiún (21) variables, y no fue fácil determinar cuáles serían relevantes para el estudio, incluso con Lasso. Además, con los valores utilizados de C, los resultados de Lasso presentaron un cambio algo abrupto entre un modelo simple y uno muy complejo que nos hubiera hecho caer en el sobreajuste.

En cambio, Ridge nos permitió mantener nuestras predicciones, precisión, y variabilidad, bajo control incluso cuando trabajamos con todas las variables, y con penalizaciones altas.

ANEXO

Código:

```
#ASIGNACIÓN 2 parte (i)


#id:7-7--7


import matplotlib.pyplot as plt

import pandas as pd

from mpl_toolkits.mplot3d import Axes3D

from sklearn.model_selection import train_test_split



#Cargamos los datos del archivo: week3


data = pd.read_csv('C:/Users/HP/Documents/ASIGNACIÓN 2 INTRODUCCION A CIENCIA DE DATOS/week3.csv')


X = data.iloc[:, :2].values  # Seleccionamos las dos primeras columnas como nuestras características

Y = data.iloc[:, 2].values  # Seleccionamos la tercera columna como la Variable Objetivo


#Como nos piden ver la tendencia de los datos de entrenamiento, dividimos los datos en entrenamiento y prueba

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)


# Graficamos los datos de entrenamiento en un gráfico de dispersión 3D

fig = plt.figure(figsize = (10, 7))
```

```
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_train[:, 0], X_train[:, 1], Y_train)

ax.set_xlabel('Característica 1')
ax.set_ylabel('Característica 2')
ax.set_zlabel('Objetivo')

plt.show()

#Características Polinómicas Adicionales

from sklearn.preprocessing import PolynomialFeatures #Utilizaremos la función PolynomialFeatures de sklearn

# Usamos PolynomialFeatures para generar características hasta grado 5
poly = PolynomialFeatures(degree=5)

# Definimos cuales son nuestras características en el dataframe
X = data[['X1', 'X2']]

# Transformamos X, para así poder generar y agregar las características polinómicas
X_poly = poly.fit_transform(X)

# Imprimimos nuestras nuevas características polinómicas
print(f"Características polinómicas:\n{poly.get_feature_names_out(['X1', 'X2'])}\n")
print(X_poly[:5])  # Mostramos las primeras cinco filas
```

```
print(f"Original columns: {X.shape[1]}, New columns: {X_poly.shape[1]}")


#Entrenamos el modelo de regresión Lasso con estas características y varios valores de C


from sklearn.linear_model import Lasso


#Definimos nuestro rango de valores de C

#Quise iniciar con valores muy bajos de C y los fui aumentando poco a poco

C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]


#Para trabajar el modelo de regresión Lasso, debemos convertir los valores de C a alpha (alpha = 1/C)


alpha_values = [1/c for c in C_values]


#Para los modelos de regresión de Lasso, el valor C y el valor de alpha (α) son inversamente proporcionales.

#Entonces, si C disminuye, α aumenta, y si C aumenta, α disminuye.


# Entrenamos un modelo para cada valor de C

for alpha, C in zip(alpha_values, C_values):


    # Creamos el modelo Lasso con el valor de alpha correspondiente

    model = Lasso(alpha=alpha, max_iter=10000)

    # Con max_iter ajustamos el número máximo de iteraciones a 10,000.
```

```
    #De esta manera, nos aseguramos que el algoritmo tenga suficiente tiempo para encontrar la mejor solución


    # Entrenamos el modelo con las características polinómicas y la variable objetivo
    model.fit(X_poly, data['Y'])


    # Imprimimos los coeficientes que obtuvimos para el modelo
    print(f"\nResultados para C = {C} (alpha = {alpha}):")
    for coef, feature in zip(model.coef_, poly.get_feature_names_out()):
        print(f"{feature}: {coef}")



import numpy as np


#Primero, creamos un conjunto de 50 valores entre -5 y 5.
#Usamos el rango de -5 a 5 para cubrir los posibles valores de las características.
grid = np.linspace(-5, 5, 50)


X_test = []
# Creamos una lista vacía donde vamos a almacenar todas las combinaciones de los valores de las características.


#Generamos dos bucles: El primero toma un valor i del conjunto de datos grid. El segundo, toma un valor j.
for i in grid:
    for j in grid:
        X_test.append([i, j])
```

#Cada par de valores (i, j), los colocamos juntos en una lista [i, j] y los añadimos a X_test usando append().

```
X_test = np.array(X_test)
```

#Convertimos la lista X_test en un arreglo de numpy usando np.array(), para poder hacer predicciones usando los valores de la lista.

```
#Veamos las primeras 5 filas de la cuadrícula
print(X_test[:5])

#GENERAR CARACTERISTICAS POLINOMICAS PARA LA CUADRICULA

# Generamos las características polinómicas para el conjunto X_test
poly = PolynomialFeatures(degree=5)
Xtest_poly = poly.fit_transform(X_test)

# Obtenemos los nombres de las características polinómicas generadas por PolynomialFeatures
feature_names = poly.get_feature_names_out(['X1', 'X2'])

# Creamos el DataFrame usando los nombres de las características polinómicas
Xtest_poly_df = pd.DataFrame(Xtest_poly, columns=feature_names)

# Mostramos las primeras filas del DataFrame resultante
print(Xtest_poly_df.head())

# PREDICCIONES
```

```
# Anteriormente, se generaron varios modelos, utilizando diferentes valores de C.
# Vamos a generar las predicciones para los modelos donde C=1, C=10, C=100 y C=1000

C_values = [1, 10, 100, 1000]

# Convertimos los valores de C a alpha
alpha_values = [1/c for c in C_values]

# Aquí almacenamos las predicciones
predictions = {}

# Obtenemos los nombres de las características polinómicas
feature_names = poly.get_feature_names_out(['X1', 'X2'])

# Convertimos X_poly a un DataFrame, usando los nombres de las características
X_poly_df = pd.DataFrame(X_poly, columns=feature_names)

# Creamos un modelo Lasso para cada valor de C
for alpha, C in zip(alpha_values, C_values):

    # Creamos el modelo Lasso
    model = Lasso(alpha=alpha, max_iter=10000)

    # Entrenamos cada modelo con las características polinómicas y la variable objetivo
    model.fit(X_poly_df, data['Y'])
```

```
    # Realizamos predicciones sobre el conjunto de prueba polinómico
    y_pred = model.predict(Xtest_poly_df)

    # Guardamos las predicciones
    predictions[f'Predicciones para C={C}'] = y_pred

    # Imprimimos algunas predicciones para ver los resultados
    print(f"\nPredicciones para C = {C} (alpha = {alpha}):")
    print(y_pred[:5])  # Mostramos las primeras 5 predicciones

# Crear la figura y el gráfico 3D
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Definimos los colores para los diferentes modelos: C=1, C=10, C=100 y C=1000
colors = {1: 'blue', 10: 'red', 100: 'yellow', 1000: 'green'}

# Necesitamos crear una malla de puntos para graficar las superficies
X1 = np.linspace(X_test[:, 0].min(), X_test[:, 0].max(), 50)
X2 = np.linspace(X_test[:, 1].min(), X_test[:, 1].max(), 50)
X1, X2 = np.meshgrid(X1, X2)

# Ajustamos las predicciones al tamaño de la malla para su uso en plot_surface
from scipy.interpolate import griddata
# Creamos un array con las coordenadas de los puntos originales para la interpolación
points = np.column_stack((X_test[:, 0], X_test[:, 1]))
```

```
# Graficamos las predicciones de cada modelo como superficies
for C in [1, 10, 100, 1000]:
    # Interpolamos los datos al grid
    y_pred = griddata(points, predictions[f'Predicciones para C={C}'], (X1, X2), method='cubic')
    ax.plot_surface(X1, X2, y_pred, color=colors[C], label=f'Predicciones para C={C}', alpha=0.4)

# Graficamos los datos de entrenamiento originales como puntos para contraste
ax.scatter(X_train[:, 0], X_train[:, 1], Y_train, color='black', label='Datos de Entrenamiento', s=50)

#### ELEMENTOS DEL GRÁFICO

# Nombres de los ejes
ax.set_xlabel('Característica 1 (X1)')  # Eje X
ax.set_ylabel('Característica 2 (X2)')  # Eje Y
ax.set_zlabel('Variable Objetivo (Y)')  # Eje Z

# Título del gráfico
ax.set_title('Predicciones Lasso para diferentes valores de C')

# Agregar leyenda para clarificar el gráfico
ax.legend()

# Mostrar el gráfico
plt.show()
```

```
#APARTADOS B Y C, CON UN MODELO DE REGRESIÓN RIDGE

from sklearn.linear_model import Ridge

#Apartado (b): Características Polinómicas y Entrenamiento de Modelos Ridge

#Características Polinómicas

poly = PolynomialFeatures(degree=5)
X = data[['X1', 'X2']]  # Seleccionamos las características
X_poly = poly.fit_transform(X)  # Transformación polinómica

# Imprimimos las características polinómicas generadas
print(f"Características polinómicas:\n{poly.get_feature_names_out(['X1', 'X2'])}\n")
print(X_poly[:5])
print(f"Original columns: {X.shape[1]}, New columns: {X_poly.shape[1]}")



# Entrenamiento del modelo Ridge con varios valores de C
C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
alpha_values = [1/c for c in C_values]  # Conversión de C a alpha

# Entrenamos un modelo Ridge para cada valor de C
for alpha, C in zip(alpha_values, C_values):
    model = Ridge(alpha=alpha, max_iter=10000)
```

```
    model.fit(X_poly, data['Y'])  # Entrenamiento del modelo
    print(f"\nResultados para C = {C} (alpha = {alpha}):")
    for coef, feature in zip(model.coef_, poly.get_feature_names_out()):
        print(f"{feature}: {coef}")

#Apartado (c): Generación Predicciones con Ridge

#CREACIÓN DE LA CUADÍCULA
import numpy as np

#Primero, creamos un conjunto de 50 valores entre -5 y 5.
#Usamos el rango de -5 a 5 para cubrir los posibles valores de las características.
grid = np.linspace(-5, 5, 50)

X_test = []
# Creamos una lista vacía donde vamos a almacenar todas las combinaciones de los valores de las características.

#Generamos dos bucles: El primero toma un valor i del conjunto de datos grid. El segundo, toma un valor j.
for i in grid:
    for j in grid:
        X_test.append([i, j])
        #Cada par de valores (i, j), los colocamos juntos en una lista [i, j] y los añadimos a X_test usando append().

X_test = np.array(X_test)
```

#Convertimos la lista X_test en un arreglo de numpy usando np.array(), para poder hacer predicciones usando los valores de la lista.

#Veamos las primeras 5 filas de la cuadrícula

print(X_test[:5])

#GENERAR CARACTERISTICAS POLINOMICAS PARA LA CUADRICULA

Generamos las características polinómicas para el conjunto X_test

poly = PolynomialFeatures(degree=5)

Xtest_poly = poly.fit_transform(X_test)

Obtenemos los nombres de las características polinómicas generadas por PolynomialFeatures

feature_names = poly.get_feature_names_out(['X1', 'X2'])

Creamos el DataFrame usando los nombres de las características polinómicas

Xtest_poly_df = pd.DataFrame(Xtest_poly, columns=feature_names)

Mostramos las primeras filas del DataFrame resultante

print(Xtest_poly_df.head())

PREDICCIONES

Anteriormente, se generaron varios modelos, utilizando diferentes valores de C.

Para mantener la consistencia con el modelo Lasso, y así poder comparar los resultados

```
#Vamos a generar las predicciones para los modelos donde C=1, C=10, C=100 y C=1000

C_values = [1, 10, 100, 1000]

# Convertimos los valores de C a alpha
alpha_values = [1/c for c in C_values]

# Aquí almacenamos las predicciones
predictions = {}

# Obtenemos los nombres de las características polinómicas
feature_names = poly.get_feature_names_out(['X1', 'X2'])

# Convertimos X_poly a un DataFrame, usando los nombres de las características
X_poly_df = pd.DataFrame(X_poly, columns=feature_names)

# Creamos un modelo Ridge para cada valor de C
for alpha, C in zip(alpha_values, C_values):

    # Creamos el modelo Ridge
    model = Ridge(alpha=alpha, max_iter=10000)

    # Entrenamos cada modelo con las características polinómicas y la variable objetivo
    model.fit(X_poly_df, data['Y'])

    # Realizamos predicciones sobre el conjunto de prueba polinómico
```

```
    y_pred = model.predict(Xtest_poly_df)

    # Guardamos las predicciones
    predictions[f'Predicciones para C={C}'] = y_pred

    # Imprimimos algunas predicciones para ver los resultados
    print(f"\nPredicciones para C = {C} (alpha = {alpha}):")
    print(y_pred[:5])  # Mostramos las primeras 5 predicciones



#GRÁFICA DE LAS PREDICCIONES DE RIDGE

# Crear la figura y el gráfico 3D
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Definimos los colores para los diferentes modelos: C=1, C=10, C=100 y C=1000
colors = {1: 'blue', 10: 'red', 100: 'yellow', 1000: 'green'}

# Necesitamos crear una malla de puntos para graficar las superficies
X1 = np.linspace(X_test[:, 0].min(), X_test[:, 0].max(), 50)
X2 = np.linspace(X_test[:, 1].min(), X_test[:, 1].max(), 50)
X1, X2 = np.meshgrid(X1, X2)

# Ajustamos las predicciones al tamaño de la malla para su uso en plot_surface
from scipy.interpolate import griddata
```

```
# Creamos un array con las coordenadas de los puntos originales para la interpolación
points = np.column_stack((X_test[:, 0], X_test[:, 1]))

# Graficamos las predicciones de cada modelo como superficies
for C in [1, 10, 100, 1000]:
    # Interpolamos los datos al grid
    y_pred = griddata(points, predictions[f'Predicciones para C={C}'], (X1, X2), method='cubic')
    ax.plot_surface(X1, X2, y_pred, color=colors[C], label=f'Predicciones para C={C}', alpha=0.4)

# Graficamos los datos de entrenamiento originales como puntos para contraste
ax.scatter(X_train[:, 0], X_train[:, 1], Y_train, color='black', label='Datos de Entrenamiento', s=50)

#### ELEMENTOS DEL GRÁFICO

# Nombres de los ejes
ax.set_xlabel('Característica 1 (X1)')  # Eje X
ax.set_ylabel('Característica 2 (X2)')  # Eje Y
ax.set_zlabel('Variable Objetivo (Y)')  # Eje Z

# Título del gráfico
ax.set_title('Predicciones Ridge para diferentes valores de C')

# Agregar leyenda para clarificar el gráfico
ax.legend()
```

```
# Mostrar el gráfico
plt.show()



#ASIGNACION 2 parte (ii)


import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge


#Cargamos los datos del archivo: week3


data = pd.read_csv('C:/Users/HP/Documents/ASIGNACIÓN 2 INTRODUCCION A CIENCIA DE DATOS/week3.csv')


X = data.iloc[:, :2].values  # Seleccionamos las dos primeras columnas como nuestras características
Y = data.iloc[:, 2].values  # Seleccionamos la tercera columna como la Variable Objetivo
```

```
#Dividimos los datos en entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

print(data.head())

#Características Polinómicas Adicionales

from sklearn.preprocessing import PolynomialFeatures #Utilizaremos la función PolynomialFeatures de sklearn

# Usamos PolynomialFeatures para generar características hasta grado 5
poly = PolynomialFeatures(degree=5)

# Definimos cuales son nuestras características en el dataframe
X = data[['X1', 'X2']]

# Transformamos X, para así poder generar y agregar las características polinómicas
X_poly = poly.fit_transform(X)

# Imprimimos nuestras nuevas características polinómicas
print(f"Características polinómicas:\n{poly.get_feature_names_out(['X1', 'X2'])}\n")
print(X_poly[:5])  # Mostramos las primeras cinco filas

print(f"Original columns: {X.shape[1]}, New columns: {X_poly.shape[1]}")
```

#Usamos el mismo modelos de regresión de Lasso con características polinómicas que desarrollamos en (i)

#Definimos nuestro rango de valores de C

C_values = [1, 10, 100, 1000]

#Para trabajar el modelo de regresión Lasso, debemos convertir los valores de C a alpha (alpha = 1/C)

alpha_values = [1/c for c in C_values]

#Para los modelos de regresión de Lasso, el valor C y el valor de alpha (α) son inversamente proporcionales.

#Entonces, si C disminuye, α aumenta, y si C aumenta, α disminuye.

Entrenamos un modelo para cada valor de C

for alpha, C in zip(alpha_values, C_values):

    # Creamos el modelo Lasso con el valor de alpha correspondiente

    model = Lasso(alpha=alpha, max_iter=10000)

    param_grid = {'alpha': alpha_values}

Entrenamos el modelo con las características polinómicas y la variable objetivo

    model.fit(X_poly, data['Y'])

    # Imprimimos los coeficientes que obtuvimos para el modelo

    print(f"\nResultados para C = {C} (alpha = {alpha}):")

```
    for coef, feature in zip(model.coef_, poly.get_feature_names_out()):
        print(f"{feature}: {coef}")

#APARTADOS A Y B (LASSO)

#Usando el modelo Lasso con características polinómicas, ahora utilizaremos validación cruzada para seleccionar C.

#(a) Usa validación cruzada de 5 particiones para graficar el promedio y la desviación estándar del error de predicción frente a C.
#Usa la función errorbar de matplotlib para esto. Necesitarás elegir el rango de valores de C para graficar, justifica tu elección.

# Usamos los mismos valores de C con los que entrenamos el modelo Lasso
C_values = [1, 10, 100, 1000]
alpha_values = [1 / c for c in C_values]

# Calculamos el MSE usando validación cruzada de 5 particiones
mse_means = []
mse_stds = []
for alpha in alpha_values:
    model.alpha = alpha  # Actualiza el valor de alpha en el modelo existente
    results = cross_validate(model, X_poly, Y, scoring='neg_mean_squared_error', cv=5, return_train_score=False)
    mse_scores = -results['test_score']  # Convertimos los scores a positivos
    mse_means.append(np.mean(mse_scores))
    mse_stds.append(np.std(mse_scores))
```

```
# Graficamos el promedio y la desviación estándar del error de predicción frente a C usando errorbar
plt.errorbar(C_values, mse_means, yerr=mse_stds, fmt='o', markersize=8, elinewidth=2, capsize=5, capthick=2)
plt.xscale('log')
plt.xlabel('C (1/alpha)')
plt.ylabel('Mean Squared Error')
plt.title('Validación Cruzada de Lasso: MSE vs C')
plt.grid(True)
plt.show()

# Resultados de la validación cruzada para cada C
print("Resultados detallados de la validación cruzada para cada C:")
for C, mean, std in zip(C_values, mse_means, mse_stds):
    print(f"C = {C}: MSE medio = {mean:.4f}, Desviación estándar = {std:.4f}")

# El mejor valor de alpha y C
best_index = np.argmin(mse_means)  # Índice del menor MSE medio
best_alpha = alpha_values[best_index]
best_C = 1 / best_alpha
print(f"El valor óptimo de alpha es: {best_alpha:.4f}, correspondiente a C = {best_C:.4f}")

#APARTADO C

#Características Polinómicas y Entrenamiento de Modelos Ridge

#Características Polinómicas
```

```
poly = PolynomialFeatures(degree=5)
X = data[['X1', 'X2']]  # Seleccionamos las características
X_poly = poly.fit_transform(X)  # Transformación polinómica

# Imprimimos las características polinómicas generadas
print(f"Características polinómicas:\n{poly.get_feature_names_out(['X1', 'X2'])}\n")
print(X_poly[:5])
print(f"Original columns: {X.shape[1]}, New columns: {X_poly.shape[1]}")

# Entrenamiento del modelo Ridge con varios valores de C
C_values = [1, 10, 100, 1000]
alpha_values = [1/c for c in C_values]  # Conversión de C a alpha

# Entrenamos un modelo Ridge para cada valor de C
for alpha, C in zip(alpha_values, C_values):
    model = Ridge(alpha=alpha, max_iter=10000)
    model.fit(X_poly, data['Y'])  # Entrenamiento del modelo
    print(f"\nResultados para C = {C} (alpha = {alpha}):")
    for coef, feature in zip(model.coef_, poly.get_feature_names_out()):
        print(f"{feature}: {coef}")

# Usamos los mismos valores de C con los que entrenamos el modelo
C_values = [1, 10, 100, 1000]
alpha_values = [1 / c for c in C_values]

# Modelo Ridge
```

```
model = Ridge(max_iter=10000)  # Se establece un número alto de iteraciones para asegurar la convergencia


# Calculamos el MSE usando validación cruzada de 5 particiones
mse_means = []
mse_stds = []
for alpha in alpha_values:
    model.alpha = alpha  # Actualiza el valor de alpha en el modelo existente
    results = cross_validate(model, X_poly, Y, scoring='neg_mean_squared_error', cv=5, return_train_score=False)
    mse_scores = -results['test_score']  # Convertimos los scores a positivos
    mse_means.append(np.mean(mse_scores))
    mse_stds.append(np.std(mse_scores))


# Graficamos el promedio y la desviación estándar del error de predicción frente a C usando errorbar
plt.errorbar(C_values, mse_means, yerr=mse_stds, fmt='o', markersize=8, elinewidth=2, capsize=5, capthick=2, color='red')
plt.xscale('log')
plt.xlabel('C (1/alpha)')
plt.ylabel('Mean Squared Error')
plt.title('Validación Cruzada de Ridge: MSE vs C')
plt.grid(True)
plt.show()


# Resultados de la validación cruzada para cada C
print("Resultados detallados de la validación cruzada para cada C:")
for C, mean, std in zip(C_values, mse_means, mse_stds):
```

```
    print(f"C = {C}: MSE medio = {mean:.4f}, Desviación estándar = {std:.4f}")


# El mejor valor de alpha y C

best_index = np.argmin(mse_means)  # Índice del menor MSE medio

best_alpha = alpha_values[best_index]

best_C = 1 / best_alpha

print(f"El valor óptimo de alpha es: {best_alpha:.4f}, correspondiente a C = {best_C:.4f}")
```