Elección Discreta: Teoría, estimación y métodos númericos

Tarea 2[*]

Germán Augusto Campos Ortíz[**] Juan José Merino Zarco[***]

28 de febrero de 2022

1. Pregunta 1: Nested Logit

1.1. Inciso 1)

Descripción: Extiende la implementación del logit condicional en la Tarea 1 para implementar el nested logit usando los datos de la base de datos "yougurt.csv". Normaliza la utilidad de la alternativa 4, seleccionando $\alpha_4 = 0$. Tu modelo debe tener dos nidos y debes estimarlo usando todas las combinaciones posibles de nidos, i.e., todas las combinaciones (3,1) y (2,2) con los números indicando la cantidad de alternativas en cada nido. Con una estructura (3,1), establece $\lambda_1 = 1$ para el nido unitario. Con una estructura de nidos (2,2) permita correlaciones especificas diferentes, esto es $\lambda_1 \neq \lambda_2$.

Solución: Se establecen 7 combinaciones en total de los nidos, 4 correspondientes a la estructura del nido (3,1) y los 3 restantes que pertenecen al nido (2,2). Las combinaciones del nido (3,1) son: (Bien 1-2-3; Bien 4), (Bien 1-2-4, Bien 3), (Bien 1-3-4, Bien 2), y (Bien 2-3-4, Bien 1). Por su parte, el nido (2,2) presenta las siguientes combinaciones: (Bien 1-2, Bien 3-4), (Bien 1-3, Bien 2-4), y (Bien 1-4, Bien 2-3).

En el código en Python, se crearon 7 funciones distintas para cada una de las combinaciones, con un sistema de código aplicando vectorización con el fin de promover la eficiencia computacional. A partir de estas funciones se puede realizar la optimización con la paqueteria

[*]El Colegio de México 2020-2022. Profesor: Edwin Muñoz Rodríguez

[**]El Colegio de México, gacampos@colmex.mx
[***]El Colegio de México, jmerino@colmex.mx

scipy.optimize para así obtener las estimaciones de los parametros.

La estructura del código de las funciones se presentan en el anexo del presente documento y también se envían en formato adjunto tipo .py.

Es importante mencionar que para la estructura del nido (3,1) fijamos $\lambda_2 = 1$ para el nido unitario, y no λ_1 como lo dice el ejercicio. Lo hicimos de esta forma sabiendo que no hay diferencia en los resultados, puesto que sea λ_1 o sea λ_2, lo importante es que esté fijado siendo igual a 1 en caso del nido unitario.

1.2. Inciso 2)

Descripción: Crea una tabla para comparar estos estimados con los obtenidos usando el logit condicional. Incluye los estimadores de los parámetros y sus errores estándar, así como el índice de verosimilitud y AIC. Para obtener los errores estándar de λ_1, λ_2 utiliza el método delta. Con base en los resultados, ¿qué modelo es preferido y por qué? ¿Qué concluyes sobre los patrones de sustitución a partir de los estimados de λ?

Solución: Tras definir las funciones mencionadas en el inciso anterior, se crearon tres funciones para la optimización y la solución de los items requeridos en el problema. La primera función será para el logit condicional ya realizado en la tarea 1, y las otra dos funciones corresponden a la solución de las distintas combinaciones de cada nido en el logit nested. Cada función fue creada para generar un dataframe con todas las soluciones que se generan a partir de diferentes formas de combinaciones de nidos, métodos y puntos iniciales. Cabe recalcar que, en el caso del primer dataframe, no habrán combinaciones de nidos ya que corresponde al logit multinominal. Entonces, se cosntruyen tres dataframe que luego son unidos en un solo y finalmente este se transforma en un documento csv con la solución completa del ejercicio.

La función del logit condicional permite crear un dataframe en el que las filas serán los distintos métodos con diferentes puntos iniciales considerados, y las columnas serán todas las estimaciones de los parámetros de interés $(\alpha_1, \alpha_2, \alpha_3, \beta_p, \beta_f)$ junto al optimo de maxima verosimilitud, el AIC y los errores estándar de los estimadores. El nombre del método y del punto inicial a su vez son creados en columnas para poder identificar la correspondencia de cada fila

Por su parte, el orden y la lógica de las dos funciones del logit nested es el siguiente. Primero, se crea un dataframe vacío de nombre *conca*. Luego, se crean 4 objetos en los cuales se genera la minimización de las cuatro combinaciones del nido (3,1); en el caso del nido (2,2) serían 3 objetos. El valor del punto inicial y el nombre del método son parámetros al interior de la función, lo que luego nos va a permitir crear filas distintas para cada combinación, método y punto inicial en el documento csv final. Posteriormente, se extraen las estimaciones óptimas de los estimadores de interés ($\alpha_1, \alpha_2, \alpha_3, \beta_p, \beta_f$, γ_1, γ_2) y estos valores se añaden al dataframe *conca* en columnas diferentes para cada parámetro. Es importante acotar que en el nido (3,1) solo se estima uno de los γ, mientras que el nido (2,2) se estimarán los dos.

Bajo este mismo razonamiento, a lo largo de la función se van creando nuevas columnas que corresponderán a los valores de máxima verosimilitud en el óptimo, así como los AIC, la razón de verosimilitud, los errores estándar de los estimadores y los errores estándar con el metodo delta para λ_1 y λ_2. Así mismo, los valores óptimos de λ_1 y λ_2 son agregados en nuevas columnas a partir de las estimaciones de γ_1 y γ_2. Finalmente, dependiendo del método, la combinación y el punto inicial, se generan nuevas columnas que indicarán de forma correcta la combinación de nido, el método usado y el punto inicial correspondiente a cada fila.

En el código posterior, uno a uno se van generando las funciones anteriores con los métodos y los puntos iniciales que escogimos, y sucesivamente se van creando los tres dataframes expuestos anteriormente para luego finalmente construir un solo dataframe y exportarlo en un documento csv.

Se ejecutaron los procesos de optimización con 5 métodos de maximización y dos puntos iniciales. Los métodos fueron: trust-constr, L-BFGS-B, Nelder-Mead, Powell, y TNC. En general, los resultados fueron consistentes ante los diversos métodos y ante cambios en el punto de inicio.

En las siguientes páginas presentaremos tablas de resumen con los principales resultados encontrados para dos de los métodos estimados. Sin embargo, todos los resultados completos podrán apreciarse al ejecutar el código completo en nuestro archivo con formato .py.

Cuadro 1: Estimaciones con el método trust-constr

	(123, 4)	**(124, 3)**	**(134, 2)**	**(234, 1)**	**(12,34)**	**(13,24)**	**(14, 23)**	**Logit_con**
alph1	1.38167	1.38775	1.34085	1.38775	1.30841	1.38126	1.27044	1.38775
alph2	0.83942	0.64350	0.59189	0.64351	0.73436	0.64231	0.59673	0.64350
alph3	-1.65849	-3.08612	-2.94856	-3.08612	-1.92961	-2.84710	-1.95511	-3.08612
bp	-26.58103	-37.05788	-36.00367	-37.05794	-28.19590	-36.50923	-33.75528	-37.05793
bf	0.37447	0.48741	0.48401	0.48741	0.38708	0.49152	0.44043	0.48741
gamma1	0.59008	13.50171	2.83703	27.88980	0.85517	2.21567	2.40116	
gamma2					0.10108	35.30639	0.24406	
lambda1	0.64338	1.00000	0.94464	1.00000	0.70165	0.90165	0.91692	
lambda2	1.00000	1.00000	1.00000	1.00000	0.52525	1.00000	0.56071	
valor_opt	-2653.76460	-2658.55671	-2658.45642	-2658.55670	-2654.09840	-2658.39663	-2652.85126	-2658.55670
Metodo	trust-constr	trust-constr	trust-constr	trust-constr	trust-constr	trust-constr	trust-constr	trust-constr
AIC	5319.52920	5329.11342	5328.91285	5329.11340	5322.19680	5330.79327	5319.70253	5327.11340
Indice Verosimilitud	0.23596	0.28879	0.17987	0.19938	0.21213	0.21085	0.21250	0.21081
alph1_SE	0.07205	0.08805	0.13448	0.08805	0.07906	0.08836	0.12802	0.08805
alph2_SE	0.07530	0.05448	0.12415	0.05448	0.06978	0.05450	0.11963	0.05448
alph3_SE	0.43384	0.14491	0.33082	0.14491	0.31637	0.41332	0.24403	0.14491
bp_SE	3.88357	2.39949	3.32996	2.39946	3.77972	2.57476	3.11067	2.39946
bf_SE	0.09984	0.11992	0.11672	0.11992	0.10209	0.11808	0.10777	0.11992
gamma1_SE	0.43920	267.61055	2.29303	262911.13970	0.50723	1.80003	1.64627	
gamma2_SE					0.48186	1188664.61000	0.33898	
ee_lambda1	0.10077	0.00037	0.11991	0.00000	0.10618	0.15962	0.12541	
ee_lambda2					0.12016	0.00000	0.08350	

Cuadro 2: Estimaciones con el método L-BFGS-B

	(123, 4)	**(124, 3)**	**(134, 2)**	**(234, 1)**	**(12,34)**	**(13,24)**	**(14, 23)**	**Logit_con**
alph1	1.38166	1.38765	1.34087	1.38748	1.30842	1.38101	1.27044	1.38772
alph2	0.83936	0.64344	0.59190	0.64331	0.73436	0.64195	0.59673	0.64349
alph3	-1.65885	-3.08618	-2.94863	-3.08554	-1.92963	-2.84397	-1.95512	-3.08609
bp	-26.58327	-37.05545	-36.00390	-37.06099	-28.19647	-36.51730	-33.75545	-37.05710
bf	0.37445	0.48735	0.48408	0.48724	0.38708	0.49108	0.44043	0.48743
gamma1	0.59032	9.19682	2.83742	7.97400	0.85520	2.19614	2.40116	
gamma2					0.10110	7.34787	0.24408	
lambda1	0.64344	0.99990	0.94466	0.99966	0.70166	0.89990	0.91692	
lambda2	1.00000	1.00000	1.00000	1.00000	0.52525	0.99936	0.56072	
valor_opt	-2653.76460	-2658.55773	-2658.45642	-2658.56263	-2654.09840	-2658.41683	-2652.85126	-2658.55670
Metodo	L-BFGS-B	L-BFGS-B	L-BFGS-B	L-BFGS-B	L-BFGS-B	L-BFGS-B	L-BFGS-B	L-BFGS-B
AIC	5319.52920	5329.11546	5328.91285	5329.12525	5322.19680	5330.83365	5319.70253	5327.11340
Indice Verosimilitud	0.23596	0.28879	0.17987	0.19938	0.21213	0.21085	0.21250	0.21081
alph1_SE	0.07205	0.08814	0.13448	0.08826	0.07906	0.08854	0.12802	0.08805
alph2_SE	0.07529	0.05451	0.12415	0.05453	0.06978	0.05453	0.11963	0.05448
alph3_SE	0.43380	0.14491	0.33082	0.14525	0.31636	0.40825	0.24403	0.14491
bp_SE	3.88304	2.40172	3.32994	2.39999	3.77969	2.57176	3.11072	2.39943
bf_SE	0.09984	0.11992	0.11673	0.11991	0.10209	0.11803	0.10777	0.11992
gamma1_SE	0.43916	31.08623	2.29386	12.97687	0.50722	1.74762	1.64630	
gamma2_SE					0.48185	7.05041	0.33899	
ee_lambda1	0.10075	0.00315	0.11991	0.00446	0.10618	0.15742	0.12542	
ee_lambda2					0.12016	0.00453	0.08350	

Con base en los resultados, el modelo preferido es aquel modelo que tiene menor valor en el criterio de información de Akaike (AIC), el cual es el modelo nested-logid con la estructura de nidos (3,1) en el que los bienes 1, 2 y 3 están en un nido, y el bien 4 está en el nido unitario. En las tablas anteriores, este modelo está en la columna de nombre (123, 4). Este modelo, a su vez, es más preferido que el logit condicional, el cual tiene un mayor valor de AIC.

En este mejor modelo hay una correlación entre los componentes no observados de la utilidad de las alternativas dentro del nido 1, es decir, el nido en donde están los productos 1, 2 y 3. La medida correlación $1 - \lambda_k$ es de 0.3566. Esta medida de correlación es baja, sin embargo, podemos asegurar que existe un cierto grado de sustitución entre las alternativas en este nido de tres productos. A su vez, existe independencia de alternativa irrelevantes al interior del nido, pero no entre nidos.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 22 00:44:10 2022

 German Campos y Juan Merino
"""

#Modules

import pandas as pd
import numpy as np
from scipy.optimize import minimize
import os
import numdifftools as nd


#Working directory
os.chdir('D:/Usuario/Desktop/Cuarto Semestre/Elección discreta/Tarea 2')

#Import raw data
df=pd.read_csv("yogurt.csv")


# Creating functions

# Logit condicional

def LL(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf=beta

    #Utilidad representativa
    v1=alpha1+betap*df['price1']+betaf*df['feat1']
    v2=alpha2+betap*df['price2']+betaf*df['feat2']
    v3=alpha3+betap*df['price3']+betaf*df['feat3']
    v4=betap*df['price4']+betaf*df['feat4']

    #Exponencial de utilidades representativas
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
```

```
    expv4=v4.apply(np.exp)

    #Inclusive value (denominador)
    inclvalue=expv1+expv2+expv3+expv4

    #Choice probabilities
    prob1=expv1/inclvalue
    prob2=expv2/inclvalue
    prob3=expv3/inclvalue
    prob4=expv4/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)


# Nested Logit

## Structure (3,1)

def LL_N_31a(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = 1


    #Representative utilities
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda1
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda1
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda1
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda2


    #Exp representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
```

```
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a=expv1+expv2+expv3
    b=expv4

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(a)**(lambda1-1))/inclvalue
    prob2=(expv2*(a)**(lambda1-1))/inclvalue
    prob3=(expv3*(a)**(lambda1-1))/inclvalue
    prob4=(expv4*(b)**(lambda2-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)


def LL_N_31b(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = 1



    #Representative utilities
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda1
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda1
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda2
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda1
```

```
    #Exp Representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a=expv1+expv2+expv4
    b=expv3

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(a)**(lambda1-1))/inclvalue
    prob2=(expv2*(a)**(lambda1-1))/inclvalue
    prob3=(expv3*(b)**(lambda2-1))/inclvalue
    prob4=(expv4*(a)**(lambda1-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)

def LL_N_31c(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = 1


    #Representative utilities
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda1
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda2
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda1
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda1
```

```
    #Exp Representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a=expv1+expv4+expv3
    b=expv2

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(a)**(lambda1-1))/inclvalue
    prob2=(expv2*(b)**(lambda2-1))/inclvalue
    prob3=(expv3*(a)**(lambda1-1))/inclvalue
    prob4=(expv4*(a)**(lambda1-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)

def LL_N_31d(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = 1


    #Utilidad representativa
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda2
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda1
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda1
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda1
```

```
    #Exp Representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a=expv4+expv2+expv3
    b=expv1

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(b)**(lambda2-1))/inclvalue
    prob2=(expv2*(a)**(lambda1-1))/inclvalue
    prob3=(expv3*(a)**(lambda1-1))/inclvalue
    prob4=(expv4*(a)**(lambda1-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)

## Estructura (2,2)

def LL_N_22a(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1,gamma2=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = np.exp(gamma2)/(np.exp(gamma2) + 1)

    #Representative utilities
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda1
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda1
```

```
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda2
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda2


    #Exp Representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a = expv1 + expv2
    b = expv3 + expv4

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(a)**(lambda1-1))/inclvalue
    prob2=(expv2*(a)**(lambda1-1))/inclvalue
    prob3=(expv3*(b)**(lambda2-1))/inclvalue
    prob4=(expv4*(b)**(lambda2-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)


def LL_N_22b(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1,gamma2=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = np.exp(gamma2)/(np.exp(gamma2) + 1)


    #Representative utilities
```

```
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda1
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda2
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda1
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda2


    #Exp Representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a = expv1 + expv3
    b = expv2 + expv4

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(a)**(lambda1-1))/inclvalue
    prob2=(expv2*(b)**(lambda2-1))/inclvalue
    prob3=(expv3*(a)**(lambda1-1))/inclvalue
    prob4=(expv4*(b)**(lambda2-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)

def LL_N_22c(beta):
    #Unpacking
    alpha1, alpha2, alpha3, betap, betaf,gamma1,gamma2=beta

    # Nested parameters
    lambda1 = np.exp(gamma1)/(np.exp(gamma1) + 1)
    lambda2 = np.exp(gamma2)/(np.exp(gamma2) + 1)
```

```
    #Representative utilities
    v1=(alpha1+betap*df['price1']+betaf*df['feat1'])/lambda1
    v2=(alpha2+betap*df['price2']+betaf*df['feat2'])/lambda2
    v3=(alpha3+betap*df['price3']+betaf*df['feat3'])/lambda2
    v4=(betap*df['price4']+betaf*df['feat4'])/lambda1


    #Exp Representative utilities
    expv1=v1.apply(np.exp)
    expv2=v2.apply(np.exp)
    expv3=v3.apply(np.exp)
    expv4=v4.apply(np.exp)

    # Nested

    a = expv1 + expv4
    b = expv2 + expv3

    #Inclusive value (denominador)
    inclvalue=(a)**(lambda1)+(b)**(lambda2)

    #Choice probabilities
    prob1=(expv1*(a)**(lambda1-1))/inclvalue
    prob2=(expv2*(b)**(lambda2-1))/inclvalue
    prob3=(expv3*(b)**(lambda2-1))/inclvalue
    prob4=(expv4*(a)**(lambda1-1))/inclvalue

    #Lilkelihood per observation
    l=df['brand1']*np.log(prob1)+df['brand2']*np.log(prob2)+df['brand3']*np.log(prob3)+df['brand4']*np.log(prob4)

    #Sumar likelihood sobre todas las observaciones
    return -np.sum(l)


# Optimize with diferents methods

## Save values

def optimizar_logit_condicional_save (point0, metodo):
    conca = pd.DataFrame()
```

```
## Añadir beta óptimo obtenido

a = minimize(LL,point0,method=metodo)

a_values = a["x"]

conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(a_values))],axis=0)

conca.rename({0: 'alph1',1: 'alph2',2: 'alph3',3: 'bp',4: 'bf'}, axis=1, inplace=True)
conca.reset_index(drop=True, inplace=True)

## Añadir valor optimo obtenido
val_a = -LL(a_values)

val_opt = pd.DataFrame([val_a])
val_opt.rename({0: 'valor_opt'}, axis=1, inplace=True)

conca = pd.concat([conca,val_opt],axis=1)

## Añadir punto inicial empleado

point_inicial = pd.DataFrame([point0])
point_inicial.rename({0: 'alph1_0',1: 'alph2_0',2: 'alph3_0',3: 'bp_0',4: 'bf_0'}, axis=1, inplace=True)
conca = pd.concat([conca,point_inicial],axis=1)

## Añadir nombre del metodo empleado

name_method = pd.DataFrame([metodo])
name_method.rename({0: 'Metodo'}, axis=1, inplace=True)
conca = pd.concat([conca,name_method],axis=1)

## Añadir AIC (-2 * mv_betas) + (2*5)

AIC = pd.DataFrame([(2*5) -2*val_a])
AIC.rename({0: 'AIC'}, axis=1, inplace=True)
conca = pd.concat([conca,AIC],axis=1)

## Añadir Indice de razón de verosimilitud
ceros = [0,0,0,0,0]
valcero_a = -LL(ceros)
```

```
razon_vero = pd.DataFrame([1-val_a/valcero_a])
razon_vero.rename({0: 'Indice Verosimilitud'}, axis=1, inplace=True)
conca = pd.concat([conca,razon_vero],axis=1)

## Añadir errores

conca_SE = pd.DataFrame()

####Hessiano
HLL = nd.Hessian(LL)(a_values)
####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse = np.linalg.inv(HLL)
####Extraer varianzas y calcular SE
SE = pd.DataFrame(np.sqrt(np.diag(HLL_inverse)))
conca_SE = pd.concat([conca_SE,SE],axis=1)

conca_SE = pd.DataFrame.transpose(conca_SE)

conca_SE.rename({0: 'alph1_SE',1: 'alph2_SE',2: 'alph3_SE',3: 'bp_SE',4: 'bf_SE'}, axis=1, inplace=True)
conca_SE.reset_index(drop=True, inplace=True)

conca = pd.concat([conca,conca_SE],axis=1)

# Añadir nidos

nid = pd.DataFrame()
n1 = [0]
n2 = [0]
n3 = [0]
n4 = [0]

nid['nest1'] = n1
nid['nest2'] = n2
nid['nest3'] = n3
nid['nest4'] = n4
nid["nest_est"] = "Logit condicional"

nid.reset_index(drop=True, inplace=True)
conca = pd.concat([conca,nid],axis=1)

return conca
```

```
def optimizar_logit_31_save (point0, metodo):

    conca = pd.DataFrame()

    ## Añadir beta óptimo obtenido

    a = minimize(LL_N_31a,point0,method=metodo)
    b = minimize(LL_N_31b,point0,method=metodo)
    c = minimize(LL_N_31c,point0,method=metodo)
    d = minimize(LL_N_31d,point0,method=metodo)

    a_values,b_values,c_values,d_values = a["x"],b["x"],c["x"],d["x"]


    conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(a_values))],axis=0)
    conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(b_values))],axis=0)
    conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(c_values))],axis=0)
    conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(d_values))],axis=0)

    conca.rename({0: 'alph1',1: 'alph2',2: 'alph3',3: 'bp',4: 'bf',5: "gamma1"}, axis=1, inplace=True)
    conca.reset_index(drop=True, inplace=True)

    ## Añadir valor optimo obtenido
    val_a, val_b, val_c, val_d = -LL_N_31a(a_values), -LL_N_31b(b_values), -LL_N_31c(c_values), -LL_N_31d(d_values)

    val_opt = pd.DataFrame([val_a, val_b, val_c, val_d])
    val_opt.rename({0: 'valor_opt'}, axis=1, inplace=True)

    conca = pd.concat([conca,val_opt],axis=1)

    ## Añadir punto inicial empleado

    point_inicial = pd.DataFrame([point0, point0, point0, point0])
    point_inicial.rename({0: 'alph1_0',1: 'alph2_0',2: 'alph3_0',3: 'bp_0',4: 'bf_0',5: "gamma1_0"}, axis=1, inplace=True)
    conca = pd.concat([conca,point_inicial],axis=1)

    ## Añadir nombre del metodo empleado

    name_method = pd.DataFrame([metodo,metodo, metodo, metodo])
```

```
name_method.rename({0: 'Metodo'}, axis=1, inplace=True)
conca = pd.concat([conca,name_method],axis=1)

## Añadir AIC (-2 * mv_betas) + (2*5)

AIC = pd.DataFrame([(2*6) -2*val_a, (2*6) -2*val_b, (2*6) -2*val_c, (2*6) -2*val_d])
AIC.rename({0: 'AIC'}, axis=1, inplace=True)
conca = pd.concat([conca,AIC],axis=1)

## Añadir Indice de razón de verosimilitud
ceros = [0,0,0,0,0,0]
valcero_a, valcero_b, valcero_c, valcero_d = -LL_N_31a(ceros), -LL_N_31b(ceros), -LL_N_31c(ceros), -LL_N_31d(ceros)

razon_vero = pd.DataFrame([1-val_a/valcero_a, 1-val_b/valcero_b, 1-val_c/valcero_c, 1-val_d/valcero_d])
razon_vero.rename({0: 'Indice Verosimilitud'}, axis=1, inplace=True)
conca = pd.concat([conca,razon_vero],axis=1)

## Añadir errores

conca_SE = pd.DataFrame()

### 31a
####Hessiano
HLL_31a = nd.Hessian(LL_N_31a)(a_values)
####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_31a = np.linalg.inv(HLL_31a)
####Extraer varianzas y calcular SE
SE_31a = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_31a)))
conca_SE = pd.concat([conca_SE,SE_31a],axis=1)

### 31b
####Hessiano
HLL_31b = nd.Hessian(LL_N_31b)(b_values)

####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_31b = np.linalg.inv(HLL_31b)
####Extraer varianzas y calcular SE
SE_31b = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_31b)))
conca_SE = pd.concat([conca_SE,SE_31b],axis=1)

### 31c
```

```
####Hessiano
HLL_31c = nd.Hessian(LL_N_31c)(c_values)
####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_31c = np.linalg.inv(HLL_31c)
####Extraer varianzas y calcular SE
SE_31c = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_31c)))
conca_SE = pd.concat([conca_SE,SE_31c],axis=1)

### 31d
####Hessiano
HLL_31d = nd.Hessian(LL_N_31d)(d_values)
####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_31d = np.linalg.inv(HLL_31d)
####Extraer varianzas y calcular SE
SE_31d = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_31d)))
conca_SE = pd.concat([conca_SE,SE_31d],axis=1)

conca_SE = pd.DataFrame.transpose(conca_SE)

conca_SE.rename({0: 'alph1_SE',1: 'alph2_SE',2: 'alph3_SE',3: 'bp_SE',4: 'bf_SE',5: "gamma1_SE"}, axis=1, inplace=True)
conca_SE.reset_index(drop=True, inplace=True)

conca = pd.concat([conca,conca_SE],axis=1)


## Añadir errores estándar de lambdas con método delta

### 31a
estim31a = (a["x"])
def f_ee_31a(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_31a)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l31a = f_ee_31a(estim31a[5])
ee_l31a = ee_l31a[0]
```

```
### 31b
estim31b = (b["x"])
def f_ee_31b(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_31b)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l31b = f_ee_31b(estim31b[5])
ee_l31b = ee_l31b[0]

### 31c
estim31c = (c["x"])
def f_ee_31c(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_31c)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l31c = f_ee_31c(estim31c[5])
ee_l31c = ee_l31c[0]

### 31d
estim31d = (d["x"])
def f_ee_31d(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_31d)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l31d = f_ee_31d(estim31d[5])
```

```
    ee_l31d = ee_l31d[0]


    val_ee_lambda = pd.DataFrame([ee_l31a, ee_l31b, ee_l31c, ee_l31d])
    val_ee_lambda.rename({0: 'ee_lambda1'}, axis=1, inplace=True)

    conca = pd.concat([conca,val_ee_lambda],axis=1)


    ## Añadir nidos

    nid = pd.DataFrame()
    n1 = [1,1,1,2]
    n2 = [1,1,2,1]
    n3 = [1,2,1,1]
    n4 = [2,1,1,1]

    nid['nest1'] = n1
    nid['nest2'] = n2
    nid['nest3'] = n3
    nid['nest4'] = n4
    nid["nest_est"] = "(3,1)"

    ## Añadir lambdas
    nid["lambda1"] = np.exp(conca["gamma1"])/(np.exp(conca["gamma1"]) + 1)
    nid["lambda2"] = 1

    nid.reset_index(drop=True, inplace=True)
    conca = pd.concat([conca,nid],axis=1)

    return conca


def optimizar_logit_22_save (point0, metodo):

    conca = pd.DataFrame()

    ## Añadir beta óptimo obtenido

    a = minimize(LL_N_22a,point0,method=metodo)
    b = minimize(LL_N_22b,point0,method=metodo)
```

```
c = minimize(LL_N_22c,point0,method=metodo)

a_values,b_values,c_values = a["x"],b["x"],c["x"]


conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(a_values))],axis=0)
conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(b_values))],axis=0)
conca = pd.concat([conca,pd.DataFrame.transpose(pd.DataFrame(c_values))],axis=0)

conca.rename({0: 'alph1',1: 'alph2',2: 'alph3',3: 'bp',4: 'bf',5: "gamma1",6: "gamma2"}, axis=1, inplace=True)
conca.reset_index(drop=True, inplace=True)

## Añadir valor optimo obtenido
val_a, val_b, val_c = -LL_N_22a(a_values), -LL_N_22b(b_values), -LL_N_22c(c_values)

val_opt = pd.DataFrame([val_a, val_b, val_c])
val_opt.rename({0: 'valor_opt'}, axis=1, inplace=True)

conca = pd.concat([conca,val_opt],axis=1)

## Añadir punto inicial empleado

point_inicial = pd.DataFrame([point0, point0, point0])
point_inicial.rename({0: 'alph1_0',1: 'alph2_0',2: 'alph3_0',3: 'bp_0',4: 'bf_0',5: "gamma1_0",6: "gamma2_0"}, axis=1, inplace=
conca = pd.concat([conca,point_inicial],axis=1)

## Añadir nombre del metodo empleado

name_method = pd.DataFrame([metodo,metodo, metodo])
name_method.rename({0: 'Metodo'}, axis=1, inplace=True)
conca = pd.concat([conca,name_method],axis=1)

## Añadir AIC (-2 * mv_betas) + (2*5)

AIC = pd.DataFrame([(2*7) -2*val_a, (2*7) -2*val_b, (2*7) -2*val_c])
AIC.rename({0: 'AIC'}, axis=1, inplace=True)
conca = pd.concat([conca,AIC],axis=1)

## Añadir Indice de razón de verosimilitud
ceros = [0,0,0,0,0,0,0]
valcero_a, valcero_b, valcero_c= -LL_N_22a(ceros), -LL_N_22b(ceros), -LL_N_22c(ceros)
```

```
razon_vero = pd.DataFrame([1-val_a/valcero_a, 1-val_b/valcero_b, 1-val_c/valcero_c])
razon_vero.rename({0: 'Indice Verosimilitud'}, axis=1, inplace=True)
conca = pd.concat([conca,razon_vero],axis=1)

## Añadir errores

conca_SE = pd.DataFrame()

### 22a
####Hessiano
HLL_22a = nd.Hessian(LL_N_22a)(a_values)
####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_22a = np.linalg.inv(HLL_22a)
####Extraer varianzas y calcular SE
SE_22a = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_22a)))
conca_SE = pd.concat([conca_SE,SE_22a],axis=1)

### 22b
####Hessiano
HLL_22b = nd.Hessian(LL_N_22b)(b_values)

####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_22b = np.linalg.inv(HLL_22b)
####Extraer varianzas y calcular SE
SE_22b = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_22b)))
conca_SE = pd.concat([conca_SE,SE_22b],axis=1)

### 22c
####Hessiano
HLL_22c = nd.Hessian(LL_N_22c)(c_values)
####Matriz de informacion ( no tomo -HLL porque estoy considerando que el modelo se multiplica por -1)
HLL_inverse_22c = np.linalg.inv(HLL_22c)
####Extraer varianzas y calcular SE
SE_22c = pd.DataFrame(np.sqrt(np.diag(HLL_inverse_22c)))
conca_SE = pd.concat([conca_SE,SE_22c],axis=1)

conca_SE = pd.DataFrame.transpose(conca_SE)

conca_SE.rename({0: 'alph1_SE',1: 'alph2_SE',2: 'alph3_SE',3: 'bp_SE',4: 'bf_SE',5: "gamma1_SE",6: "gamma2_SE"}, axis=1, inplac
conca_SE.reset_index(drop=True, inplace=True)
```

```
conca = pd.concat([conca,conca_SE],axis=1)

## Añadir errores estándar de lambdas con método delta

### 22a
estim22a = (a["x"])
def f_ee_22a(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada], [0]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_22a)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l1_22a = f_ee_22a(estim22a[5])
ee_l1_22a = ee_l1_22a[0]

def f_ee_22a2(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [0] , [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_22a)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l2_22a = f_ee_22a2(estim22a[6])
ee_l2_22a = ee_l2_22a[0]

### 22b
estim22b = (b["x"])
def f_ee_22b(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada], [0]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_22b)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee
```

```
ee_l1_22b = f_ee_22b(estim22b[5])
ee_l1_22b = ee_l1_22b[0]

def f_ee_22b2(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [0] , [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_22b)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l2_22b = f_ee_22b2(estim22b[6])
ee_l2_22b = ee_l2_22b[0]

### 22c
estim22c = (c["x"])
def f_ee_22c(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [derivada], [0]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_22c)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l1_22c = f_ee_22c(estim22c[5])
ee_l1_22c = ee_l1_22c[0]

def f_ee_22c2(g):
    derivada = np.exp(g)/((1+ np.exp(g))**2)
    mat = [[0], [0], [0], [0], [0], [0] , [derivada]]
    mat_t = np.transpose(mat)
    res = np.dot(mat_t, HLL_inverse_22c)
    res2 = np.dot(res, mat)
    ee = np.sqrt(res2)
    return ee

ee_l2_22c = f_ee_22c2(estim22c[6])
ee_l2_22c = ee_l2_22c[0]
```

```
    val_ee_lambda1 = pd.DataFrame([ee_l1_22a, ee_l1_22b, ee_l1_22c])
    val_ee_lambda1.rename({0: 'ee_lambda1'}, axis=1, inplace=True)

    conca = pd.concat([conca,val_ee_lambda1],axis=1)

    val_ee_lambda2 = pd.DataFrame([ee_l2_22a, ee_l2_22b, ee_l2_22c])
    val_ee_lambda2.rename({0: 'ee_lambda2'}, axis=1, inplace=True)

    conca = pd.concat([conca,val_ee_lambda2],axis=1)

    # Añadir nidos

    nid = pd.DataFrame()
    n1 = [1,1,1]
    n2 = [1,2,2]
    n3 = [2,1,2]
    n4 = [2,2,1]


    nid['nest1'] = n1
    nid['nest2'] = n2
    nid['nest3'] = n3
    nid['nest4'] = n4
    nid["nest_est"] = "(2,2)"

    ## Añadir lambdas
    nid["lambda1"] = np.exp(conca["gamma1"])/(np.exp(conca["gamma1"]) + 1)
    nid["lambda2"] = np.exp(conca["gamma2"])/(np.exp(conca["gamma2"]) + 1)

    nid.reset_index(drop=True, inplace=True)
    conca = pd.concat([conca,nid],axis=1)

    return conca

# Creacion de la base de datos

##Correr los modelos con distintos metodos y puntos iniciales

### Logit condicional
```

```
base_giant_1 = pd.DataFrame()

try:
    save_val = optimizar_logit_condicional_save([0,0,0,0,0],'trust-constr')
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([1.38,0.83,-1.65,-26.58,0.37],'trust-constr')
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([0,0,0,0,0],"L-BFGS-B")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([1.38,0.83,-1.65,-26.58,0.37],"L-BFGS-B")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([0,0,0,0,0],"Nelder-Mead")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([1.38,0.83,-1.65,-26.58,0.37],"Nelder-Mead")
```

```
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([0,0,0,0,0],"Powell")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([1.38,0.83,-1.65,-26.58,0.37],"Powell")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([0,0,0,0,0],"TNC")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_condicional_save([1.38,0.83,-1.65,-26.58,0.37],"TNC")
    base_giant_1 = pd.concat([base_giant_1,save_val],axis=0)
    base_giant_1.reset_index(drop=True, inplace=True)
except:
    pass


### Nested logit

base_giant_31 = pd.DataFrame()
try:
    save_val = optimizar_logit_31_save([0,0,0,0,0,0],'trust-constr')
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
```

```
except:
    pass

try:
    save_val = optimizar_logit_31_save([1.38,0.83,-1.65,-26.58,0.37,0.59],'trust-constr')
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_31_save([0,0,0,0,0,0],"L-BFGS-B")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_31_save([1.38,0.83,-1.65,-26.58,0.37,0.59],"L-BFGS-B")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_31_save([0,0,0,0,0,0],"Nelder-Mead")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_31_save([1.38,0.83,-1.65,-26.58,0.37,0.59],"Nelder-Mead")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_31_save([0,0,0,0,0,0],"Powell")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
```

```
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass


try:
    save_val = optimizar_logit_31_save([1.38,0.83,-1.65,-26.58,0.37,0.59],"Powell")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_31_save([0,0,0,0,0,0],"TNC")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass


try:
    save_val = optimizar_logit_31_save([1.38,0.83,-1.65,-26.58,0.37,0.59],"TNC")
    base_giant_31 = pd.concat([base_giant_31,save_val],axis=0)
    base_giant_31.reset_index(drop=True, inplace=True)
except:
    pass




base_giant_22 = pd.DataFrame()

try:
    save_val = optimizar_logit_22_save([0,0,0,0,0,0,0],'trust-constr')
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_22_save([1.38,0.83,-1.65,-26.58,0.37,0.59,1],'trust-constr')
```

```
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_22_save([0,0,0,0,0,0,0],"L-BFGS-B")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass


try:
    save_val = optimizar_logit_22_save([1.38,0.83,-1.65,-26.58,0.37,0.59,1],"L-BFGS-B")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_22_save([0,0,0,0,0,0,0],"Nelder-Mead")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_22_save([1.38,0.83,-1.65,-26.58,0.37,0.59,1],"Nelder-Mead")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_22_save([0,0,0,0,0,0,0],"Powell")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass
```

```
try:
    save_val = optimizar_logit_22_save([1.38,0.83,-1.65,-26.58,0.37,0.59,1],"Powell")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

try:
    save_val = optimizar_logit_22_save([0,0,0,0,0,0,0],"TNC")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass


try:
    save_val = optimizar_logit_22_save([1.38,0.83,-1.65,-26.58,0.37,0.59,1],"TNC")
    base_giant_22 = pd.concat([base_giant_22,save_val],axis=0)
    base_giant_22.reset_index(drop=True, inplace=True)
except:
    pass

##Eliminar el archivo csv en caso de que exista

try:
    os.remove('logit_condicional.csv')
except:
    pass

try:
    os.remove('Nested_logit_31.csv')
except:
    pass

try:
    os.remove('Nested_logit_22.csv')
except:
    pass

try:
```

```
    os.remove('Nested_logit_all.csv')
except:
    pass

#Guardar los modelos en archivo csv

base_giant_22.to_csv('Nested_logit_22.csv', mode='a', index=False, header=True)
base_giant_22 = base_giant_22.dropna()

base_giant_31.to_csv('Nested_logit_31.csv', mode='a', index=False, header=True)
base_giant_31 = base_giant_31.dropna()

base_giant_1.to_csv('logit_condicional.csv', mode='a', index=False, header=True)
base_giant_1 = base_giant_1.dropna()


base_giant_nested = pd.concat([base_giant_31,base_giant_22, base_giant_1],axis=0)


base_giant_nested = base_giant_nested.reindex(columns=\
                                              ['alph1','alph2','alph3',\
                                               'bp','bf','gamma1','gamma2',\
                                               "lambda1","lambda2",\
                                               'valor_opt','Metodo',\
                                               'AIC','Indice Verosimilitud',\
                                               "nest_est",\
                                               'nest1','nest2','nest3',\
                                               'nest4','alph1_0','alph2_0',\
                                               'alph3_0','bp_0','bf_0',\
                                               'gamma1_0','gamma2_0','alph1_SE',\
                                               'alph2_SE','alph3_SE','bp_SE',\
                                               'bf_SE','gamma1_SE','gamma2_SE',\
                                               'ee_lambda1','ee_lambda2'])


base_giant_nested.to_csv('Nested_logit_all.csv', mode='a', index=False, header=True)

#Combinación de nidos con el menor AIC
AIC_low = base_giant_nested[base_giant_nested.AIC == base_giant_nested.AIC.min()]
print(AIC_low)
```