TRABAJO PRACTICO
REDES NEURONALES

UNIVERSIDAD NACIONAL
DE SAN MARTIN

ECyT

MATEMATICA III

JULIAN BARBERIS

Análisis de la Base de Datos

Selección de la Base de Datos

Elegí una base de datos de UC Irvine Machine Learning Repository, https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset, con la cual podremos analizar si un hongo que encontramos en la naturaleza es comestible o no.

```
import pandas as pd
import numpy as np

data = pd.read_csv('mushrooms.csv', delimiter=';')
df = pd.DataFrame(data)

df
```

```
      class  cap-diameter cap-shape cap-surface cap-color  \
0         p         15.26         x           g         o
1         p         16.60         x           g         o
2         p         14.07         x           g         o
3         p         14.17         f           h         e
4         p         14.64         x           h         o
...     ...           ...       ...         ...       ...
61064     p          1.18         s           s         y
61065     p          1.27         f           s         y
61066     p          1.27         s           s         y
61067     p          1.24         f           s         y
61068     p          1.17         s           s         y

      does-bruise-or-bleed gill-attachment gill-spacing gill-color  \
0                        f               e          NaN          w
1                        f               e          NaN          w
2                        f               e          NaN          w
3                        f               e          NaN          w
4                        f               e          NaN          w
...                    ...             ...          ...        ...
61064                    f               f            f          f
61065                    f               f            f          f
61066                    f               f            f          f
61067                    f               f            f          f
61068                    f               f            f          f

       stem-height  ...  stem-root stem-surface stem-color veil-type  \
0            16.95  ...          s            y          w         u
1            17.99  ...          s            y          w         u
2            17.80  ...          s            y          w         u
3            15.77  ...          s            y          w         u
4            16.53  ...          s            y          w         u
...            ...  ...        ...          ...        ...       ...
61064         3.93  ...        NaN          NaN          y       NaN
61065         3.18  ...        NaN          NaN          y       NaN
61066         3.86  ...        NaN          NaN          y       NaN
61067         3.56  ...        NaN          NaN          y       NaN
61068         3.25  ...        NaN          NaN          y       NaN


      veil-color has-ring ring-type spore-print-color hábitat season
```

```
0              w        t         g                NaN       d       w
1              w        t         g                NaN       d       u
2              w        t         g                NaN       d       w
3              w        t         p                NaN       d       w
4              w        t         p                NaN       d       w
...          ...      ...       ...                ...     ...     ...
61064        NaN        f         f                NaN       d       a
61065        NaN        f         f                NaN       d       a
61066        NaN        f         f                NaN       d       u
61067        NaN        f         f                NaN       d       u
61068        NaN        f         f                NaN       d       u

[61069 rows x 21 columns]
```

Describiendo cada columna del conjunto de datos:

1. class = si el hongo es comestible o no, variable categórica
2. cap-diameter = diámetro de la tapa/sombrero, variable continua
3. cap-shape = forma de la tapa/sombrero, variable categórica
4. cap-surface = superficie de la tapa/sombrero, variable categórica
5. cap-color = color de la tapa/sombrero, variable categórica
6. does-bruise-or-bleed = ¿tiene moretones o sangra?, variable categórica
7. gill-attachment = fijación de las láminas que se encuentran debajo del sombrero, variable categórica
8. gill-spacing = espacio entre las branquias, variable categórica
9. gill-color = color de las láminas, variable categórica
10. stem-height = altura del tallo, variable continua
11. stem-width = ancho del tallo, variable continua
12. stem-root = forma de transición tallo-raíz, variable categórica
13. stem-surface = superficie del tallo, variable categórica
14. stem-color = color del tallo, variable categórica
15. veil-type = tipo de velo, variable categórica
16. veil-color = color del velo, variable categórica
17. has-ring = ¿tiene anillo?, variable categórica
18. ring-type = tipo de anillo, variable categórica
19. spore-print-color = impresión de esporas en color, variable categórica
20. hábitat = hábitat en el que se encontró el hongo, variable categórica
21. season = estación del año en la que se encontró el hongo, variable categórica

Normalizando el conjunto de datos

Por una parte, decidí transformar las variables representadas con caracteres a variables numéricas, debido a que de esta forma es más fácil analizarlas. Por otra parte, me di cuenta que lo ideal era tener todos los variables dentro de una misma escala de valores, primero entre 0 y 12 y luego entre 0 y 1.

```
df_class = {                #COMESTIBLE O NO?
    'p' : 0,    #venenoso
    'e' : 12    #comestible
}

df_cap_shape = {            #FORMA SOMBRERO
    'b': 0,     #campana
    'o': 2,     #otros
    'c': 4,     #conica
    's': 6,     #hundida
    'x': 8,     #convexa
    'f': 10,    #plano
    'p': 12     #esferica
}

df_cap_surface = {          #SUPERFICIE SOMBRERO
    'i': 0,     #fibrosa
    'g': 1.3,   #surcosa
    'y': 2.6,   #escamosa
    's': 3.9,   #liso
    'h': 5.2,   #brillante
    'l': 6.5,   #como cuero
    'k': 7.8,   #sedosa
    't': 9.1,   #pegajosa
    'w': 10.4,  #arrugada
    'e': 11.7,  #carnosa
    '' : 12     #deconocido
}

df_cap_color = {          #COLOR SOMBRERO
    'r': 0,     #verde
    'e': 1.091, #rojo
    'p': 2.182, #rosa
    'o': 3.273, #naranja
    'k': 5.455, #negro
    'y': 6.546, #amarillo
    'w': 7.637, #blanco
    'u': 4.364, #violeta
    'n': 8.728, #marron
    'g': 9.819, #gris
    'l': 10.91, #azul
    'b': 12     #pulido
}

df_does_bruise_bleed = {    #TIENE MORETONES O SANGRA?
    'f': 0,     #falso
    '' : 6,     #desconocido
    't': 12     #verdadero
}
```

```
df_gill_attachment = {      #FIJACION DE LAS LAMINAS
    '' : 0,     #desconocido
    'a': 1.71,  #adherida
    'd': 3.43,  #decurrente
    'f': 5.14,  #ninguna
    'x': 6.86,  #anexada
    's': 8.57,  #ondeada
    'e': 10.29, #libre
    'p': 12     #porosa
}

df_gill_spacing = {         #ESPACIADO DE LAS LAMINAS
    'f': 0,     #ninguna
    'c': 4,     #cerca
    '' : 8,     #desconocido
    'd': 12     #distante
}

df_gill_color = {           #COLOR DE LAS LAMINAS
    'e': 0,     #rojo
    'n': 1.091, #marron
    'r': 2.182, #verde
    'k': 3.273, #negro
    'f': 4.364, #ninguno
    'y': 5.455, #amarillo
    'p': 6.546, #rosa
    'o': 7.637, #naranja
    'u': 8.728, #violeta
    'g': 9.819, #gris
    'w': 10.91, #blanco
    'b': 12     #pulido
}

df_stem_root = {            #TRANSICION TALLO-RAIZ
    'b': 0,     #bulboso
    's': 2,     #hinchado
    'c': 4,     #hernia
    'u': 6,     #copa
    'e': 8,     #igual
    'z': 10,    #rizomorfo
    'r': 12     #arraigado
}

df_stem_surface = {         #SUPERFICIE DE LAS LAMINAS
    'g': 0,     #surcosa
    'f': 1.5,   #ninguna
    'h': 3,     #brillante
    'y': 4.5,   #escamosa
    't': 6,     #pegajosa
    'k': 7.5,   #sedosa
    'i': 9,     #fibrosa
    '' : 10.5,  #desconocido
    's': 12     #liso
}

df_stem_color = {           #COLOR DE LAS LAMIANS
    'p': 0,     #rosa
    'r': 1,     #verde
```

```
    'k': 2,     #negro
    'e': 3,     #rojo
    'y': 4,     #amarillo
    'u': 5,     #violeta
    'o': 6,     #naranja
    'f': 7,     #ninguno
    'n': 8,     #marron
    'l': 9,     #azul
    'w': 10,    #blanco
    'g': 11,    #gris
    'b': 12     #pulido
}

df_veil_type = {            #TIPO DE VELO
    'u': 0,     #universal
    '' : 6,     #desconocido
    'p': 12     #parcial
}

df_veil_color = {           #COLOR DE VELO
    'k': 0,     #negro
    'n': 1,     #marron
    'r': 2,     #verde
    'l': 3,     #azul
    'u': 4,     #violeta
    'p': 5,     #rosa
    'e': 6,     #rojo
    'o': 7,     #naranja
    'y': 8,     #amarillo
    'b': 9,     #pulido
    'g': 10,    #gris
    'w': 11,    #blanco
    'f': 12     #ninguno
}

df_has_ring = {             #TIENE ANILLO?
    't': 0,     #verdadero
    'f': 12     #falso
}

df_ring_type = {            #TIPO DE ANILLO
    'z': 0,     #faja
    'p': 1.091, #colgante
    'e': 2.182, #evanescente
    'f': 3.273, #ninguno
    'r': 4.364, #resplandeciente
    'c': 5.455, #lleno de telarañas
    's': 6.546, #enfundado
    'y': 7.637, #escamoso
    'g': 8.728, #estriado
    '' : 9.819, #desconocido
    'l': 10.91, #largo
    'm': 12     #movible
}

df_spore_print_color = {    #IMPRESION DE ESPORAS EN COLOR
    'n': 0,     #marron
    'u': 1.71,  #violeta
    'r': 3.43,  #verde
```

```
    'k': 5.14,  #negro
    'p': 6.86,  #rosa
    '' : 8.57,  #desconocido
    'w': 10.29, #blanco
    'g': 12     #gris
}

df_habitat= {               #HABITAT
    'p': 0,     #camino
    'g': 1.71,  #pasto
    'h': 3.43,  #brezales
    'd': 5.14,  #madera
    'm': 6.86,  #prado
    'l': 8.57,  #hojas
    'u': 10.29, #urbano
    'w': 12     #residuos
}

df_season= {                #ESTACION
    'a': 0,     #otoño
    'u': 4,     #verano
    'w': 8,     #invierno
    's': 12     #primavera
}

df_cap_diameter = {         #DIAMETRO SOMBRERO
    (0, 5.195):         0,
    (5.196, 10.390):    1,
    (10.391, 15.585):   2,
    (15.586, 20.780):   3,
    (20.781, 25.975):   4,
    (25.976, 31.170):   5,
    (31.171, 36.395):   6,
    (36.396, 41.560):   7,
    (41.561, 46.755):   8,
    (46.756, 51.950):   10,
    (51.951, 57.145):   11,
    (57.146, 62.340):   12
}

df_stem_height= {           #ALTURA DE LAS LAMINAS
    (0, 2.827):         0,
    (2.828, 5.653):     1,
    (5.654, 8.480):     2,
    (8.481, 11.306):    3,
    (11.307,14.133):    4,
    (14.134, 16.959):   5,
    (16.960, 19.786):   6,
    (19.787, 22.612):   7,
    (22.613, 25.439):   8,
    (25.440, 28.265):   10,
    (28.266, 31.092):   11,
    (31.093, 33.920):   12
}

df_stem_width= {            #ANCHO DE LAS LAMINAS
    (0, 8.6592):          0,
    (8.6593, 17.3183):    1,
    (17.3184, 25.9775):   2,
```

```
    (25.9776, 34.6366):  3,
    (34.6367, 43.2958):  4,
    (43.2959, 51.9549):  5,
    (51.9550, 60.6141):  6,
    (60.6142, 69.2732):  7,
    (69.2733, 77.9324):  8,
    (77.9325, 86.5915):  10,
    (86.5916, 95.2507):  11,
    (95.2508, 103.9100): 12
}
```

Reemplazo los valores e imprimo el DataFrame actualizado

```
reemplazo = {
    'class': df_class,
    'cap-shape':  df_cap_shape,
    'cap-surface': df_cap_surface,
    'cap-color': df_cap_color,
    'does-bruise-or-bleed': df_does_bruise_bleed,
    'gill-attachment': df_gill_attachment,
    'gill-spacing': df_gill_spacing,
    'gill-color': df_gill_color,
    'stem-root': df_stem_root,
    'stem-surface': df_stem_surface,
    'stem-color':  df_stem_color,
    'veil-type':  df_veil_type,
    'veil-color': df_veil_color,
    'has-ring':  df_has_ring,
    'ring-type': df_ring_type,
    'spore-print-color': df_spore_print_color,
    'hábitat':  df_habitat,
    'season': df_season
}

def reemplazo_variables(valores, rango):
    for rango_dic, grupo in rango.items():
        if rango_dic[0] <= valores <= rango_dic[1]:
            return grupo
    return -1

df['cap-diameter'] = df['cap-diameter'].apply(lambda x: reemplazo_variables(x, df_cap_
diameter))
df['stem-height'] = df['stem-height'].apply(lambda x: reemplazo_variables(x, df_stem_h
eight))
df['stem-width'] = df['stem-width'].apply(lambda x: reemplazo_variables(x, df_stem_wid
th))

for col, rep in reemplazo.items():
    if col in df.columns:
        df[col] = df[col].replace(rep)

df
```

```
  df[col] = df[col].replace(rep)
```

	class	cap-diameter	cap-shape	cap-surface	cap-color	\
0	0	2	8	1.3	3.273	
1	0	3	8	1.3	3.273	
2	0	2	8	1.3	3.273	

```
3          0             2         10         5.2      1.091
4          0             2          8         5.2      3.273
...      ...           ...        ...         ...        ...
61064      0             0          6         3.9      6.546
61065      0             0         10         3.9      6.546
61066      0             0          6         3.9      6.546
61067      0             0         10         3.9      6.546
61068      0             0          6         3.9      6.546

       does-bruise-or-bleed  gill-attachment  gill-spacing  gill-color  \
0                         0            10.29           NaN      10.910
1                         0            10.29           NaN      10.910
2                         0            10.29           NaN      10.910
3                         0            10.29           NaN      10.910
4                         0            10.29           NaN      10.910
...                     ...              ...           ...         ...
61064                     0             5.14           0.0       4.364
61065                     0             5.14           0.0       4.364
61066                     0             5.14           0.0       4.364
61067                     0             5.14           0.0       4.364
61068                     0             5.14           0.0       4.364

       stem-height  ...  stem-root stem-surface  stem-color  veil-type  \
0                5  ...          2          4.5          10        0.0
1                6  ...          2          4.5          10        0.0
2                6  ...          2          4.5          10        0.0
3                5  ...          2          4.5          10        0.0
4                5  ...          2          4.5          10        0.0
...            ...  ...        ...          ...         ...        ...
61064            1  ...        NaN          NaN           4        NaN
61065            1  ...        NaN          NaN           4        NaN
61066            1  ...        NaN          NaN           4        NaN
61067            1  ...        NaN          NaN           4        NaN
61068            1  ...        NaN          NaN           4        NaN

       veil-color  has-ring  ring-type  spore-print-color  hábitat  season
0            11.0         0      8.728                NaN     5.14       8
1            11.0         0      8.728                NaN     5.14       4
2            11.0         0      8.728                NaN     5.14       8
3            11.0         0      1.091                NaN     5.14       8
4            11.0         0      1.091                NaN     5.14       8
...           ...       ...        ...                ...      ...     ...
61064         NaN        12      3.273                NaN     5.14       0
61065         NaN        12      3.273                NaN     5.14       0
61066         NaN        12      3.273                NaN     5.14       4
61067         NaN        12      3.273                NaN     5.14       4
61068         NaN        12      3.273                NaN     5.14       4

[61069 rows x 21 columns]
```

Reemplazo los valores de las columnas con NaN

```
values = {"gill-attachment": 0,"veil-type": 7, "stem-surface": 7, "cap-surface": 0, "d
oes-bruise-or-bleed": 5, "gill-spacing": 6, "ring-type":9,"spore-print-color":5}
df =df.fillna(value=values)

df = df.drop(columns=['cap-surface', 'stem-root']) #Elimino las columnas porque no me
deja imprimir
df = df.divide(12)
```

```
df
```

```
       class  cap-diameter  cap-shape  cap-color  does-bruise-or-bleed  \
0        0.0      0.166667   0.666667   0.272750                   0.0
1        0.0      0.250000   0.666667   0.272750                   0.0
2        0.0      0.166667   0.666667   0.272750                   0.0
3        0.0      0.166667   0.833333   0.090917                   0.0
4        0.0      0.166667   0.666667   0.272750                   0.0
...      ...           ...        ...        ...                   ...
61064    0.0      0.000000   0.500000   0.545500                   0.0
61065    0.0      0.000000   0.833333   0.545500                   0.0
61066    0.0      0.000000   0.500000   0.545500                   0.0
61067    0.0      0.000000   0.833333   0.545500                   0.0
61068    0.0      0.000000   0.500000   0.545500                   0.0

       gill-attachment  gill-spacing  gill-color  stem-height  stem-width  \
0             0.857500           0.5    0.909167     0.416667    0.083333
1             0.857500           0.5    0.909167     0.500000    0.166667
2             0.857500           0.5    0.909167     0.500000    0.166667
3             0.857500           0.5    0.909167     0.416667    0.083333
4             0.857500           0.5    0.909167     0.416667    0.083333
...                ...           ...         ...          ...         ...
61064         0.428333           0.0    0.363667     0.083333    0.000000
61065         0.428333           0.0    0.363667     0.083333    0.000000
61066         0.428333           0.0    0.363667     0.083333    0.000000
61067         0.428333           0.0    0.363667     0.083333    0.000000
61068         0.428333           0.0    0.363667     0.083333    0.000000

       stem-surface  stem-color  veil-type  veil-color  has-ring  ring-type  \
0          0.375000    0.833333   0.000000    0.916667       0.0   0.727333
1          0.375000    0.833333   0.000000    0.916667       0.0   0.727333
2          0.375000    0.833333   0.000000    0.916667       0.0   0.727333
3          0.375000    0.833333   0.000000    0.916667       0.0   0.090917
4          0.375000    0.833333   0.000000    0.916667       0.0   0.090917
...             ...         ...        ...         ...       ...        ...
61064      0.583333    0.333333   0.583333         NaN       1.0   0.272750
61065      0.583333    0.333333   0.583333         NaN       1.0   0.272750
61066      0.583333    0.333333   0.583333         NaN       1.0   0.272750
61067      0.583333    0.333333   0.583333         NaN       1.0   0.272750
61068      0.583333    0.333333   0.583333         NaN       1.0   0.272750

       spore-print-color   hábitat    season
0               0.416667  0.428333  0.666667
1               0.416667  0.428333  0.333333
2               0.416667  0.428333  0.666667
3               0.416667  0.428333  0.666667
4               0.416667  0.428333  0.666667
...                  ...       ...       ...
61064           0.416667  0.428333  0.000000
61065           0.416667  0.428333  0.000000
61066           0.416667  0.428333  0.333333
61067           0.416667  0.428333  0.333333
61068           0.416667  0.428333  0.333333

[61069 rows x 19 columns]
```

Análisis de Correlaciones:

Calculando Matriz Correlación

La misma nos muestra la correlación entre todas las variables numéricas de tu conjunto de datos. Un valor cercano a 1 indica una fuerte correlación positiva, un valor cercano a -1 indica una fuerte correlación negativa, y un valor cercano a 0 indica una baja correlación.

```
df_stats = df.describe().T                              # Estadisticas descriptivas del Da
taFrame traspuestas para facilitar la normalización
df_n = (df - df_stats['mean']) / df_stats['std']    # Normalizo el Data Frame, a cada
valor le resto la media y lo divido por la desviación estándar
df_n.describe()

df_n.corr()
```

```
                          class  cap-diameter  cap-shape  cap-color  \
class                  1.000000      0.171592   0.180186   0.234617
cap-diameter           0.171592      1.000000   0.109947  -0.003012
cap-shape              0.180186      0.109947   1.000000  -0.032422
cap-color              0.234617     -0.003012  -0.032422   1.000000
does-bruise-or-bleed   0.019889      0.172994   0.059482  -0.062094
gill-attachment        0.199589      0.323842   0.205439   0.065166
gill-spacing           0.107190      0.044362   0.158788   0.026272
gill-color             0.182936      0.087594   0.100004   0.060382
stem-height            0.114401      0.374196   0.184356   0.026601
stem-width             0.196259      0.638695   0.133522   0.053590
stem-surface           0.199410      0.001100   0.099058   0.116330
stem-color             0.235295      0.027054  -0.016841   0.266865
veil-type              0.052677     -0.082287   0.005281   0.036567
veil-color             0.339433      0.112367  -0.103018  -0.213543
has-ring               0.057559     -0.057616  -0.036800  -0.014603
ring-type              0.152631      0.131434   0.041431   0.084947
spore-print-color      0.127169      0.044257   0.066556   0.064043
hábitat                0.166170      0.117337   0.131941   0.022427
season                 0.088445      0.056138  -0.101134   0.056055

                       does-bruise-or-bleed  gill-attachment  gill-spacing  \
class                              0.019889         0.199589      0.107190
cap-diameter                       0.172994         0.323842      0.044362
cap-shape                          0.059482         0.205439      0.158788
cap-color                         -0.062094         0.065166      0.026272
does-bruise-or-bleed               1.000000         0.157100     -0.001115
gill-attachment                    0.157100         1.000000     -0.030788
gill-spacing                      -0.001115        -0.030788      1.000000
gill-color                        -0.028811        -0.005954      0.127342
stem-height                        0.069007         0.193936      0.051840
stem-width                         0.167987         0.358039     -0.066270
stem-surface                      -0.055949        -0.039031      0.248879
stem-color                        -0.009960        -0.027848      0.102164
veil-type                          0.038548        -0.110844      0.022489
veil-color                        -0.049337         0.060747      0.158718
has-ring                          -0.019195        -0.040525      0.099011
ring-type                          0.086251         0.003488      0.088046
spore-print-color                 -0.034266         0.071854      0.065365
hábitat                            0.026595         0.148102     -0.090771
season                            -0.103325        -0.043590     -0.035367

                       gill-color  stem-height  stem-width  stem-surface  \
```

class	0.182936	0.114401	0.196259	0.199410
cap-diameter	0.087594	0.374196	0.638695	0.001100
cap-shape	0.100004	0.184356	0.133522	0.099058
cap-color	0.060382	0.026601	0.053590	0.116330
does-bruise-or-bleed	-0.028811	0.069007	0.167987	-0.055949
gill-attachment	-0.005954	0.193936	0.358039	-0.039031
gill-spacing	0.127342	0.051840	-0.066270	0.248879
gill-color	1.000000	0.113758	0.081692	0.067623
stem-height	0.113758	1.000000	0.399397	0.027557
stem-width	0.081692	0.399397	1.000000	-0.083331
stem-surface	0.067623	0.027557	-0.083331	1.000000
stem-color	0.218346	0.125464	0.069647	0.067465
veil-type	-0.209686	-0.285215	-0.023492	0.035304
veil-color	0.425322	0.182905	0.115792	-0.096746
has-ring	0.012937	-0.265256	0.013889	0.110324
ring-type	0.063993	0.263741	0.034503	-0.043729
spore-print-color	0.031174	0.026979	0.172169	0.098665
hábitat	0.036978	0.057441	0.141539	0.078629
season	-0.035379	-0.038792	0.021901	-0.008538

	stem-color	veil-type	veil-color	has-ring	ring-type	\
class	0.235295	0.052677	0.339433	0.057559	0.152631	
cap-diameter	0.027054	-0.082287	0.112367	-0.057616	0.131434	
cap-shape	-0.016841	0.005281	-0.103018	-0.036800	0.041431	
cap-color	0.266865	0.036567	-0.213543	-0.014603	0.084947	
does-bruise-or-bleed	-0.009960	0.038548	-0.049337	-0.019195	0.086251	
gill-attachment	-0.027848	-0.110844	0.060747	-0.040525	0.003488	
gill-spacing	0.102164	0.022489	0.158718	0.099011	0.088046	
gill-color	0.218346	-0.209686	0.425322	0.012937	0.063993	
stem-height	0.125464	-0.285215	0.182905	-0.265256	0.263741	
stem-width	0.069647	-0.023492	0.115792	0.013889	0.034503	
stem-surface	0.067465	0.035304	-0.096746	0.110324	-0.043729	
stem-color	1.000000	-0.095111	0.069160	-0.092967	0.082054	
veil-type	-0.095111	1.000000	-0.420073	0.347093	-0.143450	
veil-color	0.069160	-0.420073	1.000000	-0.154057	0.252383	
has-ring	-0.092967	0.347093	-0.154057	1.000000	-0.337493	
ring-type	0.082054	-0.143450	0.252383	-0.337493	1.000000	
spore-print-color	0.093605	0.032802	-0.111352	0.062947	0.056378	
hábitat	-0.000248	-0.031177	0.101149	-0.038203	0.031438	
season	0.051795	0.000258	0.173727	-0.029059	-0.022074	

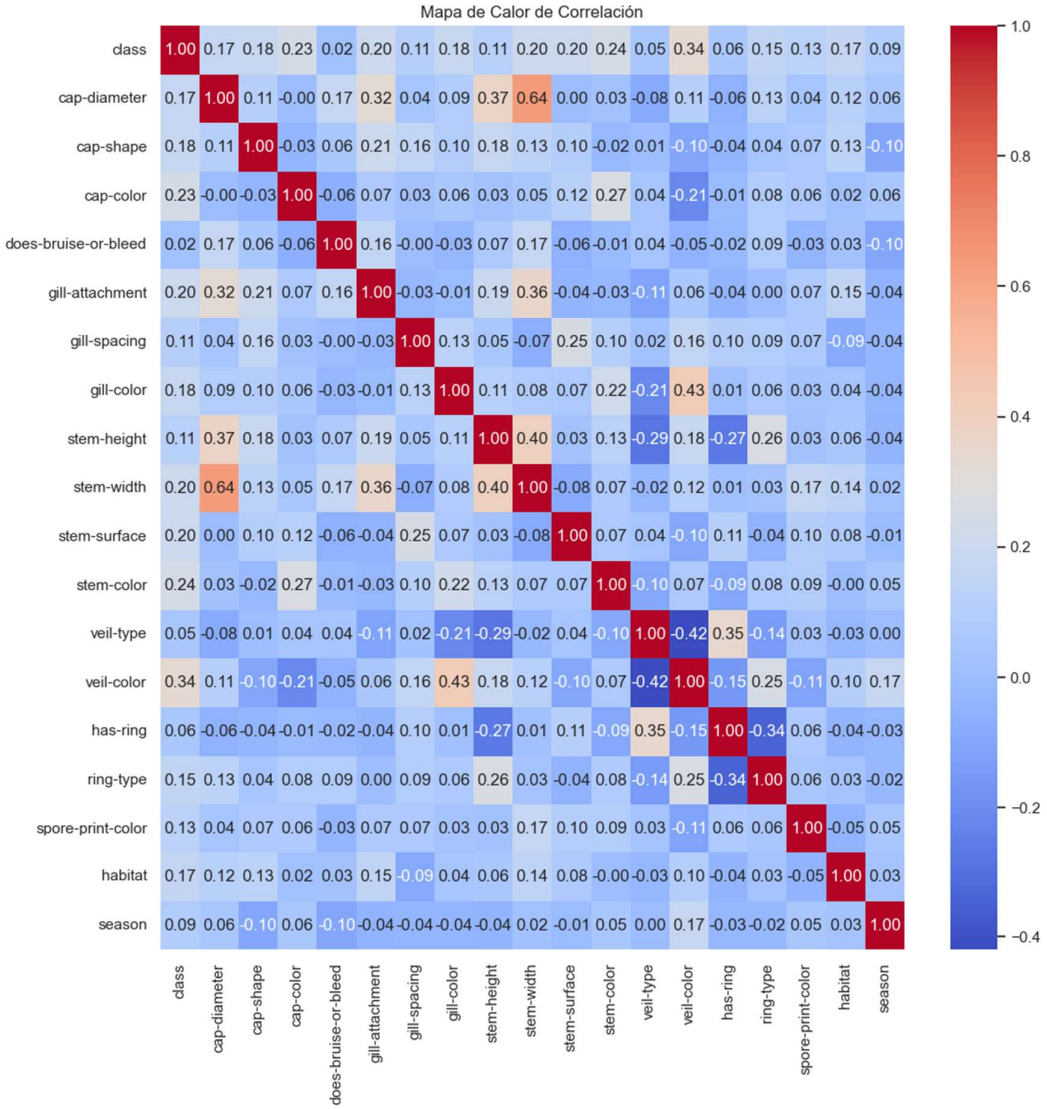
	spore-print-color	hábitat	season
class	0.127169	0.166170	0.088445
cap-diameter	0.044257	0.117337	0.056138
cap-shape	0.066556	0.131941	-0.101134
cap-color	0.064043	0.022427	0.056055
does-bruise-or-bleed	-0.034266	0.026595	-0.103325
gill-attachment	0.071854	0.148102	-0.043590
gill-spacing	0.065365	-0.090771	-0.035367
gill-color	0.031174	0.036978	-0.035379
stem-height	0.026979	0.057441	-0.038792
stem-width	0.172169	0.141539	0.021901
stem-surface	0.098665	0.078629	-0.008538
stem-color	0.093605	-0.000248	0.051795
veil-type	0.032802	-0.031177	0.000258
veil-color	-0.111352	0.101149	0.173727
has-ring	0.062947	-0.038203	-0.029059
ring-type	0.056378	0.031438	-0.022074
spore-print-color	1.000000	-0.047750	0.046499

```
hábitat                    -0.047750  1.000000  0.029496
season                      0.046499  0.029496  1.000000
```

Analizando la Matriz Correlación

Con la representación visual de la matriz podemos identificar y analizar rápidamente las correlaciones más fuertes utilizando un mapa de calor (heatmap).

```python
import seaborn as sns
import matplotlib.pyplot as plt

matriz_correlacion = df.corr() # Calculo la matriz de correlación

sns.set(style="whitegrid") # Configuro el estilo de seaborn

plt.figure(figsize=(12, 12)) # Configurar el diseño del gráfico

sns.heatmap(matriz_correlacion, annot=True, fmt=".2f", cmap='coolwarm') # Creo el mapa de calor de correlación con los valores numéricos

plt.title('Mapa de Calor de Correlación') # Añado el título

plt.show()# Mostrar el gráfico
```



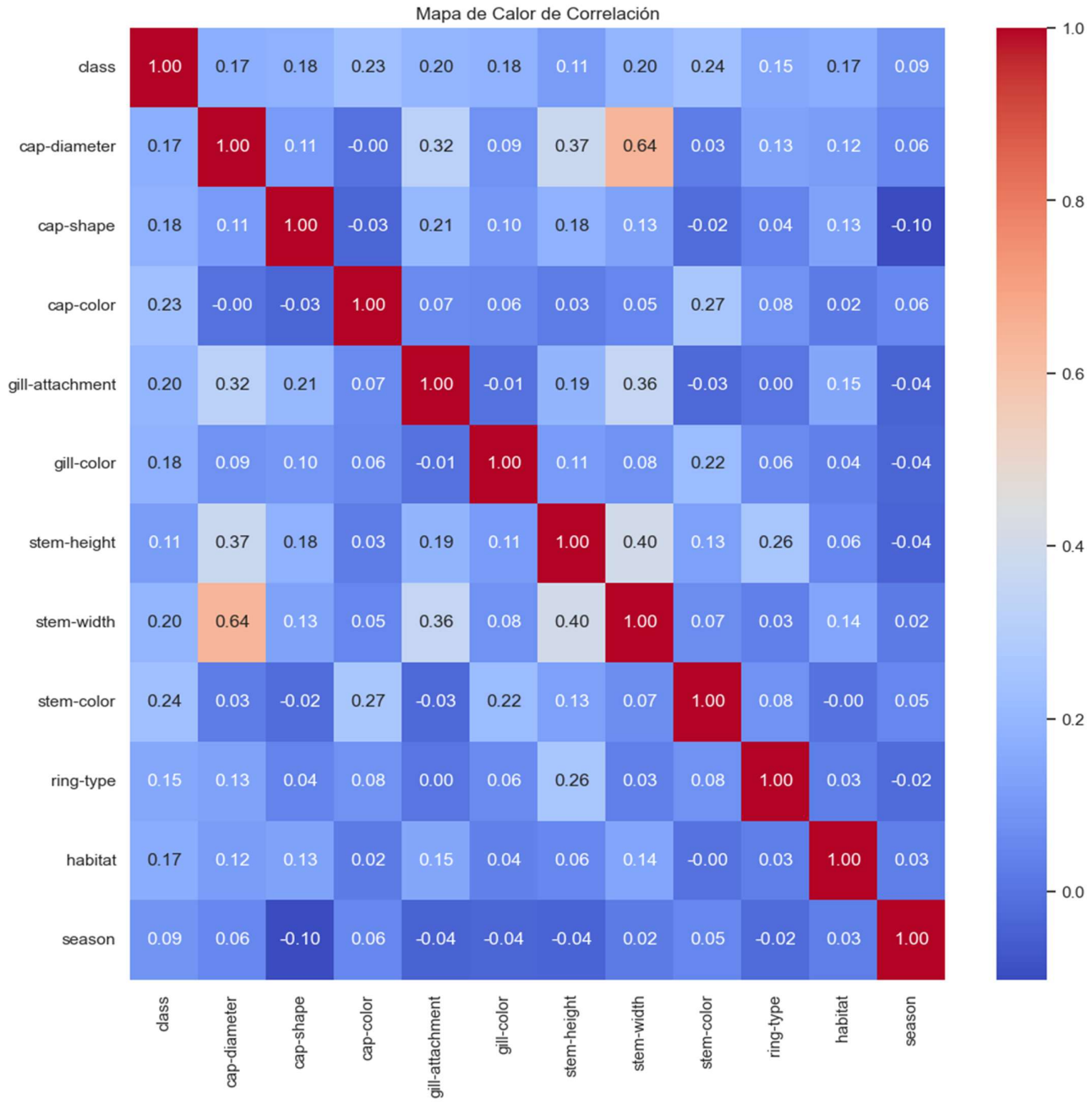
1. **'does-bruise-or-bleed'** y **'has-ring'** no considero que tengan gran impacto a la hora de la determinar si un hongo es comestible o no, por ende, no las tendré en cuenta.
2. **'gill-spacing'** y **'veil-type'** tampoco tienen demasiada relación y tienen muchas filas vacías por ende tampoco las tendré en cuenta.
3. **'veil-color'** es la variable que más aporta pero debido a que tiene muchos registros vacíos, no me queda otra que eliminarla ya que al eliminar los datos atípicos, me hace eliminar casi todo el data frame. Algo similar pasa con **stem-surface**
4. Tanto **'gill-color'**, **'stem-color'** y **'cap-color'**, son las columnas que más aportan para determinar si un hongo es comestible o no, de más está decir que el resto también aporta su granito de arena.

```
df = df.drop(columns=['does-bruise-or-bleed', 'gill-spacing', 'veil-type', 'has-ring',
'veil-color', 'spore-print-color', 'stem-surface'])

df
```

```
       class  cap-diameter  cap-shape  cap-color  gill-attachment  gill-color  \
0        0.0      0.166667   0.666667   0.272750         0.857500    0.909167
1        0.0      0.250000   0.666667   0.272750         0.857500    0.909167
2        0.0      0.166667   0.666667   0.272750         0.857500    0.909167
3        0.0      0.166667   0.833333   0.090917         0.857500    0.909167
4        0.0      0.166667   0.666667   0.272750         0.857500    0.909167
...      ...           ...        ...        ...              ...         ...
61064    0.0      0.000000   0.500000   0.545500         0.428333    0.363667
61065    0.0      0.000000   0.833333   0.545500         0.428333    0.363667
61066    0.0      0.000000   0.500000   0.545500         0.428333    0.363667
61067    0.0      0.000000   0.833333   0.545500         0.428333    0.363667
61068    0.0      0.000000   0.500000   0.545500         0.428333    0.363667

       stem-height  stem-width  stem-color  ring-type   hábitat    season
0         0.416667    0.083333    0.833333   0.727333  0.428333  0.666667
1         0.500000    0.166667    0.833333   0.727333  0.428333  0.333333
2         0.500000    0.166667    0.833333   0.727333  0.428333  0.666667
3         0.416667    0.083333    0.833333   0.090917  0.428333  0.666667
4         0.416667    0.083333    0.833333   0.090917  0.428333  0.666667
...            ...         ...         ...        ...       ...       ...
61064     0.083333    0.000000    0.333333   0.272750  0.428333  0.000000
61065     0.083333    0.000000    0.333333   0.272750  0.428333  0.000000
61066     0.083333    0.000000    0.333333   0.272750  0.428333  0.333333
61067     0.083333    0.000000    0.333333   0.272750  0.428333  0.333333
61068     0.083333    0.000000    0.333333   0.272750  0.428333  0.333333

[61069 rows x 12 columns]
```

```
import seaborn as sns
import matplotlib.pyplot as plt

matriz_correlacion = df.corr() # Calculo la matriz de correlación

sns.set(style="whitegrid") # Configuro el estilo de seaborn

plt.figure(figsize=(12, 12)) # Configurar el diseño del gráfico

sns.heatmap(matriz_correlacion, annot=True, fmt=".2f", cmap='coolwarm') # Creo el mapa
de calor de correlación con los valores numéricos
```

```
plt.title('Mapa de Calor de Correlación') # Añado el título

plt.show()# Mostrar el gráfico
```



Análisis de Factibilidad:

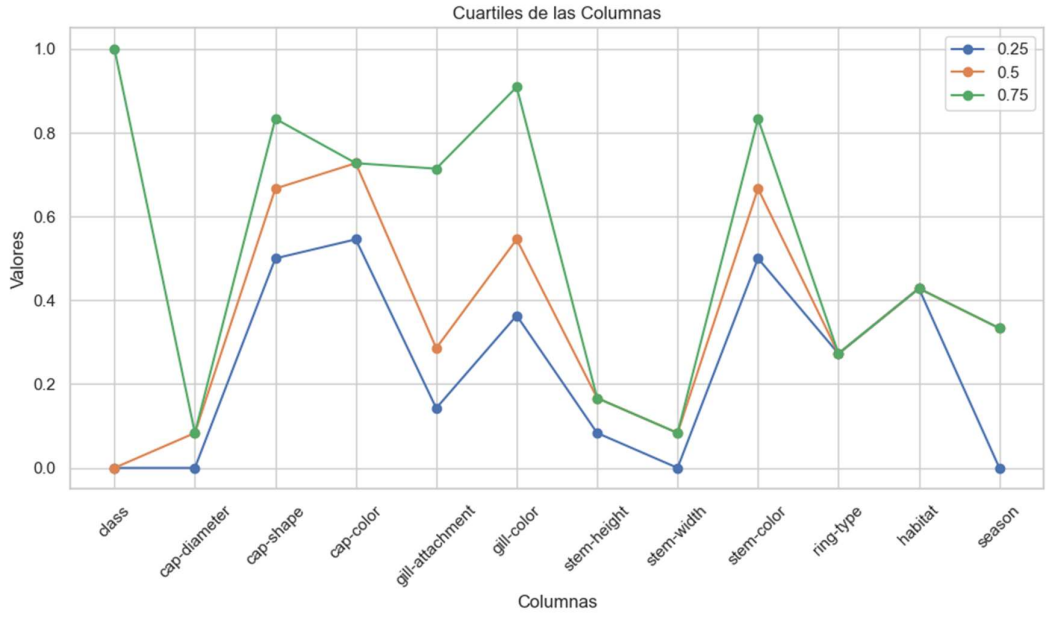
¿Es esta base de datos adecuada para entrenar una red neuronal de clasificación?

La base de datos cuenta con un tamaño suficiente de aproximadamente 61000 registros (luego de limpiar los datos atípicos queda de 22000 valores aprox) y considero que esta cantidad es más que suficiente para el análisis y modelización de una red neuronal. Las características seleccionadas como entradas son relevantes para la clasificación. Considero que esta base de datos es ideal para entrenar una red neuronal de clasificación y obtener resultados precisos.

El **propósito** de entrenar esta red neuronal será predecir si un hongo encontrado en la naturaleza es comestible o no, según determinadas características como pueden ser el color, el tamaño, la forma, etc.

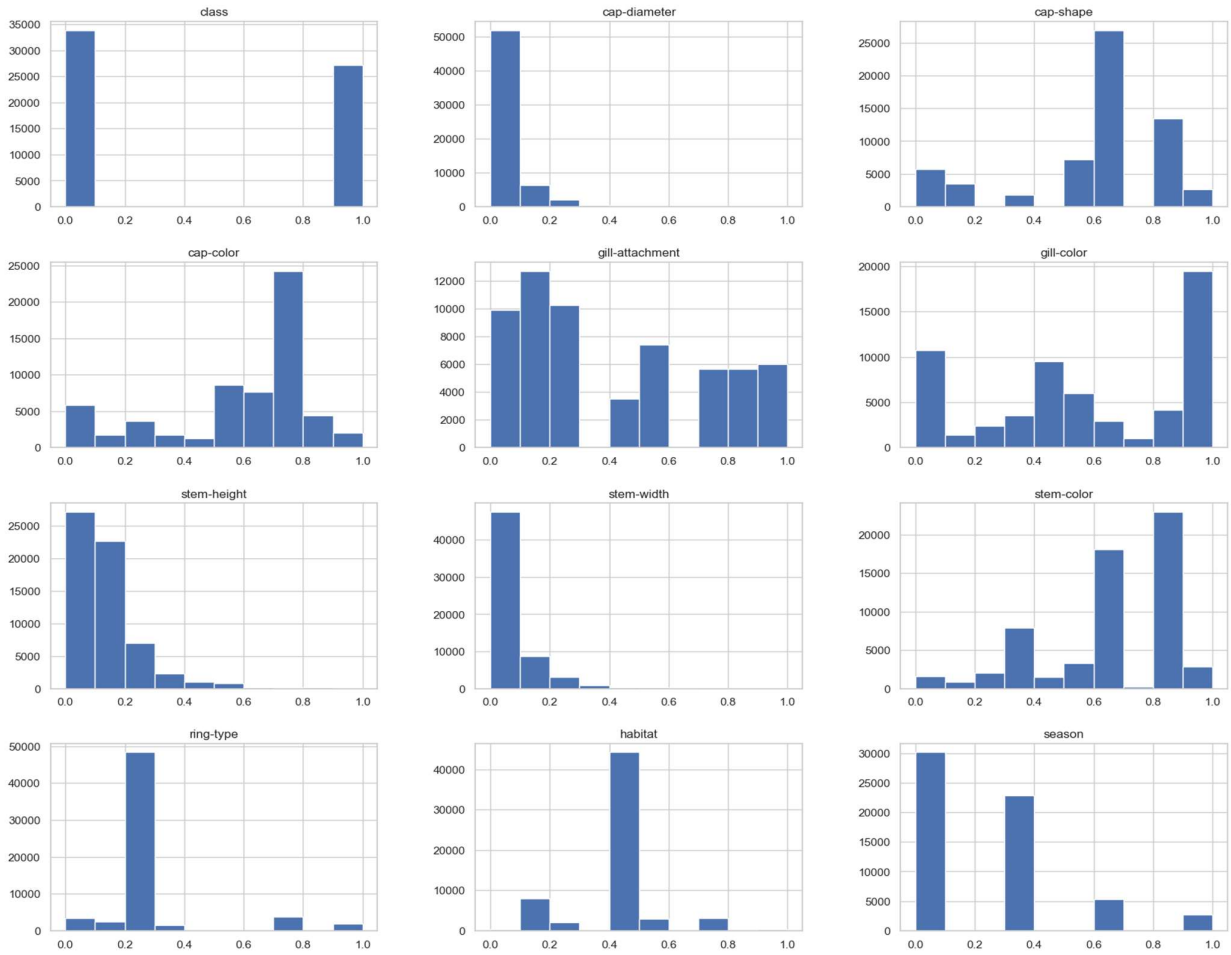
Detección de Valores Atípicos y Mediana

```
import matplotlib.pyplot as plt

# Obtener nombres de todas las columnas excepto la última
columnas = df.columns

# Configurar el diseño del gráfico
fig, ax = plt.subplots(figsize=(10, 6))

# Calcular los cuartiles para cada columna
quartiles = df.quantile([ 0.25, 0.5, 0.75])

# Transponer los datos para que cada fila represente un cuartil
quartiles = quartiles.T

# Graficar los cuartiles
for q in quartiles.columns:
    ax.plot(columnas, quartiles[q], marker='o', label=q)

# Añadir las etiquetas de los ejes y el título
ax.set_xlabel('Columnas')
ax.set_ylabel('Valores')
ax.set_title('Cuartiles de las Columnas')
ax.legend()

# Rotar las etiquetas del eje x para mayor claridad
plt.xticks(rotation=45)

# Mostrar el gráfico
plt.tight_layout()
plt.show()
```



```
df.hist(figsize=(20, 17))
```

```
array([[<Axes: title={'center': 'class'}>,
        <Axes: title={'center': 'cap-diameter'}>,
        <Axes: title={'center': 'cap-shape'}>],
       [<Axes: title={'center': 'cap-color'}>,
        <Axes: title={'center': 'gill-attachment'}>,
        <Axes: title={'center': 'gill-color'}>],
       [<Axes: title={'center': 'stem-height'}>,
        <Axes: title={'center': 'stem-width'}>,
        <Axes: title={'center': 'stem-color'}>],
       [<Axes: title={'center': 'ring-type'}>,
        <Axes: title={'center': 'hábitat'}>,
        <Axes: title={'center': 'season'}>]], dtype=object)
```




```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Identificar valores atípicos utilizando el criterio del IQR
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))

# Contar el número de valores atípicos por fila
num_outliers = outliers.sum()
print("Filas con Valores Atípicos:\n", num_outliers)

# Saco del df las filas con valores atípicos (y reinserto la columna de salida origina
l)
df = df.where(np.invert(outliers))
```

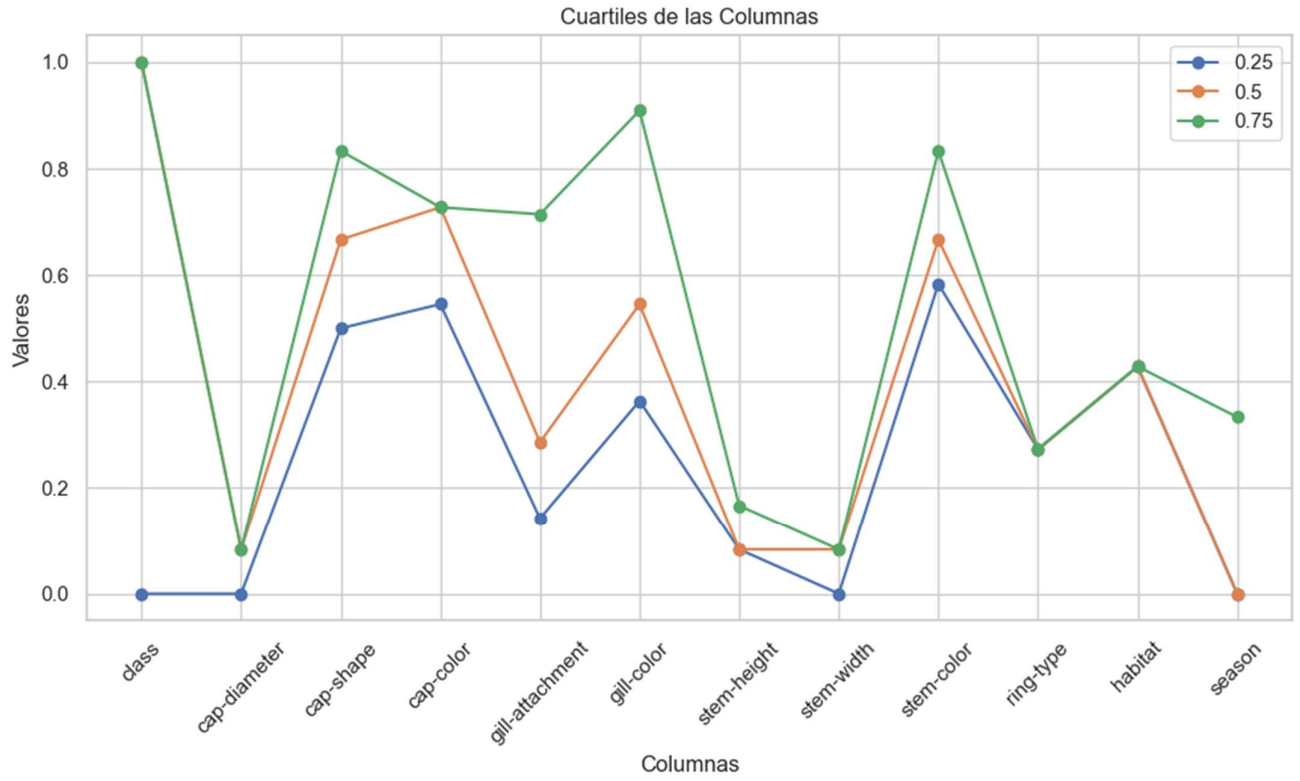
```
Filas con Valores Atípicos:
 class                  0
cap-diameter        2729
cap-shape              0
cap-color          11176
gill-attachment        0
gill-color             0
stem-height         4391
stem-width          4830
stem-color             0
ring-type          12708
hábitat            16860
season              2727
dtype: int64
```

Analizando los dos graticos considero tener datos atípicos en mi Base de Datos, por ende, pasaremos a eliminarlos

```
# Función para eliminar outliers usando la regla de las 3 desviaciones estándar
def remove_outliers(col):
    z_scores = np.abs((col - col.mean()) / col.std())
    threshold = 3
    return col[z_scores < threshold]

# Aplicar la función a cada columna y crear un nuevo DataFrame
df_limpia = df.apply(remove_outliers)

df_limpia = df_limpia.dropna(how='any')

df_limpia
```

```
       class  cap-diameter  cap-shape  cap-color  gill-attachment  gill-color  \
2472     1.0      0.000000   0.333333   0.727333         0.857500    0.909167
2481     1.0      0.000000   0.333333   0.727333         0.857500    0.909167
2485     1.0      0.083333   0.333333   0.727333         0.857500    0.909167
2495     1.0      0.083333   0.333333   0.727333         0.857500    0.909167
2506     1.0      0.083333   0.333333   0.727333         0.857500    0.909167
...      ...           ...        ...        ...              ...         ...
61064    0.0      0.000000   0.500000   0.545500         0.428333    0.363667
61065    0.0      0.000000   0.833333   0.545500         0.428333    0.363667
61066    0.0      0.000000   0.500000   0.545500         0.428333    0.363667
61067    0.0      0.000000   0.833333   0.545500         0.428333    0.363667
61068    0.0      0.000000   0.500000   0.545500         0.428333    0.363667

       stem-height  stem-width  stem-color  ring-type   hábitat    season
2472      0.250000    0.083333    0.833333    0.27275  0.428333  0.000000
2481      0.250000    0.083333    0.666667    0.27275  0.428333  0.333333
2485      0.250000    0.083333    0.666667    0.27275  0.428333  0.000000
2495      0.250000    0.083333    0.666667    0.27275  0.428333  0.333333
2506      0.250000    0.083333    0.666667    0.27275  0.428333  0.333333
...            ...         ...         ...        ...       ...       ...
61064     0.083333    0.000000    0.333333    0.27275  0.428333  0.000000
61065     0.083333    0.000000    0.333333    0.27275  0.428333  0.000000
61066     0.083333    0.000000    0.333333    0.27275  0.428333  0.333333
61067     0.083333    0.000000    0.333333    0.27275  0.428333  0.333333
61068     0.083333    0.000000    0.333333    0.27275  0.428333  0.333333

[22418 rows x 12 columns]
```

```python
import matplotlib.pyplot as plt

# Obtener nombres de todas las columnas excepto la última
columnas = df_limpia.columns

# Configurar el diseño del gráfico
fig, ax = plt.subplots(figsize=(10, 6))

# Calcular los cuartiles para cada columna
quartiles = df_limpia.quantile([ 0.25, 0.5, 0.75])

# Transponer los datos para que cada fila represente un cuartil
quartiles = quartiles.T

# Graficar los cuartiles
for q in quartiles.columns:
    ax.plot(columnas, quartiles[q], marker='o', label=q)

# Añadir las etiquetas de los ejes y el título
ax.set_xlabel('Columnas')
ax.set_ylabel('Valores')
ax.set_title('Cuartiles de las Columnas')
ax.legend()

# Rotar las etiquetas del eje x para mayor claridad
plt.xticks(rotation=45)

# Mostrar el gráfico
plt.tight_layout()
plt.show()
```

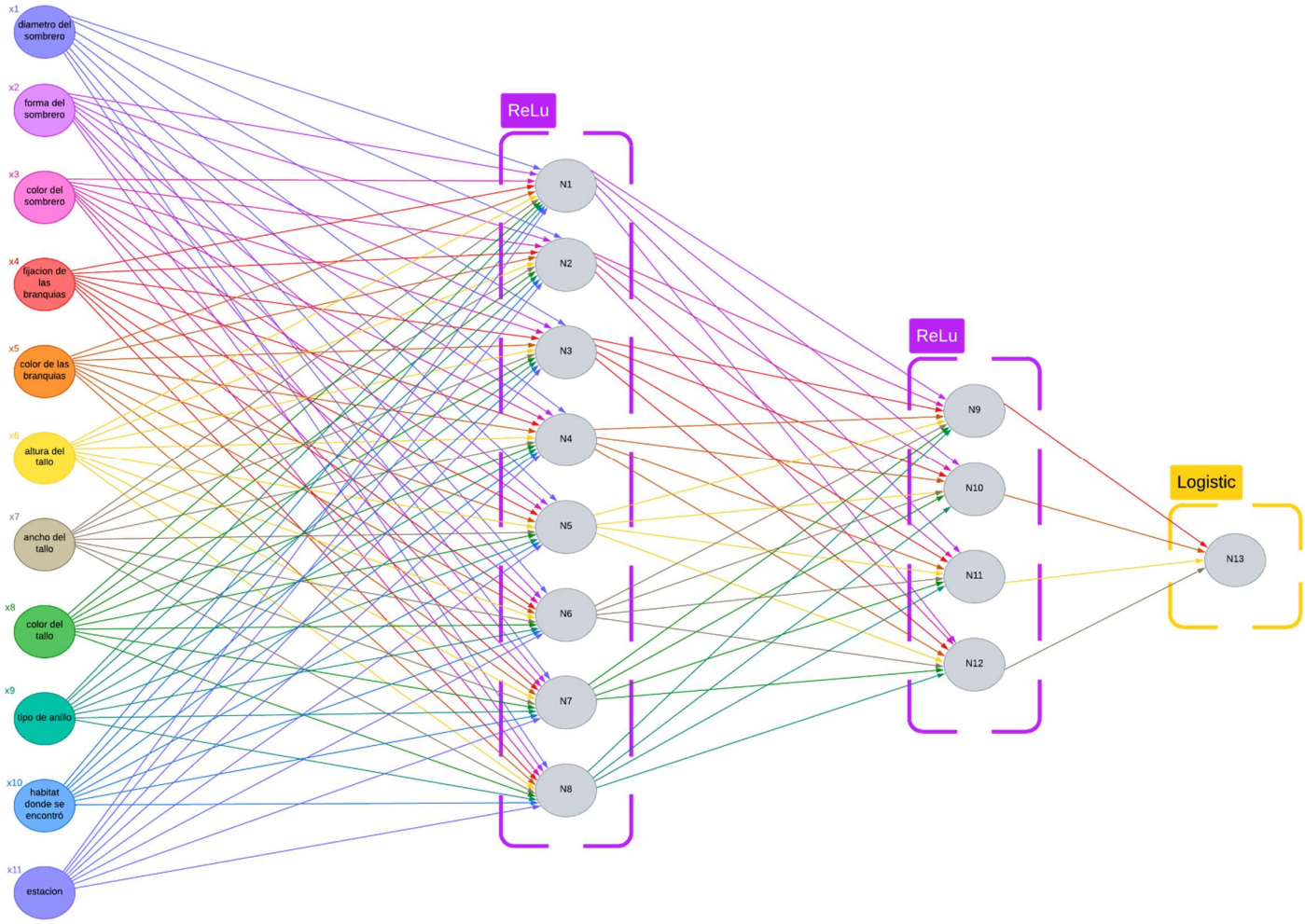


Luego de limpiar los datos atípicos de mi Data Frame los guardamos en otro archivo para luego realizar la Red Neuronal.

```
df_limpia.to_csv("mushrooms_limpios.csv", index=False)
```

Desarrollo de la Red Neuronal

Arquitectura de la Red



Como podemos observar en el grafico anterior, la red tendrá **2 capas ocultas y 1 de salida**.

En la **primer capa oculta** contaremos con **8 neuronas**, en la **segunda capa oculta** tendremos **4 neuronas** y en la **capa de salida tendremos 1 neurona** que nos indicara si el hongo es **comestible o no**.

Para las **capas ocultas** utilizaremos la **función de activación ReLu** ya que convierte los valores negativos en 0 y mitiga el problema del gradiente desvaneciente. Por otro lado, para la **capa de salida** utilizaremos la **función de activación Logística** ya que devuelve valores entre 0 y 1 y eso es ideal para nuestra red.

Implementación en numpy de la Red

Forward Propagation

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

all_data = pd.read_csv("mushrooms_limpios.csv") #22418 (Cantidad de registros)

all_inputs = all_data.iloc[:, 1:].values # Selecciono todas las columnas de entrada de
l conjunto de datos, menos la primera que es la de salida
all_outputs = all_data.iloc[:, 0].values # Selecciono la columna de salida del conjunt
o de datos

scaler = StandardScaler() # Escalo los datos de entrada
all_inputs = scaler.fit_transform(all_inputs) # Transformo las entradas para tener una
media de 0 y una desviación estándar de 1

X_train, X_test, Y_train, Y_test = train_test_split(all_inputs, all_outputs, test_size
=1 / 3) # Divido los conjuntos de datos de entrenamiento y prueba
n = X_train.shape[0]

# Defino funciones de activación
relu = lambda x: np.maximum(x, 0)  # ReLU porque no tengo valores negativos
logistic = lambda x: 1 / (1 + np.exp(-x)) # Logística para la capa de salida (salida b
inaria)

np.random.seed(22) # Inicializo semilla en 22 para reproducibilidad

w_hidden = np.random.rand(8, 11)   # Pesos primera capa oculta
w_hidden2 = np.random.rand(4, 8)   # Pesos segunda capa oculta
w_output = np.random.rand(1, 4)    # Pesos capa de salida

b_hidden = np.random.rand(8, 1)    # Sesgos primera capa oculta
b_hidden2 = np.random.rand(4, 1)   # Sesgos segunda capa oculta
b_output = np.random.rand(1, 1)    # Sesgos capa de salida

# Función de forward propagation
def forward_prop(X):
    Z1 = w_hidden @ X + b_hidden       # Entrada ponderada de la primera capa oculta
    A1 = relu(Z1)                      # ReLU en la primera capa oculta
    Z2 = w_hidden2 @ A1 + b_hidden2    # Entrada ponderada de la segunda capa oculta
    A2 = relu(Z2)                      # ReLU en la segunda capa oculta
    Z3 = w_output @ A2 + b_output      # Entrada ponderada de la capa de salida
    A3 = logistic(Z3)                  # Logistica en la capa de salida
    return Z1, A1, Z2, A2, Z3, A3
```

Precisión No entrenada

```
test_predictions = forward_prop(X_test.transpose())[5]  # Consulto A3 (Capa de salida)
test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y_test) #
Comparo valores predichos con reales
accuracy = sum(test_comparisons.astype(int) / X_test.shape[0]) # Calculo la precisión

print(accuracy)
```

```
0.5365984209821758
```

Entrenamiento y Evaluacion

Back Propagation

```
L = 0.05 # Tasa de aprendizaje

# Derivadas de las funciones de activación
d_relu = lambda x: x > 0
d_logistic = lambda x: np.exp(-x) / (1 + np.exp(-x)) ** 2

# Función de backward propagation
def backward_prop(Z1, A1, Z2, A2, Z3, A3, X, Y):
    # Calculo de las derivadas para obtener la derivada del costo con respecto a mis p
esos y sesgos (W3, W2, W1, B3, B2 B1)
    dC_dA3 = 2 * (A3 - Y)
    dA3_dZ3 = d_logistic(Z3)
    dZ3_dW3 = A2
    dZ3_dA2 = w_output
    dC_dZ3 = dC_dA3 * dA3_dZ3

    dC_dA2 = dZ3_dA2.T @ dC_dZ3
    dA2_dZ2 = d_relu(Z2)
    dZ2_dW2 = A1
    dZ2_dA1 = w_hidden2
    dC_dZ2 = dC_dA2 * dA2_dZ2

    dC_dA1 = dZ2_dA1.T @ dC_dZ2
    dA1_dZ1 = d_relu(Z1)
    dZ1_dW1 = X
    dC_dZ1 = dC_dA1 * dA1_dZ1

    # Calculo los gradientes de la función de costo con respecto a los pesos y sesgos
de cada capa.
    # keepdims sirve para mantener las dimensiones de entrada
    dC_dW3 = dC_dZ3 @ dZ3_dW3.T
    dC_dB3 = np.sum(dC_dZ3, axis=1, keepdims=True)

    dC_dW2 = dC_dZ2 @ dZ2_dW2.T
    dC_dB2 = np.sum(dC_dZ2, axis=1, keepdims=True)

    dC_dW1 = dC_dZ1 @ dZ1_dW1.T
    dC_dB1 = np.sum(dC_dZ1, axis=1, keepdims=True)

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2, dC_dW3, dC_dB3
```

Descenso de Gradiente Estocástico

```
for i in range(120_000):
    # Selecciono aleatoriamente uno de los datos de entrenamiento
    idx = np.random.choice(n, 1, replace=False)
    X_sample = X_train[idx].transpose()
    Y_sample = Y_train[idx]

    # Los paso aleatoriamente por la red neuronal
    Z1, A1, Z2, A2, Z3, A3 = forward_prop(X_sample)

    # Realizo retropropagación y devuelvo los pesos y sesgos
    dW1, dB1, dW2, dB2, dW3, dB3 = backward_prop(Z1, A1, Z2, A2, Z3, A3, X_sample, Y_s
ample)
```

```
    # Actualizo pesos y sesgos con la taza de aprendizaje
    w_hidden -= L * dW1
    b_hidden -= L * dB1
    w_hidden2 -= L * dW2
    b_hidden2 -= L * dB2
    w_output -= L * dW3
    b_output -= L * dB3
```

Precisión Red Entrenada

```
# Calculo de precisión de entrenamiento
train_predictions = forward_prop(X_train.transpose())[5]  # Consulto A3 (Capa de salid
a)
train_comparisons = np.equal((train_predictions >= .5).flatten().astype(int), Y_train)
# Comparo valores predichos con reales
train_accuracy = sum(train_comparisons.astype(int)) / X_train.shape[0] # Calculo la pr
ecisión de entrenamiento

print(f"Train Accuracy: {train_accuracy}")


# Calculo de precisión de prueba
test_predictions = forward_prop(X_test.transpose())[5]  # Consulto A3 (Capa de salida)
test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y_test) #
Comparo valores predichos con reales
test_accuracy = sum(test_comparisons.astype(int)) / X_test.shape[0] # Calculo la preci
sión de prueba

print(f"Test Accuracy: {test_accuracy}")
```

```
Train Accuracy: 0.8891267982602877
Test Accuracy: 0.8908069048574869
```

Como podemos observar en el grafico anterior, cuando alcanzamos las 120000 iteraciones aproximadamente, conseguimos la menor variación del resultado y la menor diferencia entre test y train.

Comparación con scikit-learn

Scikit-Learn

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

df = pd.read_csv("mushrooms_limpios.csv")


X = df.values[:, 1:] # Selecciono todas las columnas de entrada del conjunto de datos,
menos la primera que es la de salida
scaler = StandardScaler() # Escalo los datos de entrada
X  = scaler.fit_transform(X)

Y = df.values[:, 0] # Selecciono la columna de salida del conjunto de datos
```

```
# Separar los datos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

nn = MLPClassifier(solver='adam',
                   hidden_layer_sizes=(8, 4, ),
                   activation='relu',
                   max_iter=120_000,
                   random_state=22,
                   learning_rate_init=.05)

nn.fit(X_train, Y_train)

print("Puntaje de entrenamiento: %f" % nn.score(X_train, Y_train))
print("Puntaje de prueba: %f" % nn.score(X_test, Y_test))
```

```
Puntaje de entrenamiento: 0.891134
Puntaje de prueba: 0.894955
```

Los resultados son casi iguales, estando un poco por encima los resultados de la librería, que es normal debido a que tiene una mejor y óptima implementación.

Claramente notamos diferencia sobre todo en el tiempo de ejecución ya que mi red tarda casi 1 minuto en ejecutar el descenso de gradiente estocástico y la de scikit-learn tarda menos de 2 segundos. Esto se debe a que la librería scikit-learn utiliza métodos de optimización que le permiten ejecutarse mucho más rápido.

Conclusión

Implementar una red neuronal para clasificación binaria desde cero me ha proporcionado una comprensión más profunda del funcionamiento de estas.

A lo largo del proyecto, he aprendido a seleccionar un Data Frame, a entenderlo, a manipularlo de manera que me sirva para desarrollar mi red. Además, aprendí a limpiarlo de valores atípicos y vacíos, y a elegir cuando esos valores me sirven o me perjudican.

Luego, aprendí a desarrollar paso a paso mi red a lo largo del desarrollo, cambiando la cantidad de iteraciones, la taza de aprendizaje y analizando sus gráficos para elegir los mejores valores posibles.

Al comparar esta experiencia con el uso de librerías como scikit-learn, es evidente que construir una red neuronal desde cero ofrece una mayor comprensión del funcionamiento de esta y una mayor personalización. Sin embargo, el tiempo de desarrollo y la probabilidad de errores son mucho mayores. Teniendo esto en cuenta, usar la librería, es mucho más práctico para cuando la necesitamos para un proyecto simple o rápido y que no merezca mucho análisis, además, con la librería podemos comparar distintos métodos de aprendizaje que pueden implementarse ante el mismo problema.

En conclusión, mi opinión es que desarrollar tu propia Red te ayuda mucho más a entender realmente su funcionamiento, su poder y sus limitaciones.