Lab IAA: Entrega Final

Modelització

Autor: César Mejía Rota

INDEX

1. Introducció

Aquest treball tracta sobre l'estudi de la supervivència de pacients amb cirrosi hepàtica. El dataset consta de 17 característiques clíniques sobre cada pacient, i el seu estat final (ja sigui mort, viu o si ha rebut un transplant). L'objectiu del treball és fer un model que predigui si un pacient sobreviurá o morirá.

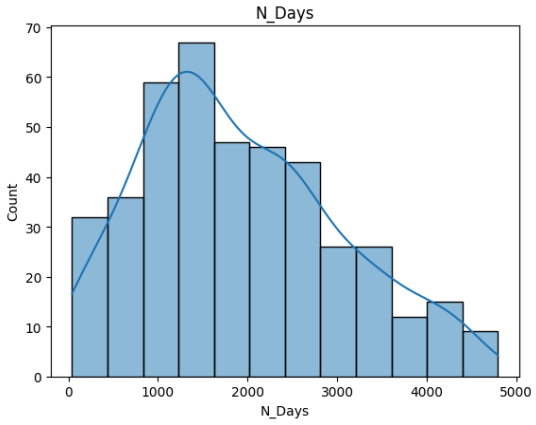
2. Anàlisi de les variables al dataset sense preprocessar

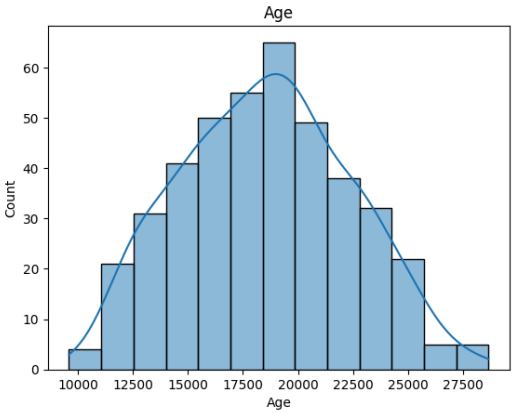
La base de dades consta de 12 variables numèriques i 7 categòriques, juntament amb la variable objectiu ('Status') que també és categórica. Les variables, pero, no venen totes de manera adequada, aleshores hem de fer unes poques coses abans de poder tractar res sobre la base de dades.

```python
df = pd.DataFrame(X, cirrhosis_patient_survival_prediction.original)

df.replace('NaNN', np.nan, inplace = True)
df.drop(['ID'], axis=1, inplace=True)

df['Stage'] = df['Stage'].astype(object)
df[['Cholesterol','Copper','Tryglicerides','Platelets']]=
df[['Cholesterol','Copper','Tryglicerides','Platelets']].astype(float)
```

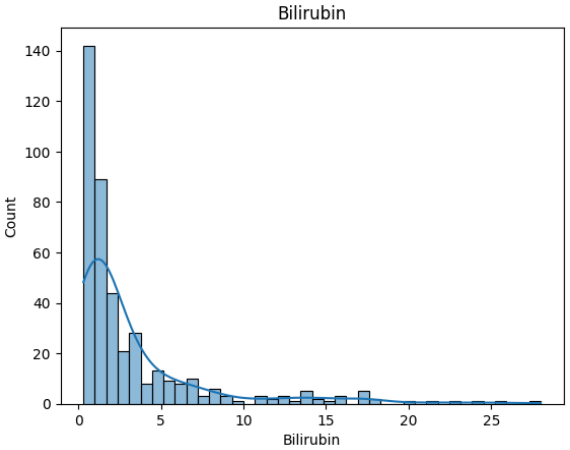
Hem de convertir la base de dades a una base de dades de pandas, que es la llibreria de python amb la que treballarem. Un cop fet això, transformem els NA ('NaNN') a 'np.nan', que es la manera en què podem treballar amb ells sense que el programa els detecti com a 'Strings'. Eliminen també la columna 'ID' de la base de dades ja que no aporta informació al model i, finalment, convertim les 5 variables mal classificades al tipus de dada que realment són.

Un cop fet això, analitzarem les variables en el dataset sense preprocessar. Tractarem primer les variables numeriques i, tot seguit, les variables numèriques.

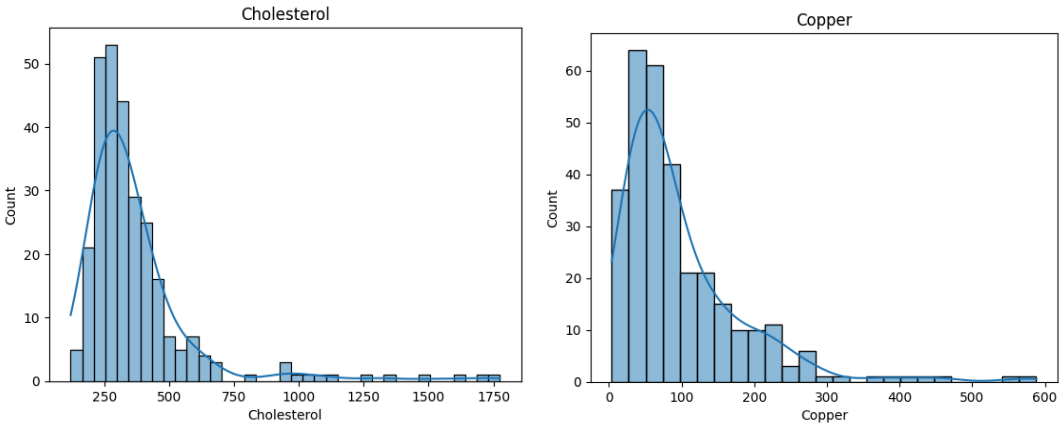




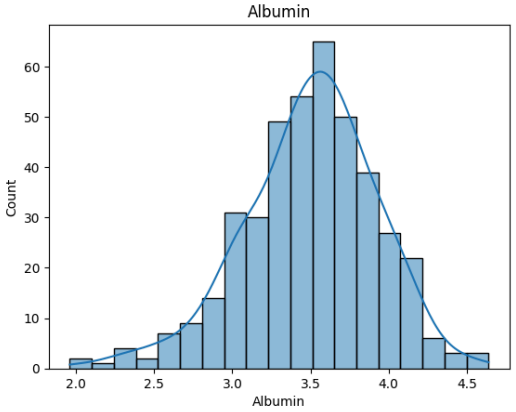
Aquests gràfics representen els dies que el pacient ha estat internat a l'hospital i l'edat del mateix, en dies. Com podem veure, totes dues distribucions semblen molt una distribució normal (mes la edad que el número de dies internat). Això ens diu que aquestas variables ja estan bastant ben distribuïdes i que el preprocessament no farà gairebé cap canvi en la seva forma.



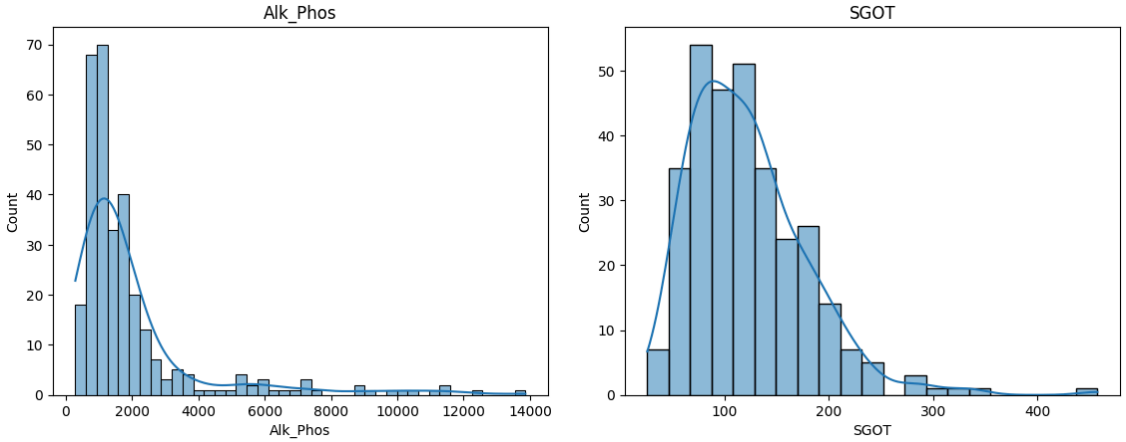
Veiem en el cas d'aquesta altre variable, la bilirrubina, la distribució segueix el que sembla una exponencial inversa, cosa que sembla raonable ja que els valors nominals de la bilirrubina son baixos en la mètrica utilitzada. Tot i així, veiem gran quantitat de outliers que intentarem eliminar durant el preprocessament.



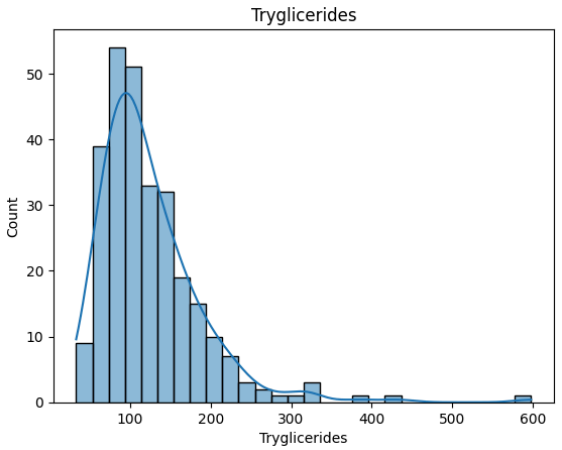
Veiem en les variables ‘Colesterol’ i ‘Copper’ que, tot i semblar que dibuixen una corba normal, aquesta esta descentralizada. Això és degut, un altre cop, als outliers que es troben a la part dreta de la gràfica. Podem veure cada cop més com el tractament d’aquests outliers és necessari per a poder fer servir aquestes dades.

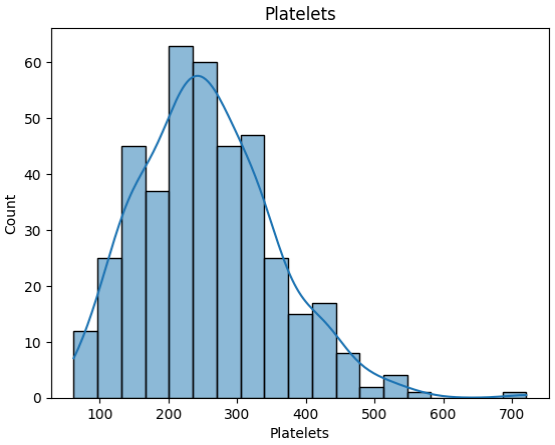


La variable ‘Albumin’ sembla estar distribuida de manera normal també, tot i que sembla tenir una petita cua a l’esquerra que a simple vista no podem dir si son outliers o no.

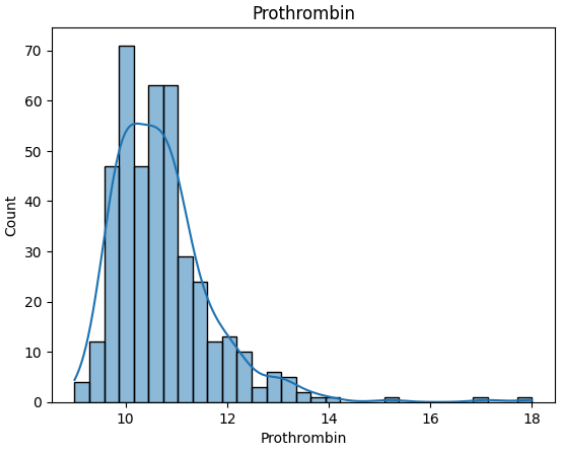


Veiem la variable ‘Alk_Phos’ (esquerra) que està molt desbalancejada i que , a més, té un molts outliers que es troben per sobre dels valors mitjans (dreta del gràfic) . La variable SGOT esta bastant més balancejada, tot i que torna a tenir outliers per sobre del valor promig, per això la corba no esta centrada (té un biaix cap a la dreta).





Ens els 'Tryglicerides' i els 'Platelets' passa una cosa similar. Tot i semblar mes o menys balancejades, veiem com les gràfiques estan mogudes cap a l'esquerra, indicant que tornen a haver-hi outliers per sobre dels valors mitjans. La variable 'Platelets' (dreta) sembla tenir una millor distribució que la variable 'Tryglicerides' (esquerra).

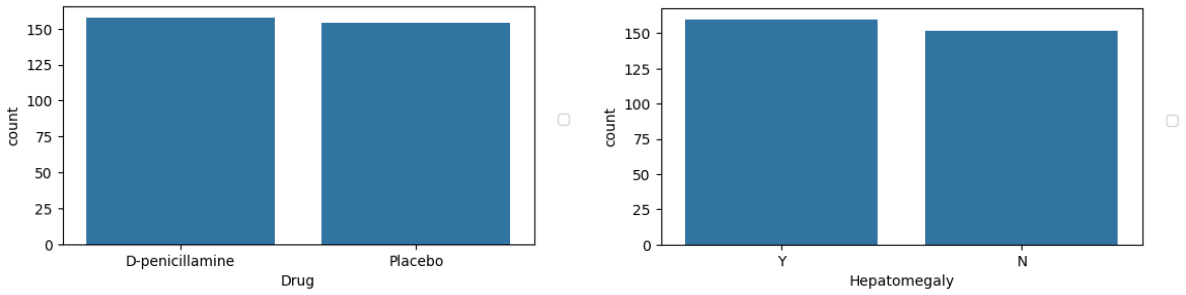


Veiem en l'últim gràfic de variables numeriques com es repeteix la tendència de les altres variables del mateix tipus. La variable, tot i no semblar estar del tot ben balancejada, segueix una tendència que recorda una gaussiana. Veiem com igualment es troba desplaçada cap a l'esquerra, cosa que indica outliers per sobre dels valors centrals.

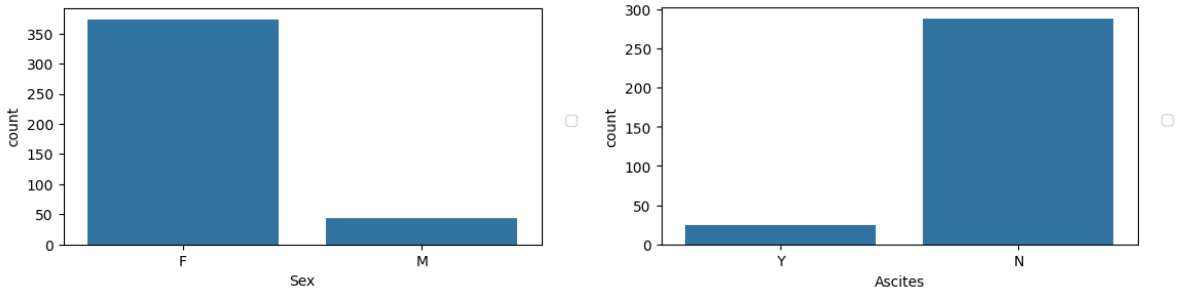
	count	mean	std	min	25%	50%	75%	max
N_Days	418.0	1917.782297	1104.672992	41.00	1092.7500	1730.00	2613.50	4795.00
Age	418.0	18533.351675	3815.845055	9598.00	15644.5000	18628.00	21272.50	28650.00
Bilirubin	418.0	3.220813	4.407506	0.30	0.8000	1.40	3.40	28.00
Cholesterol	284.0	369.510563	231.944545	120.00	249.5000	309.50	400.00	1775.00
Albumin	418.0	3.497440	0.424972	1.96	3.2425	3.53	3.77	4.64
Copper	310.0	97.648387	85.613920	4.00	41.2500	73.00	123.00	588.00
Alk_Phos	312.0	1982.655769	2140.388824	289.00	871.5000	1259.00	1980.00	13862.40
SGOT	312.0	122.556346	56.699525	26.35	80.6000	114.70	151.90	457.25
Tryglicerides	282.0	124.702128	65.148639	33.00	84.2500	108.00	151.00	598.00
Platelets	407.0	257.024570	98.325585	62.00	188.5000	251.00	318.00	721.00
Prothrombin	416.0	10.731731	1.022000	9.00	10.0000	10.60	11.10	18.00

En la taula de valors següent, veiem com les variables no estan normalitzades i com, a més dels outliers, hi ha molts missings, arribant a haber mes de 100 missings en algunes variables, com el colesterol o els triglicèrids.

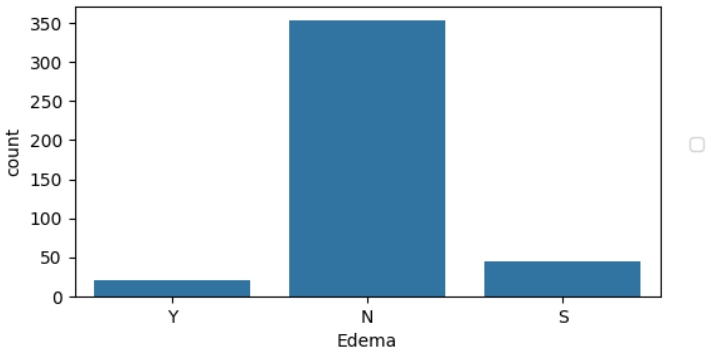
Un cop acabat amb l'analisi de les variables numeriques sense preprocessar, farem un petit anàlisi de les variables categòriques abans de tractar-les:



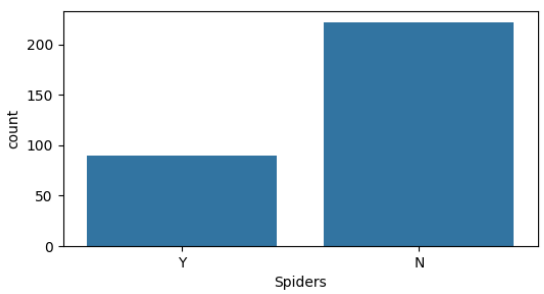
Com podem veure, les variables 'Drug' (dreta) i 'Hepatomegaly' (esquerra) están ben balancejades sense modificar-les. Era d'esperar que la variable 'Drug' estigues balancejada ja que no tindria sentit fer un estudi clínic sense donar els diferents tipus de tractament al mateix número de pacients.

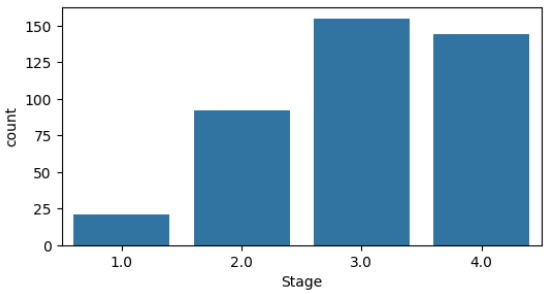


Altres variables com 'Sex' (dreta) i 'Ascites' (esquerra) estan altament desbalancejades. Veiem com hi ha moltes poques persones de sexe masculí i com també i ha molts pocs pacients amb ascites.

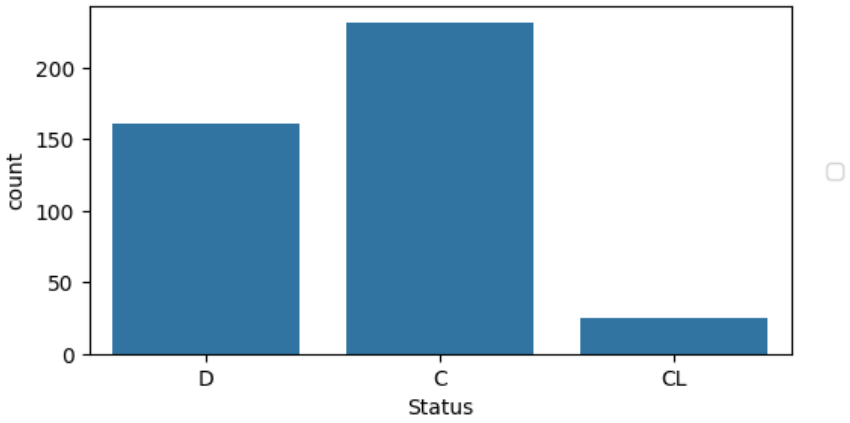


Veiem també com, similarment a les dues anteriors variables, la variable 'Edema' està molt desbalancejada, i tot els valors es concentren en 'N', deixant 'Y' i 'S' amb gairebe cap valor.





Les variables 'Spiders' (dreta) i 'Stage' (esquerra) també estan desbalancejades, pero podem observar com el desbalanceig és molt menys que en les classes anteriorment mencionades (tot i que Stage 1 té molt pocs entries). La classe 'Spiders_Y' arriba a gairebé a 100 entries, mentre que els elements de altres classes desbalancejades no arriben gairebé a 20 entries. A la classe 'Stage' veiem com 3 de les 4 categories tenen 100 o més entries ('Stage_2.0' no arriba als 100 entries per molt poquet) que es una distribució que no es exageradament desbalancejada.



Veiem com la variable 'Status', la nostre variable objectiu, està també força desbalancejada. Veiem com tot i tenir molts entries de 'D' i 'C', 'CL' no te gairebé entries. Per millorar el rendiment dels models que entrenarem, balancejarem aquesta classe (la classe resposta). Ho farem, però, després de particionar el nostra dataset.

3. Tractament de missings i de outliers

Primerament, mirem els missings de la nostra base de dades:

```python
#NaN_detection
df.isna().sum().sort_values(ascending=False)
round(100*                    (df.isnull().sum()                    /
len(df)).sort_values(ascending=False),2)
```

Aquesta funció ens retorna, primerament la quantitat de missings a cada columna en valor, i després en percentatge:

Variable	NA	NA%
Trygliceride s	136	32.54
Cholesterol	134	32.06
Copper	108	25.84
Drug	106	25.36
Ascites	106	25.36
Hepatomegaly	106	25.36
Spiders	106	25.36
SGOT	106	25.36
Alk_Phos	106	25.36
Platelets	11	2.63
Stage	6	1.44
Prothrombin	2	0.48

Com podem observar, hi ha moltes variables amb molts missings, per tant serà necessari o bé imputar-los o be eliminar-los. Degut a el petit tamany de la base de dades, és més interessant imputar-los, tot i que si trobem files amb molts missings serà possiblement millor eliminar-les, ja que imputar tants valors faria que la fila tinges massa valors generats i no suficients valors originals.

El següent que farem sera estudiar els outliers. El que farem sera detectar els outliers que tenen les nostres variables numèriques mitjançant els quartils i els transformarem a NA, per després imputar-los:

```python
for elem in Ncolumns:
  Q1 = df[elem].quantile(0.25)
  Q3 = df[elem].quantile(0.75)
  IQR = Q3 - Q1
  lim_inf = Q1 - 1.5 * IQR
  lim_sup = Q3 + 1.5 * IQR

  df.loc[(df[elem] < lim_inf) | (df[elem] > lim_sup), elem] = np.nan
df.describe().T
```

Un cop fet això tindrem encara més missings que abans, per tant el següent que farem sera eliminar les files que tinguin més missings.

# FILES	# MISSINGS
189	0
65	1
36	2
17	3
4	4
1	5
75	9
24	10
7	11

Veient la quantitat de files amb missings i quants missings te cadascuna, decidim eliminar totes les que tinguin mes de 4 missings amb el següent codi:

```python
df['missings'] = df.isnull().sum(axis=1)
df['missings'].value_counts()
df = df[df['missings'] < 4]
df = df.drop(['missings'], axis=1)
df.info()
```

Ara, després d'eliminar moltes files del dataset, veurem quants missings tenim a cada variable, on mirem els missings a les variables numeriques i categòriques per separat. D'aquesta manera, podem veure si tindrem que imputar valors numèrics, valors de classe o totes dues.

NUM DE FILES TOTALS = 307	
Variables numèriques	
Variable	# Files sense missings
N_Days	307
Age	307
Bilirubin	280
Cholesterol	263
Albumin	301
Copper	289
Alk_Phos	274
SGOT	302
Tryglicerides	271
Platelets	299
Prothrombin	296
Variables categòriques	
Status	307
Drug	307
Sex	307
Ascites	307
Hepatomegaly	307
Spiders	307
Edema	307
Stage	307

Com veiem a la taula, les columnes categòriques de la base de dades han quedat desprovistes de missings després d'eliminar les files amb més missings, i per tant només haurem d'imputar els valors faltants a les columnes numèriques.

4. Recodificació, imputació i balanceig

El següent que farem és recodificar les variables categòriques a variables booleanes i tot seguit convertim les booleanes a 1 i 0 per a que les dades puguin ser utilitzades per els models que només accepten variables numèriques com a input. Per fer això hem fet servir el següent codi:

```python
cat = cat.drop('Status', axis = 1)
cat.info()
dummies = pd.get_dummies(cat)

boolean = dummies.astype(int)
```

Un cop tenim això fet, concatenem les noves columnes a la nostre base de dades i fem 'drop' de les antigues variables categòriques:

```python
df = pd.concat([df,boolean],axis=1)
df = df.drop(cat, axis=1)
```

I ara ja tenim les variables categòriques codificades com a numeriques en columnes binàries. Fet això, ja podem particionar la nostre base de dades. Farem una partició de 80% train i 20% test utilitzant la funció 'train_test_split' de la llibreria 'model_selection' de scikit-learn.

Un cop partit el dataset, procedim a imputar els missings. Per fer això, farem servir el 'KNNImputer' de scikit-learn. Farem fit de l'imputer a la nostra divisió del train i després imputarem el train i el test amb el fit que hem fet al train. Per fer-ho farem servir el següent codi:

```python
from sklearn.impute import KNNImputer

imputerKNN = KNNImputer(n_neighbors=5)
imputerKNN = imputerKNN.fit(X_train)
X_train = imputerKNN.transform(X_train)
X_test = imputerKNN.transform(X_test)

X_train = pd.DataFrame(X_train, columns = X.columns)
X_test = pd.DataFrame(X_test, columns = X.columns)

X_train.info()
X_test.info()
```

Un cop fet això, veiem que al nostre dataset ja no hi han missings tampoc a les variables numeriques (els missings a les variables categòriques desapareixen al eliminar les files amb més de 4 missings) i ja només ens queda balancejar la classe objectiu. Per fer això, utilitzarem el mètode d'oversampling conegut com a 'SMOTE'. Decidim fer oversampling i no un altre mètode (con undersampling) ja que el dataset amb el que partim té massa poques dades com per fer altre cosa que no sigui oversampling. Utilitzarem 'SMOTE' de la llibreria 'imblearn':

```python
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=69)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

Un cop fet això només ens queda escalar o normalitzar el dataset i podrem començar a treballar en els nostres models.

Tot i que en un primer moment la idea va ser escalar el dataset, la conclusió a la que vam arribar va ser normalitzar-lo, ja que la normalització es útil i una millor opció sobre el fet de escalar a l'hora de fer PCA i d'entrenar models com el KNN.

Per normalitzar les variables numèriques fem servir 'StandardScaler' de la llibreria 'preprocessing' de scikit-learn. No només normalitza la base de dades (mitjana de 0 i desviació estándar de 1) sinó que, a més, la escala de manera que resol el problema de decisió que tenim. Si bé és cert que no escala les variables entre -1 i 1 (com era la nostra idea inicial), redueix el seu tamany considerablement i evita conflictes degut al tamany dels números de les dades (ja que els valors poden ser molt diferents depenent de la variable, on variables com la edad poden prende valors molt grans i variables com la bilirrubina agafen valors que no superen gairebé la desena). El codi emprat és el següent:

```python
scaler = StandardScaler()

scaler.fit(X_train[Ncolumns])
X_train[Ncolumns] = scaler.transform(X_train[Ncolumns])
X_test[Ncolumns] = scaler.transform(X_test[Ncolumns])
```

Un cop fet això, les variables numèriques queden escalades i normalitzades. Veiem una taula comparativa de aquestes variables abans i després de la normalització:

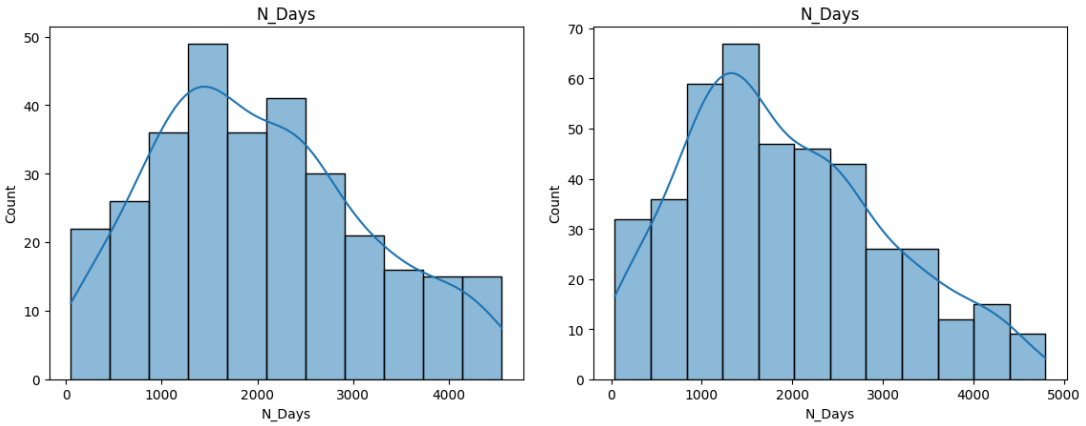
	count	mean	std	min	25%	50%	75%	max
N_Days	402.0	1814.600671	1007.607001	51.00	1047.889272	1569.616175	2384.452012	4556.00
Age	402.0	17218.933293	3739.016801	9598.00	14626.885648	16440.500000	19716.000000	28650.00
Bilirubin	402.0	2.478347	1.635403	0.30	1.000000	2.222247	3.400000	7.20
Cholesterol	402.0	334.399293	90.150648	120.00	263.000000	324.717947	393.005436	614.00
Albumin	402.0	3.526821	0.347785	2.53	3.310000	3.540000	3.770000	4.52
Copper	402.0	87.311447	50.313206	4.00	48.250000	78.172716	119.774673	243.00
Alk_Phos	402.0	1394.479432	601.072210	310.00	960.443861	1283.224578	1688.185039	3472.00
SGOT	402.0	123.454875	41.933215	26.35	92.000000	120.900000	151.900000	246.45
Tryglicerides	402.0	118.626252	41.318821	33.00	88.000000	113.014014	140.845076	243.00
Platelets	402.0	272.453369	84.192282	70.00	214.000000	278.000000	321.941418	493.00
Prothrombin	402.0	10.563218	0.683665	9.10	10.000000	10.502468	11.000000	12.70

	count	mean	std	min	25%	50%	75%	max
N_Days	402.0	0.000000e+00	1.001246	-1.752467	-0.761871	-0.243438	0.566254	2.724093
Age	402.0	-2.651279e-16	1.001246	-2.040758	-0.694107	-0.208451	0.668673	3.061048
Bilirubin	402.0	-7.070077e-17	1.001246	-1.333654	-0.905091	-0.156792	0.564266	2.890747
Cholesterol	402.0	2.695467e-16	1.001246	-2.381197	-0.792987	-0.107525	0.650901	3.105348
Albumin	402.0	-1.121270e-15	1.001246	-2.869769	-0.624210	0.037942	0.700093	2.859284
Copper	402.0	2.253587e-16	1.001246	-1.657920	-0.777333	-0.181863	0.646027	3.098243
Alk_Phos	402.0	4.418798e-17	1.001246	-1.806490	-0.723002	-0.185325	0.489245	3.460665
SGOT	402.0	-2.386151e-16	1.001246	-2.318589	-0.751053	-0.061003	0.679189	2.936774
Tryglicerides	402.0	2.651279e-16	1.001246	-2.074913	-0.742142	-0.135997	0.538411	3.013850
Platelets	402.0	3.535038e-16	1.001246	-2.407651	-0.695149	0.065963	0.588530	2.622823
Prothrombin	402.0	-1.405178e-15	1.001246	-2.142922	-0.824848	-0.088970	0.639678	3.129373

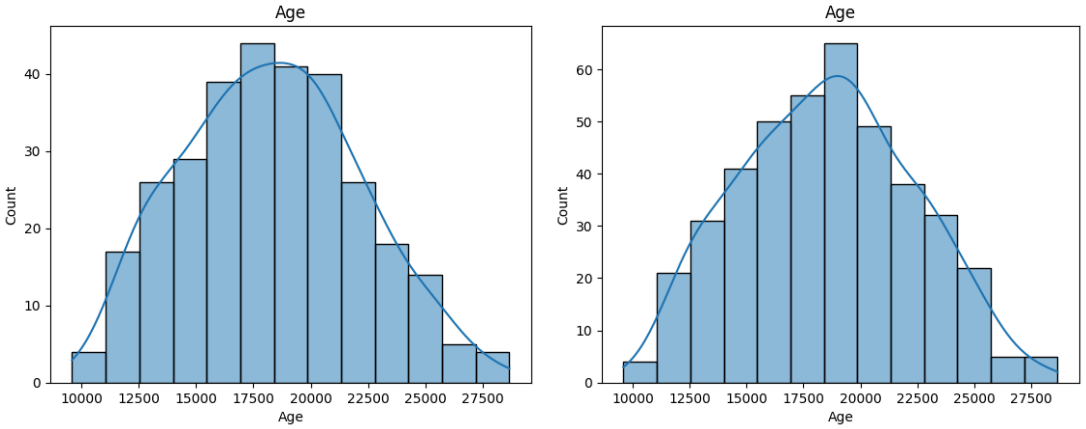
Com podem veure, els valors han sigut escalats i les dades han estat normalitzades. Veiem la normalització molt fàcilment mirant la mitjana (mean) i la desviació estàndard (std). La mitjana, tot i que no en tots els casos es 0, sempre pren un valor molt petit (de l'ordre de 10^{-15} en el cas en què es més alta, que es un número molt petit) i la desviació estàndard és en tots els casos 1 (1.001246 exactament). Veiem també com les dades han estat escalades de manera molt clara en variables com els dies (N_Days) o en l'edat (Age) que prenien abans com a valors màxim i mínim els valors més alts de gairebé cap variable del dataset i ara prenen valors molt similars a la resta de variables (entre -2.5 i 3.5, de manera aproximada)

5. Anàlisis de variables després del tractament de dades

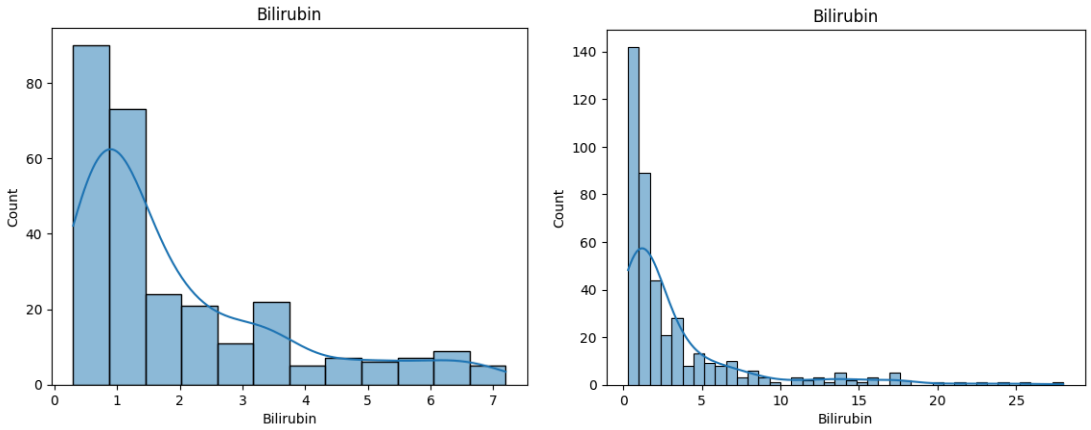
Procedirem ara a comparar les variables modificades durant el tractament de dades per veure com han sigut modificades amb l'objectiu de poder ser usades a l'hora de crear els nostres models predictius. Primerament, analitzarem les variables numèriques:



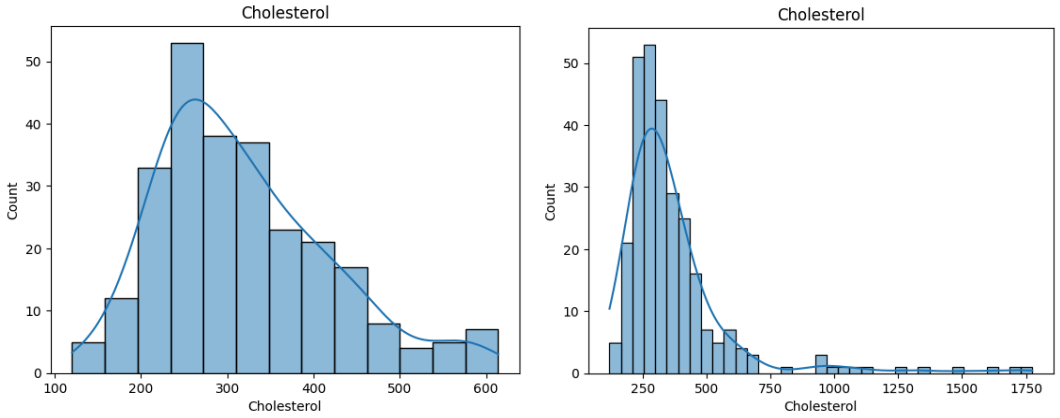
Veiem que, tal i com haviem predit, la variable 'N_Days' no ha variat gaire. Si bé és cert, la gràfica de la variable modificada (esquerra) és més plana que la gràfica de la variable sense modificar (esquerra). Això és degut a la normalització que hem dut a terme, tot i que com era previsible la variable ja tenia una distribució molt bona, i per això els canvis son poc destacables.



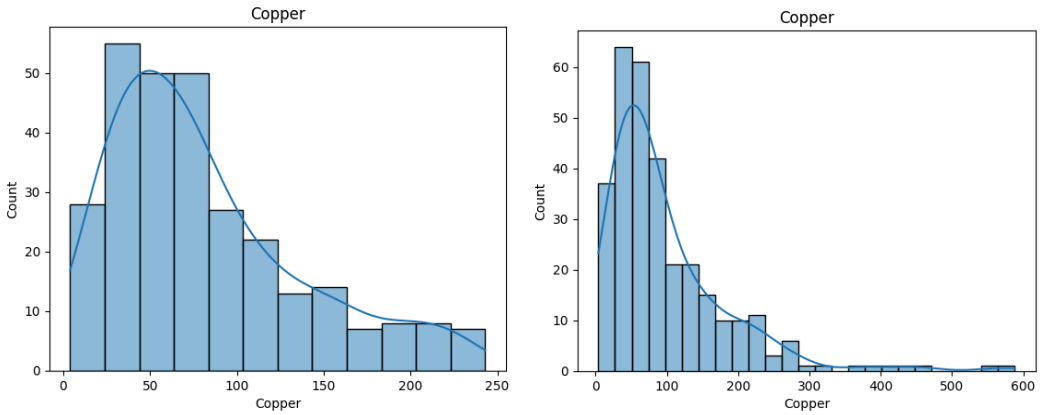
Passa el mateix amb la variable 'Age' que amb la variable anterior. Veiem també com la punta a la gràfica de la variable tractada (dreta) és més plana que la de la gráfica que conté la variable sense tractar (esquerra) i, deduïm que això és degut a la normalització que ha patit la variable. Era una variable amb una distribució molt bona i per això veiem que les modificacions que ha rebut son pràcticament nules.



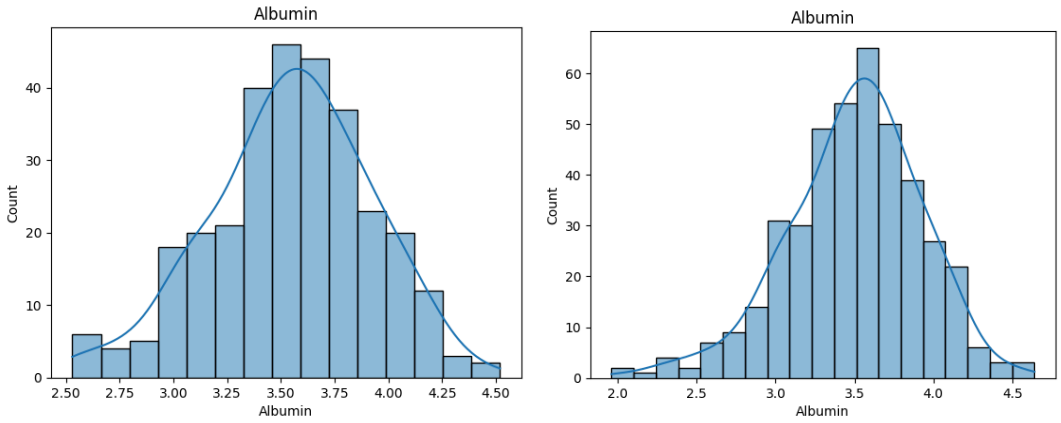
La primera variable que sí que veiem modificada es la variable ‘Bilirubin’. Veiem com, tot i que manté la tendència exponencial inversa, la variable tractada (esquerra) no té uns primers valors tan exageradament alts com tenia abans comparat amb la resta de valors. A més observem com els outliers han desaparegut, on veiem que a la variable tractada els valors només arriben fins a 7 (aproximadament) mentre que a la gráfica que representa la variable no tractada (dreta) la cua d’outliers és molt gran, superant inclús valors com el 25 (un valor molt alt tenint en compte que els valors es concentren al voltant del 1~3)



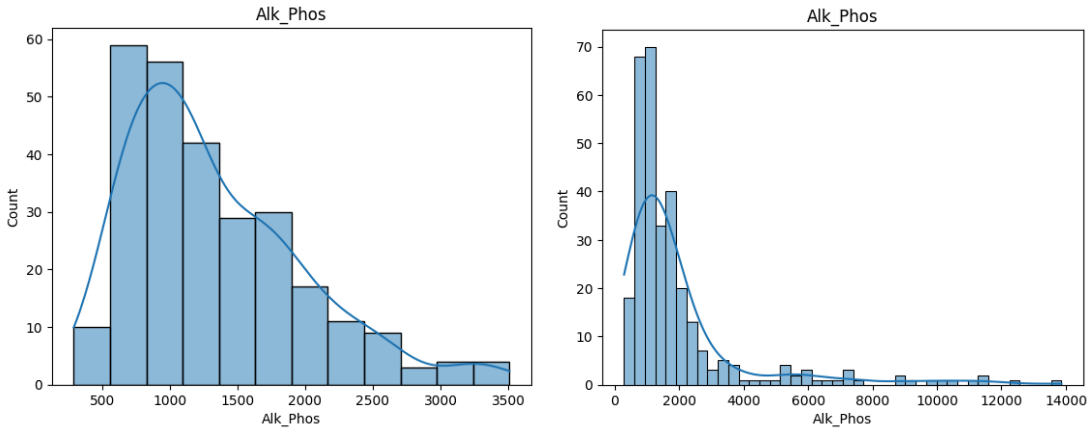
De manera molt similar a la variable anterior, veiem com la variable ‘Cholesterol’ ara és molt més plana. Segueix una distribució semblant a una normal i es pot veure clarament com els seus outliers han sigut eliminats: els valors de la gràfica de les dades tractades (esquerra) no supera els 600 mentre que en la gràfica de la variable sense modificar (esquerra) els valors arriben fins a 1750. La corba, a més, està molt més centrada després del tractament de dades, fet que implica que el biaix que tenia a la dreta és molt més petit que abans.



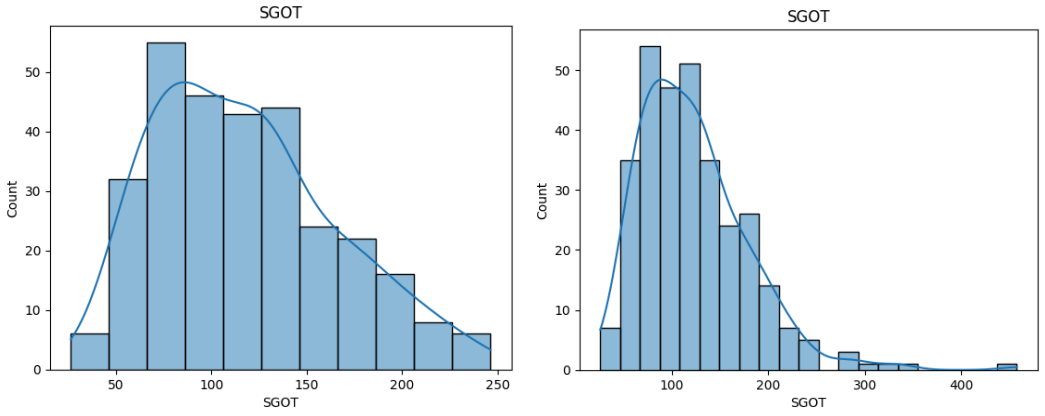
Veiem en la variable ‘Copper’ com, en la gràfica que mostra les dades processades (esquerra) la corba és molt menys escarpada que a la de les dades sense tractar (dreta). Veiem també com els outliers han sigut tractats i ja no hi són, tot i que el biaix que tenen les dades a la dreta no s’ha corregit gairebé res i això fa que la corba no estigui centrada.



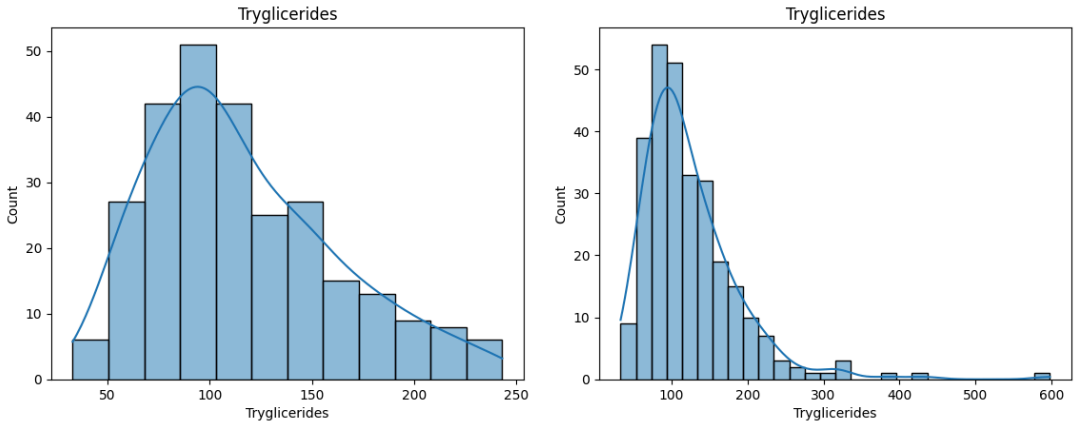
En el cas de la variable ‘Albumin’ veiem com, a simple vista, no ha rebut grans modificacions. veiem però, com a les dades no tractades (dreta) existeix un petit biaix a l’esquerra que, despues del tractament de les dades (gràfica de l’esquerra) ja no hi és. A més, veiem com, un altre cop, la normalització ha fet que la corba sigui lleugerament més plana, i veiem com ara cap valor dels valors centrals destaca de manera clara sobre la resta.



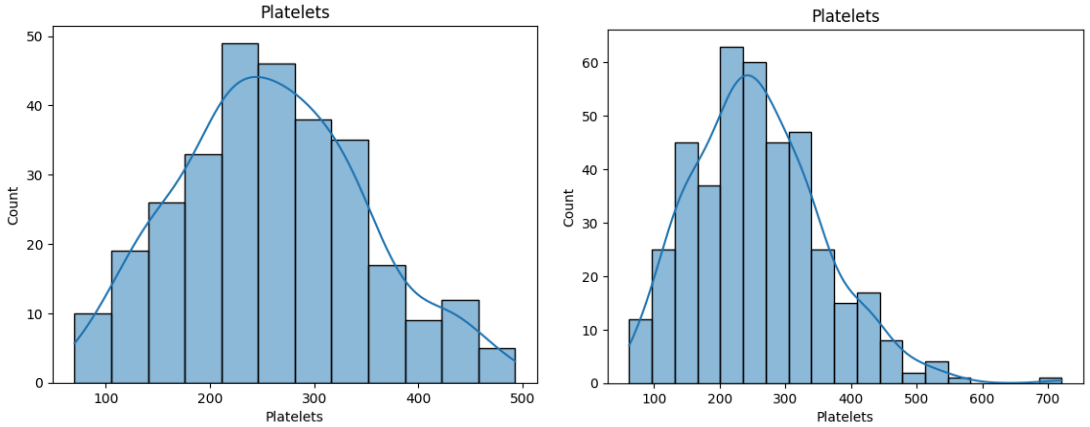
La següent variable és ‘Alk_Phos’. Veiem com, en aquest cas, les dades processades (esquerra) sí que tenen una gran diferència amb les dades sense tractar (dreta). Les dades no processades ens feien intuir una corba exponencial negativa, però després de fer tot el tractament, veiem com la distribució que tenim és una espècie de corba normal amb un biaix a la dreta que la desplaça cap a l’esquerra respecte el centre. Veiem també com tots els outliers han sigut eliminats respecte a les dades inicials, fet que juntament amb la normalització de les dades ha dut a terme aquesta millora significativa en les dades.



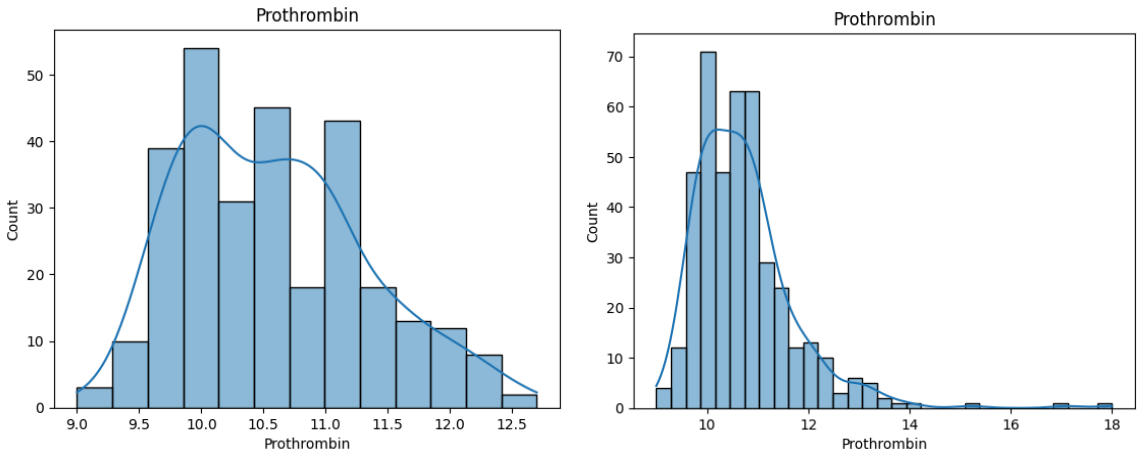
El canvi més destacable que ha rebut la variable ‘SGOT’ ha sigut la eliminació dels outliers superiors que tenien les dades base (dreta). Com podem observar a la gràfica on es veuen les dades tractades (esquerra), el biaix que tenien les dades a la dreta s’ha vist reduït en gran part. Podem veure també com la corba és molt més plana degut a la normalització que hem dut a terme, que també escala les dades.



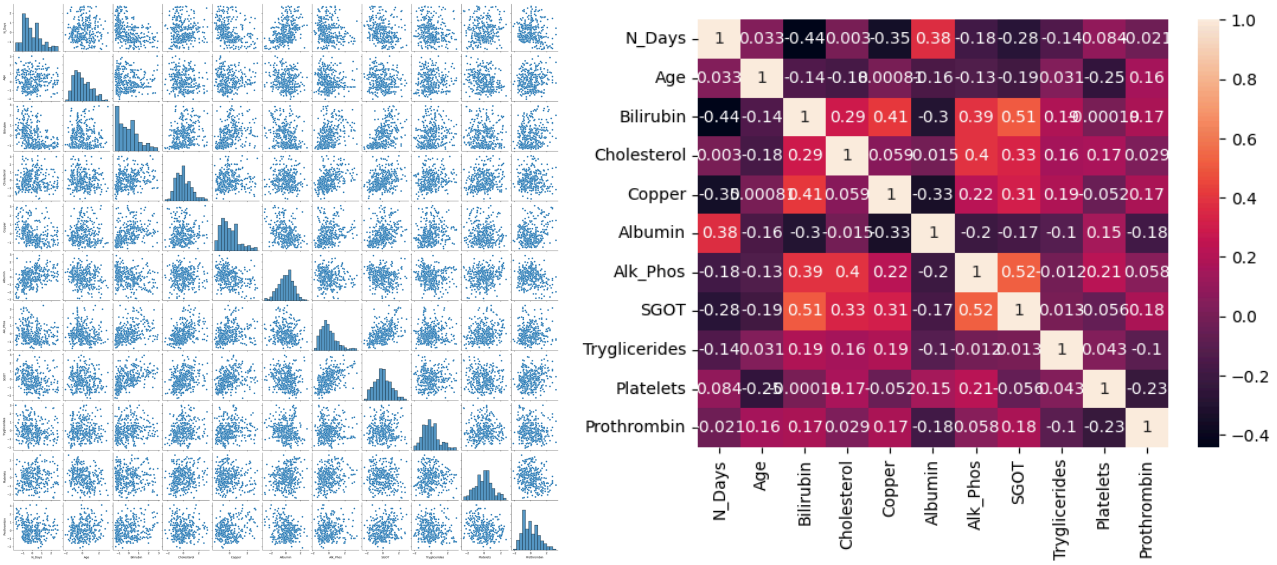
En la variable 'Trygliceride' podem observar també un gran canvi entre les dades processades (esquerra) i les dades sense processar (dreta). Veiem com els outliers han siguit eliminats com en totes les anteriors variables, i com això ha arreglat, en certa manera, el biaix que tenien les dades a la dreta (tot i que encara es pot veure una mica de biaix). El fet d'escalar les dades amb la normalització ha fet que la corba sigui molt menys escarpada. Els valors centrals de la mateixa també estan molt més nivellats.



En la variable 'Platelets' podem observar com els pocs outliers que tenien les dades sense processar (dreta) han sigut eliminats i això ha arreglat el desplaçament que tenien les dades cap a l'esquerra. Veiem també com en les dades processades (esquerra) la corba és molt menys escarpada i com totes les dades es mantenen dins de la mateixa, fet que ens deixa veure que les dades segueixen una distribució normalitzada.

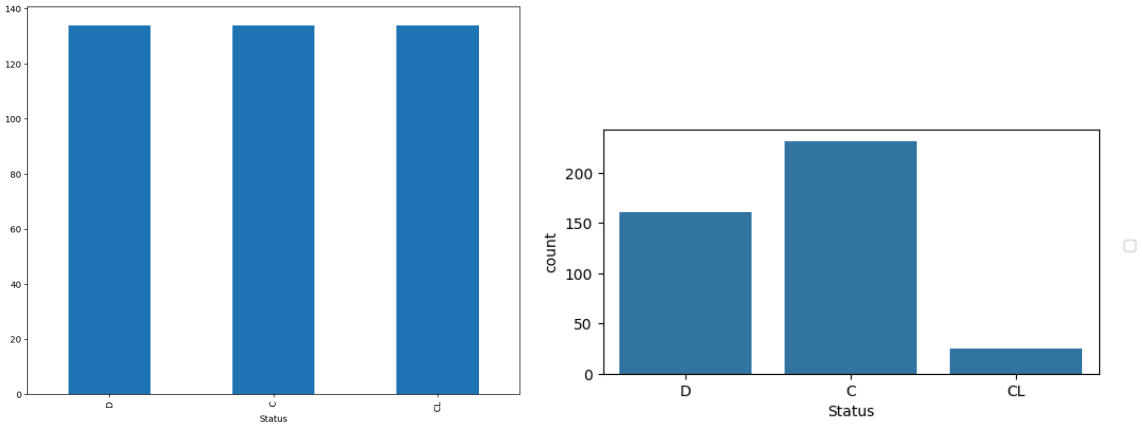


Per acabar amb les variables numèriques, veiem la variable 'Prothrombin'. Veiem que el principal canvi a les dades processades (esquerra) és l'eliminació dels outliers que podem apreciar a les dades sense tractar (dreta). Veiem també com els valors han sigut escalats i normalitzats, fet que fa que la corba sigui molt més plana. Observem també com el biaix que tenia a la dreta s'ha reduït en alta mesura, presumiblement degut a l'eliminació de outliers.



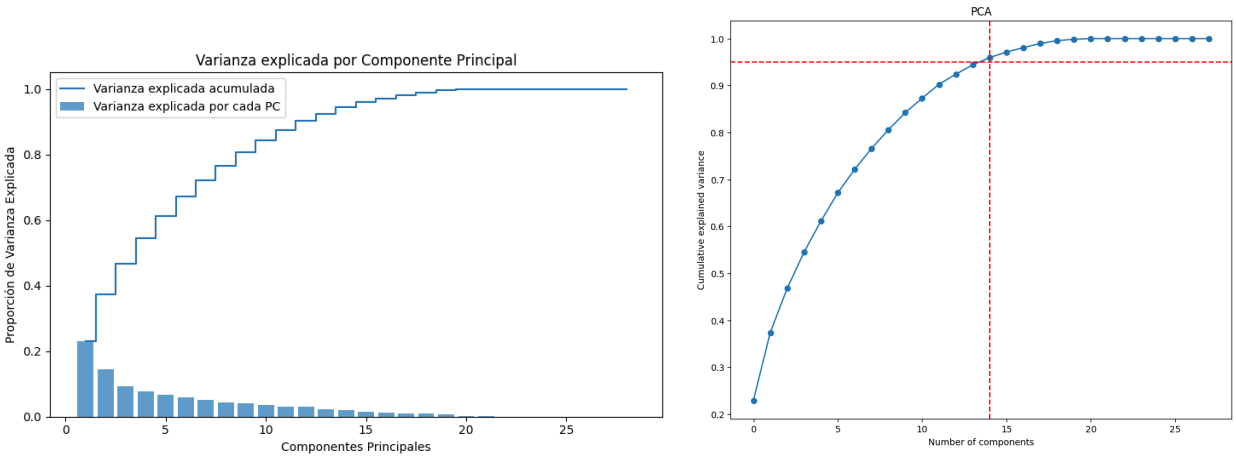
Fem també un pairplot (esquerra) i una matriu de correlació (dreta) de totes les variables numèriques amb l'objectiu de buscar relacions entre elles i veure si podem eliminar alguna. Com podem observar, cap variable está relacionada amb cap altre i per tant cap variable es realment prescindible pel fet de ser redundant.

També analitzarem els canvis que ha rebut la variable resposta després de balancejar-la:



Veiem com, un cop balancejada, la classe 'Status' pren exactament la mateixa quantitat de valors per a cadascuna dels seus possibles valors. Això fara que el model que entrenem sigui capaç de predir amb millor exactitud qualsevol de les 3 respostes possibles.

Per últim, farem un anàlisi de la dimensionalitat per veure si alguna variable es poc representativa i es pot eliminar. Per fer això farem un PCA (Principal Component Analysis) i veurem com de representatives són les variables.



El primer gràfic (esquerra) ensenya la variança explicada de cada element del nostre dataset sobre la cirrosi hepàtica. El segon Mostra també això però en una corba on cada punt representa cada nou component principal. Veiem com, tot i la zona recta plana que veiem al final, el 95% del dataset (indicat per la línia vermella de punts) es veu explicat per 14 components. El 80%, que sería el mínimo necessari, és veu explicat per unes 8/9 dimensions. És per això que, donat que la reducció de dimensionalitat no facilitaria l'enteniment de les dades degut a que reduir les dimensions ens genera una base de dades de dimensions no interpretables d'igual manera. Per aquest motiu no reduirem la dimensionalitat de la base de dades.

6. Modelització: KNN

KNN es un algorisme que classifica un nou punt en les dades segons els seus K veïns (elements més propers a ell). La motivació d'aquest model és la seva fàcil comprensió. A més, al tenir un dataset petit, no correm el risc de que es torni computacionalment molt costós. La mètrica que modificarem en aquest model es la K, el nombre de veïns que té en compte per assignar el valor. Per fer el primer model utilitzem una k arbitraria (en aquest cas utilitzem k = 3):

Python

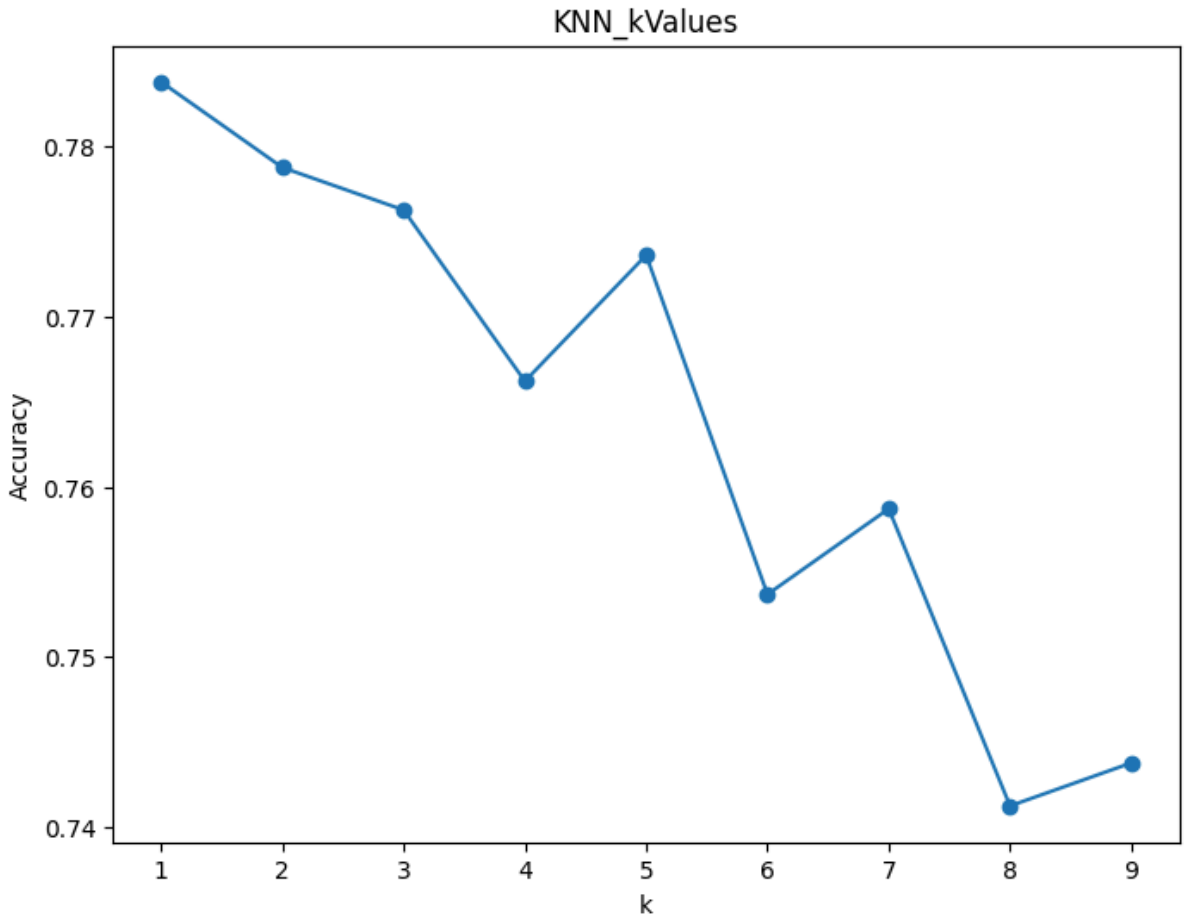
```
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)
predictKNN = knn_model.predict(X_train)

accuracy_KNN = accuracy_score(y_train, predictKNN)
conf_matrix_KNN = confusion_matrix(y_train, predictKNN)

print(f"Precisión del modelo: {round((accuracy_KNN*100) , 3)} %")
print("Matriz de Confusión:")
print(conf_matrix_KNN)

print(classification_report(y_train, predictKNN))
```

Al dur a terme aquest entrenament, els resultats obtinguts sobre la partició de train són una accuracy del 88.56% i una F1-Score de 0.84, 0.94 i 0.87 per a cada predicció possible. Per intentar millorar aquests resultats, farem una iteració provant diferents valors per a K (desde k=1 fins a k=9). Per fer això farem servir el següent codi:

Python

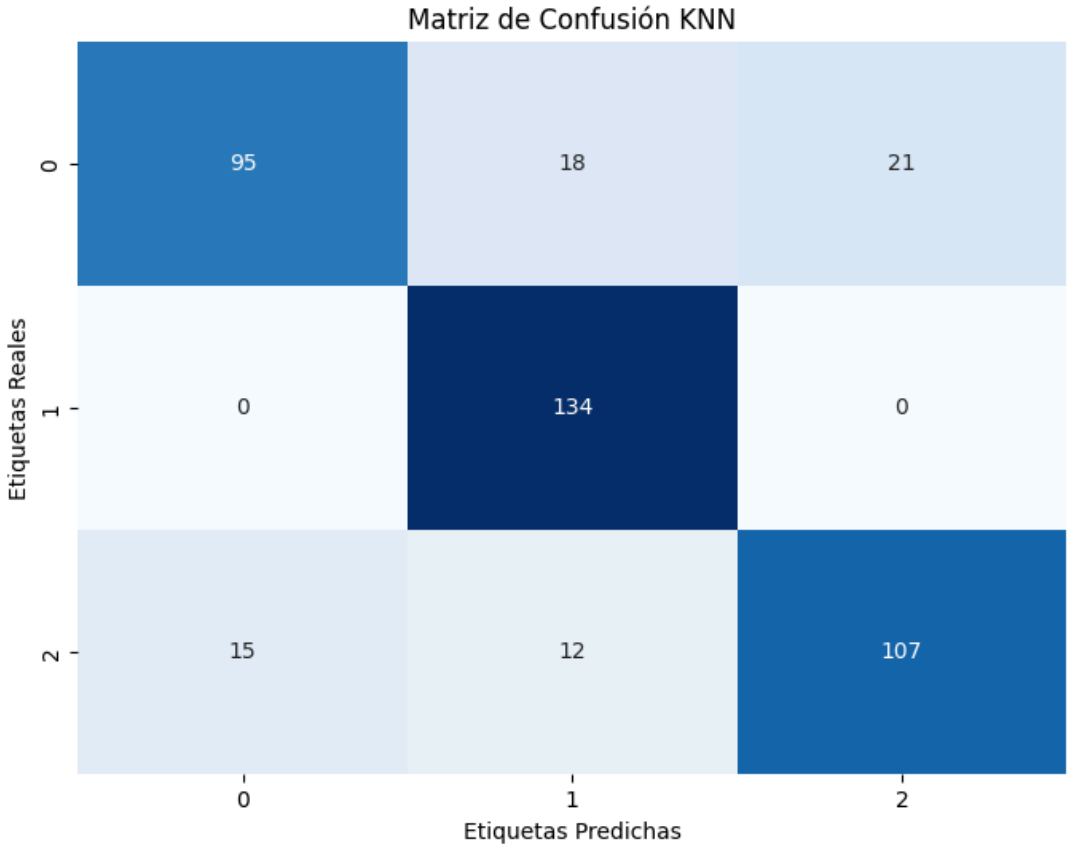
```
k_values = list(range(1, 10))
mean_accuracy_scores = []
for k in k_values:
  fold_accuracy_scores = []
  kf = KFold(n_splits=5, shuffle=True, random_state=69)
  for train_index, val_index in kf.split(X_train):
              X_train_fold,  X_val_fold  =  X_train.iloc[train_index],
X_train.iloc[val_index]
              y_train_fold,  y_val_fold  =  y_train.iloc[train_index],
y_train.iloc[val_index]
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train_fold, y_train_fold)
    prediction = knn_model.predict(X_val_fold)
    acc = accuracy_score(y_val_fold, prediction)
    fold_accuracy_scores.append(acc)
```

Dividim el train en 5 divisions més i iterem utilitzant de manera combinatòria 4 d'aquestes com a un nou train i la restant com a validation (5 cops per valor de K ja que tenim 5 divisions del dataframe). Un cop realitzat aquest experiment, grafiquem els resultats i veiem el següent:



Com podem veure, els millors valors obtinguts d'accuracy els tenen els valors 1, 2 i 3 de K. Podem veure com el següent millor valor es k=5, i per tant ens quedarem amb aquest valor. Veiem també com els números parells tenen un rendiment significativament pitjor que els imparells i com a mesura que augmentem el valor de k l'accuracy disminueix.

Executem, doncs el model amb k=5 i obtenim uns valors una mica pitjors (0.84 accuracy, 0.78, 0.90 i 0.82 F1 score). Decidim quedar-nos amb k=5 amb la idea d'evitar l'overfit que té k=3. No vol dir això que k=5 no estigui overfitted, només vol dir que segurament ho estigui menys que k=3. Buscant un millor resultat en un cas real, doncs, agafem k=5. El model de KNN que hem escollit prediu la següent matriu de confusió:



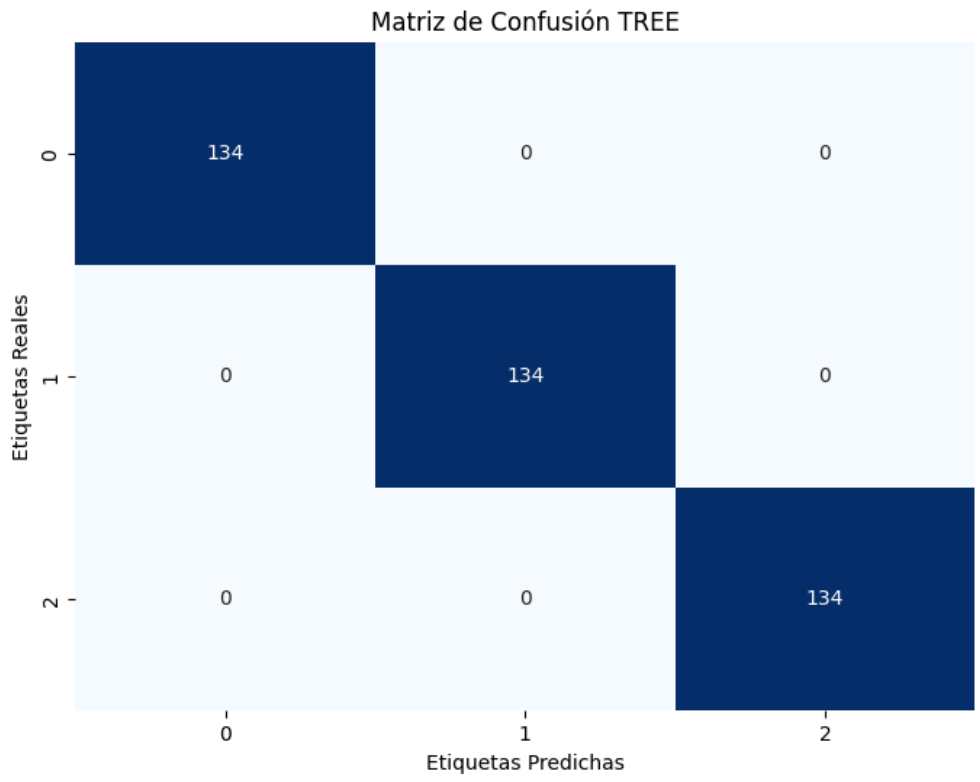
	precision	recall	f1-score	support
C	0.86	0.71	0.78	134
CL	0.82	1.00	0.90	134
D	0.84	0.80	0.82	134
accuracy			0.84	402
macro avg	0.84	0.84	0.83	402
weighted avg	0.84	0.84	0.83	402

7. Modelització: Arbre de decisió

Els arbres de decisió son un tipus de model interpretable que genera relacions causa-efecte per prendre decisions. L'inconvenient es que tendeixen a fer overfit a les dades donades al dataframe del train. La motivació és, doncs la seva fàcil interpretabilitat. Per intentar gestionar l'overfitting, ajustarem la profunditat màxima, el minim de samples per split i el mínim de samples per fulla. Provem primerament de fer un arbre de decisions sense establir cap paràmetre:

```python
decision_tree = DecisionTreeClassifier(random_state=69)
decision_tree.fit(X_train, y_train)

predictTREE = decision_tree.predict(X_train)
accuracyTREE = accuracy_score(y_train, predictTREE)

conf_matrix_TREE = confusion_matrix(y_train, predictTREE)
print(f"Precisión del modelo: {round((accuracyTREE*100), 3)} %")
```

Com habiem predit, l'arbre sense ajustar mètriques fa overfit, de fet fa un overfit exagerat, ja que la precisió obtinguda d'aquest model és del 100%.



Per evitar aquest overfit, buscarem un model millor fent cross validation amb una funció anomenada 'GridSearchCV'. A aquesta funció li passem un diccionari amb els diferents valors que volem provar per a cada mètrica i fa la combinatoria dels diferents valors. Un cop realitza el càlcul, ens retorna els valors que millors resultats han donat.

Python

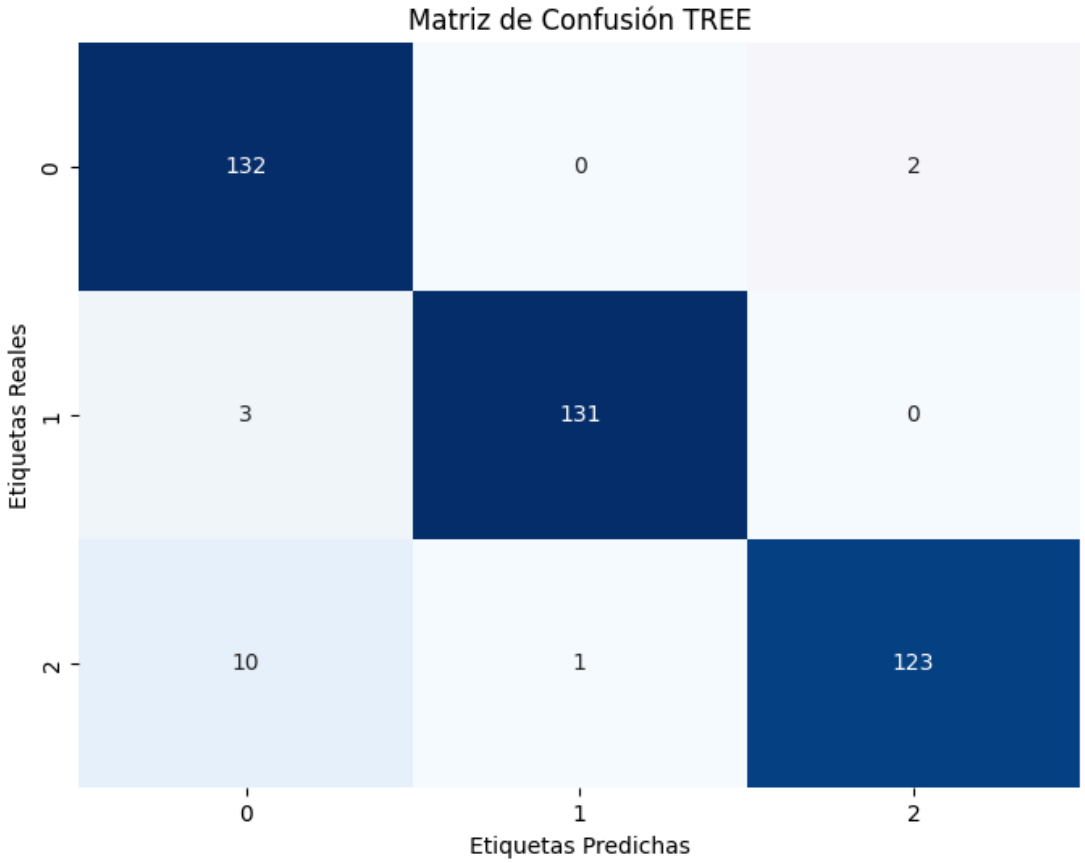
```
param_grid_TREE = {
  'max_depth': [None, 1, 3, 5, 7, 9, 13, 15],
  'min_samples_split': [1, 2, 5, 7, 10, 15],
  'min_samples_leaf': [1, 2, 4, 6, 8, 10]
}

decision_tree = DecisionTreeClassifier(random_state=69)
grid_search=GridSearchCV(estimator=decision_tree,
param_grid=param_grid_TREE, cv=5)
grid_search.fit(X_train, y_train)

best_params_TREE = grid_search.best_params_
print("Mejores hiperparámetros encontrados:")
print(best_params_TREE)
print('best score')
print(f"{grid_search.best_score_}")
```

Els valors obtinguts han sigut {'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split': 2} i els ha obtingut un model amb una score de 0.7861. Ara procedirem a provar el model en la nostra partició de train, i veure si és millor que l'anterior:

Python

```
best_decision_tree=DecisionTreeClassifier(**best_params_TREE,
random_state=69)
fit_TREE = best_decision_tree.fit(X_train, y_train)

predictTREE = fit_TREE.predict(X_train)
accuracyTREE = accuracy_score(y_train, predictTREE)

conf_matrix_TREE = confusion_matrix(y_train, predictTREE)
```

Un cop fem això, obtenim una precisió del 96%, que segurament també estigui overfitted, però clarament esta menys overfitted que el primer model que hem fet amb un decision tree. A continuació veiem la matriu de confusió i el classification report del model final d'arbre de decisions.



	precision	recall	f1-score	support
C	0.91	0.99	0.95	134
CL	0.99	0.98	0.98	134
D	0.98	0.92	0.95	134
accuracy			0.96	402
macro avg	0.96	0.96	0.96	402
weighted avg	0.96	0.96	0.96	402

8. Modelització: SVM

El SVM o Support Vector Machine tenen una interpretabilitat nula, ja que busquen el millor hiperplà que separa les classes en l'espai del dataset. La motivació per utilitzar aquest tipus de model es el fet de que són molt útils en espais de altes dimensions (com el nostre). A més, no resulta computacionalment costós ja que el dataset només consta d'unes 400 línies. Les mètriques que modificarem buscant el millor SVM possible són 'C', el kernel y el gamma. Abans d'això, però, provarem de fer un model sense establir cap hyperparametre prèviament.

Python

```
svm_model = SVC(random_state=69)
svm_model.fit(X_train, y_train)

predictSVM = svm_model.predict(X_train)
accuracySVM = accuracy_score(y_train, predictSVM)
```

Executar això ens retorna una precisió del 94.5% i una F1_score de 0.92, 9.99 i 0.92. Tot i que aquests valors són molt alts, intentarem millorar-los fent cross validation, un altre cop amb la funció 'GridSearchCV' un altre cop.

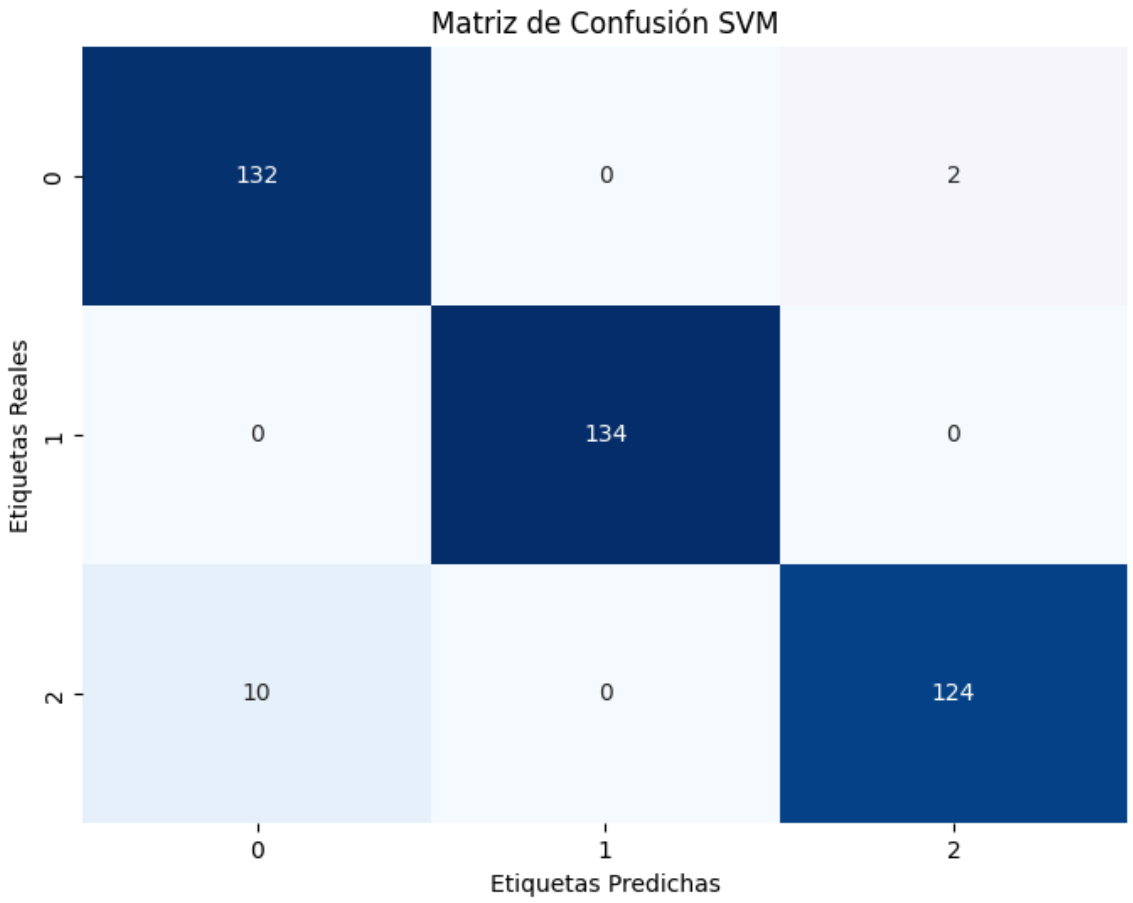
Python

```
param_grid_SVM = {
  'C' : [0.1, 1, 10, 100],
  'gamma' : ['scale', 'auto'],
  'kernel' : ['linear', 'rbf', 'poly']
}

grid_search=GridSearchCV(estimator=svm_model,
param_grid=param_grid_SVM, cv=5)
grid_search.fit(X_train, y_train)

best_params_SVM = grid_search.best_params_
print("Mejores hiperparámetros encontrados:")
print(best_params_SVM)
print('best score')
print(f"{grid_search.best_score_}")
```

Els valors obtinguts han sigut {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'} i els ha obtingut un model amb una score de 0.8905. Ara procedirem a provar el model en la nostra partició de train, i veure si és millor que l'anterior:

```python
best_svm_model = SVC(**best_params_SVM, random_state=69)
fit_SVM = best_svm_model.fit(X_train, y_train)

predictSVM = fit_SVM.predict(X_train)
accuracySVM = accuracy_score(y_train, predictSVM)

print(f"Accuracy del SVM con los mejores hyperparametros: ,
{round((accuracySVM*100), 3)} %")

conf_matrix_SVM = confusion_matrix(y_train, predictSVM)
print("Matriz de Confusión:")
print(conf_matrix_SVM)

print(classification_report(y_train, predictSVM))
```

Aquest model ens retorna, sobre el nostre train, una precisió del 97% i uns F1_scores que es troben tots per sobre de 0.96 (0.97, 1 i 0.96). El model esta overfitted, pero creiem que pot arribar a predir bé els valors desitjats (overfitted per útil). A continuació es veuen la matriu de confusió i el classification report del model final de SVM:



	precision	recall	f1-score	support
C	0.93	0.99	0.96	134
CL	1.00	1.00	1.00	134
D	0.98	0.93	0.95	134
accuracy			0.97	402
macro avg	0.97	0.97	0.97	402
weighted avg	0.97	0.97	0.97	402

9. Modelització: EBM

El EBM o Explainable Boosting Machine es un tipus de model que es caracteritza per tenir una alta interpretabilitat. Combina múltiples arbres de decisió per millorar la precisió decisiva. D'aquesta manera, es capaç de donar unes prediccions molt precises. La motivació és, doncs, el fet de que són interpretables i, a més, tenen una capacitat predictiva molt bona. Les mètriques que modificarem en busca del millor model possible són el learning rate, el màxim de bins i d'interaccions, el número d'interaccions i el mínim de mostres per fulla.

En aquest cas, buscarem directament el millor model possible utilitzant el 'Grid Search' per escollir els millors paràmetres:

Python

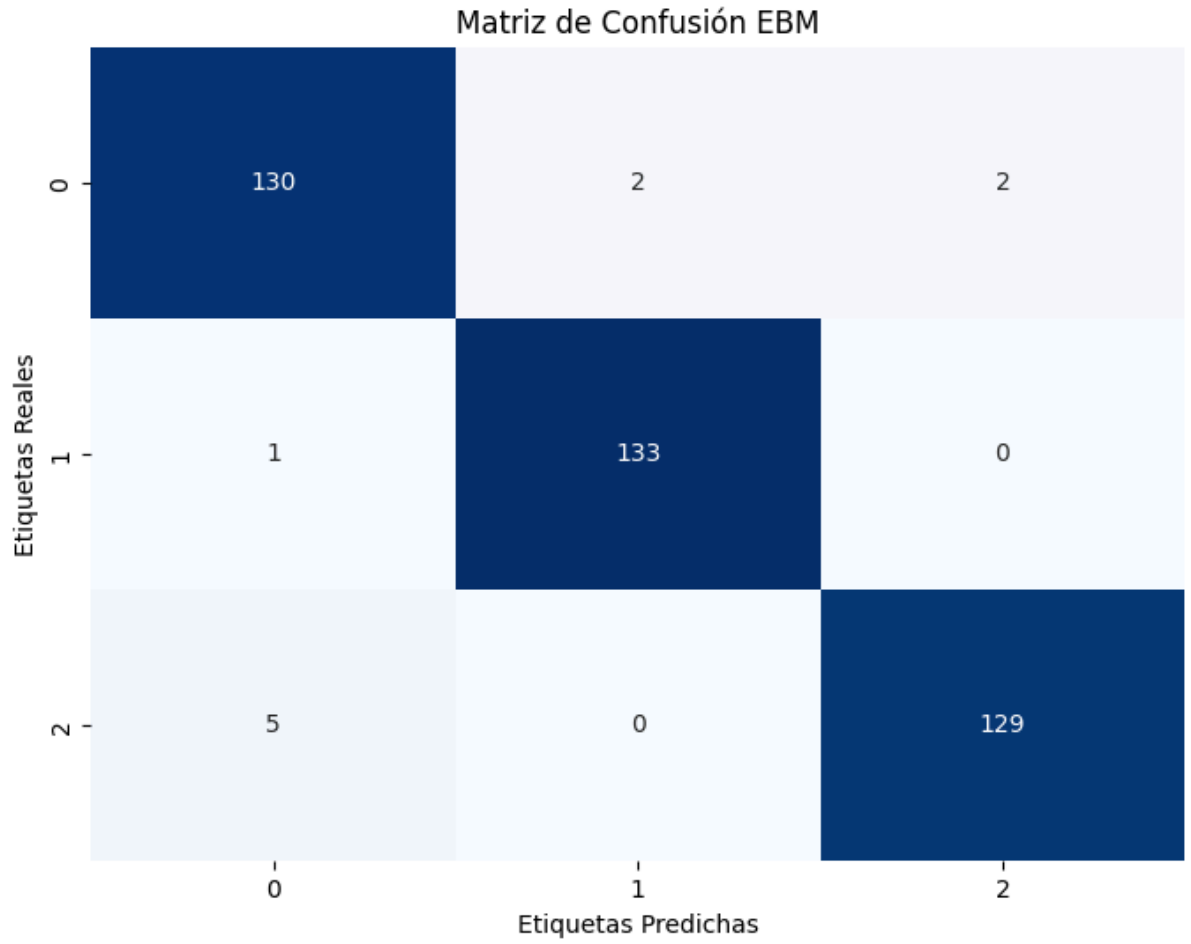
```
ebm = ExplainableBoostingClassifier(random_state=69)

param_grid_EBM = {
  'learning_rate': [0.01, 0.1, 0.5, 1],
  'max_bins': [128, 256, 512],
  'max_interaction_bins': [8, 16, 32, 64],
  'interactions': [0, 2, 5],
  'min_samples_leaf': [5, 10]
}

grid_search_EBM = GridSearchCV(
  ebm,
  param_grid_EBM,
  cv=5,
  scoring=make_scorer(accuracy_score)
)

grid_search_EBM.fit(X_train, y_train)
```

Un cop realitzem aquesta búsqueda, els millors paràmetres escollits han sigut {'interactions': 0, 'learning_rate': 0.5, 'max_bins': 512, 'max_interaction_bins': 8, 'min_samples_leaf': 5} donats per un model de EBM que ha donat una score de 0.825. Cal destacar que aquest model ha trigat significativament més en entrenarse (uns 5 minuts, comparat amb els 30 segons - 1 minut que han trigat la resta). Un cop tenim el model, el provem en el train i veiem que tal ho fa:

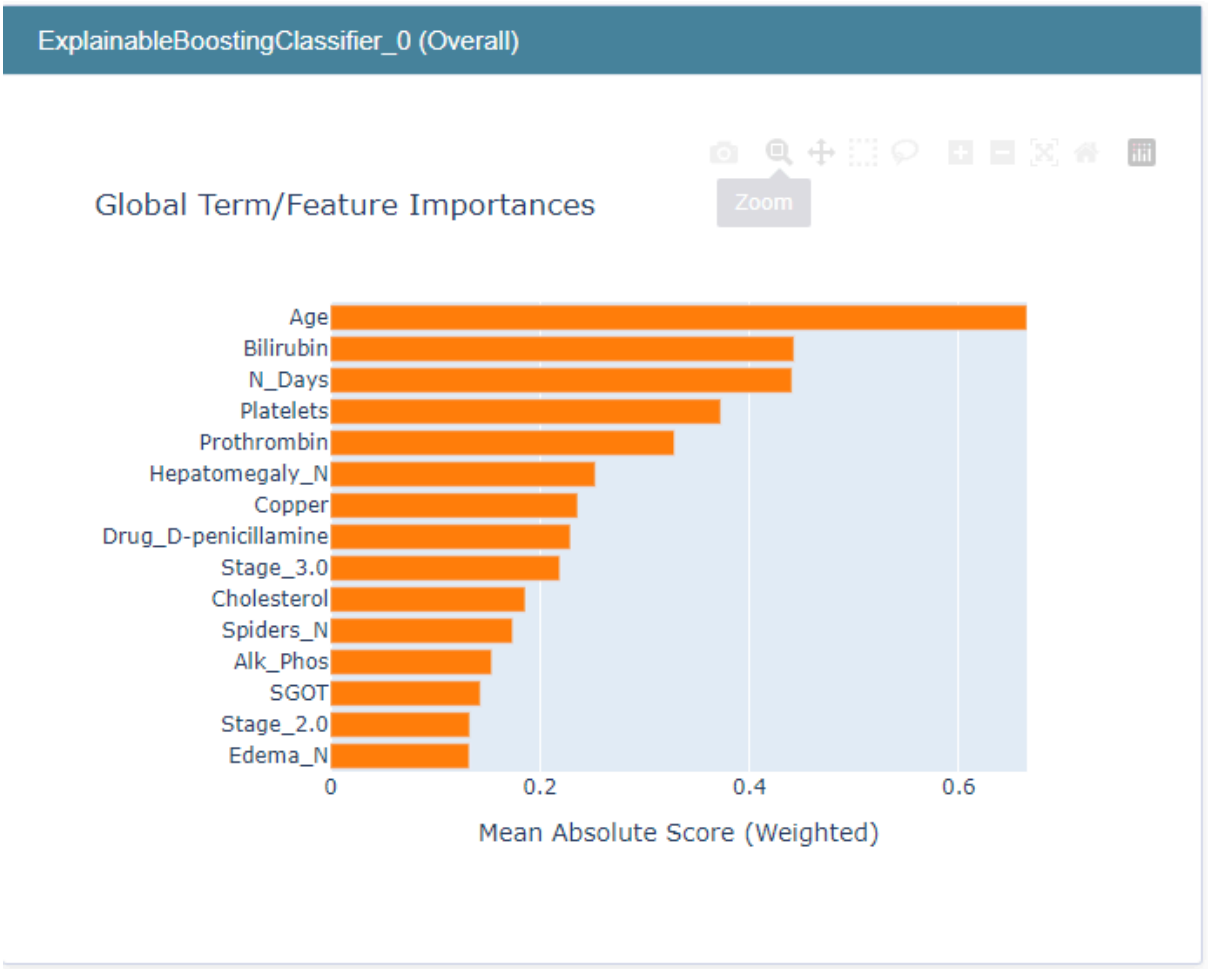
Python

```
best_ebm=ExplainableBoostingClassifier(random_state=69,
**best_params_EBM)
fit_EBM = best_ebm.fit(X_train, y_train)
predictEBM = fit_EBM.predict(X_train)
accuracyEBM = accuracy_score(y_train, predictEBM)
```

Els resultats que ha obtingut aquest model han sigut un 97.5% de precisió i uns F1-score de 0.96, 0.99 i 0.97. A continuació veiem la matriu de confusió, el classification report i el gràfic de la importància de les variables per al model:



	precision	recall	f1-score	support
C	0.96	0.97	0.96	134
CL	0.99	0.99	0.99	134
D	0.98	0.96	0.97	134
accuracy			0.98	402
macro avg	0.98	0.98	0.98	402
weighted avg	0.98	0.98	0.98	402



Com podem veure al gràfic mostrat, la variable més important és l'edat, seguida per la bilirubina i pel número de dies hospitalitzat. Altres variables a destacar podrien ser 'Platelets' o 'Prothrombin'.

10. Selecció de model

Per escollir el millor model, tindrem en compte l'score, el F1-score i la precisió. Ho farem utilitzant unes taules comparatives, la primera de totes la que compara els scores obtinguts al grid search (KNN no te score del grid search doncs no tindrem en compte el seu score a l'hora de decidir)

MODEL	**SCORE**
KNN	∅
Decision Tree	0.7861
SVM	0.8905
EBM	0.825

.

Com veiem a la taula, el millor score es l'obtingut per el model d'SVM, seguit pel model EBM i pel Decision Tree. Caldrà veure la resta de mètriques abans de fer una decisió. La següent taula representa els valors de F1 score per a cada variable resposta:

MODEL	**F1_Score**		
	C	CL	D
KNN	0.78	0.90	0.82
Decision Tree	0.95	0.98	0.95
SVM	0.96	1	0.95
EBM	0.96	0.99	0.97

Els dos millors models semblen ser el SVM i el EBM. Veiem com deixen les coses els valors de precisió:

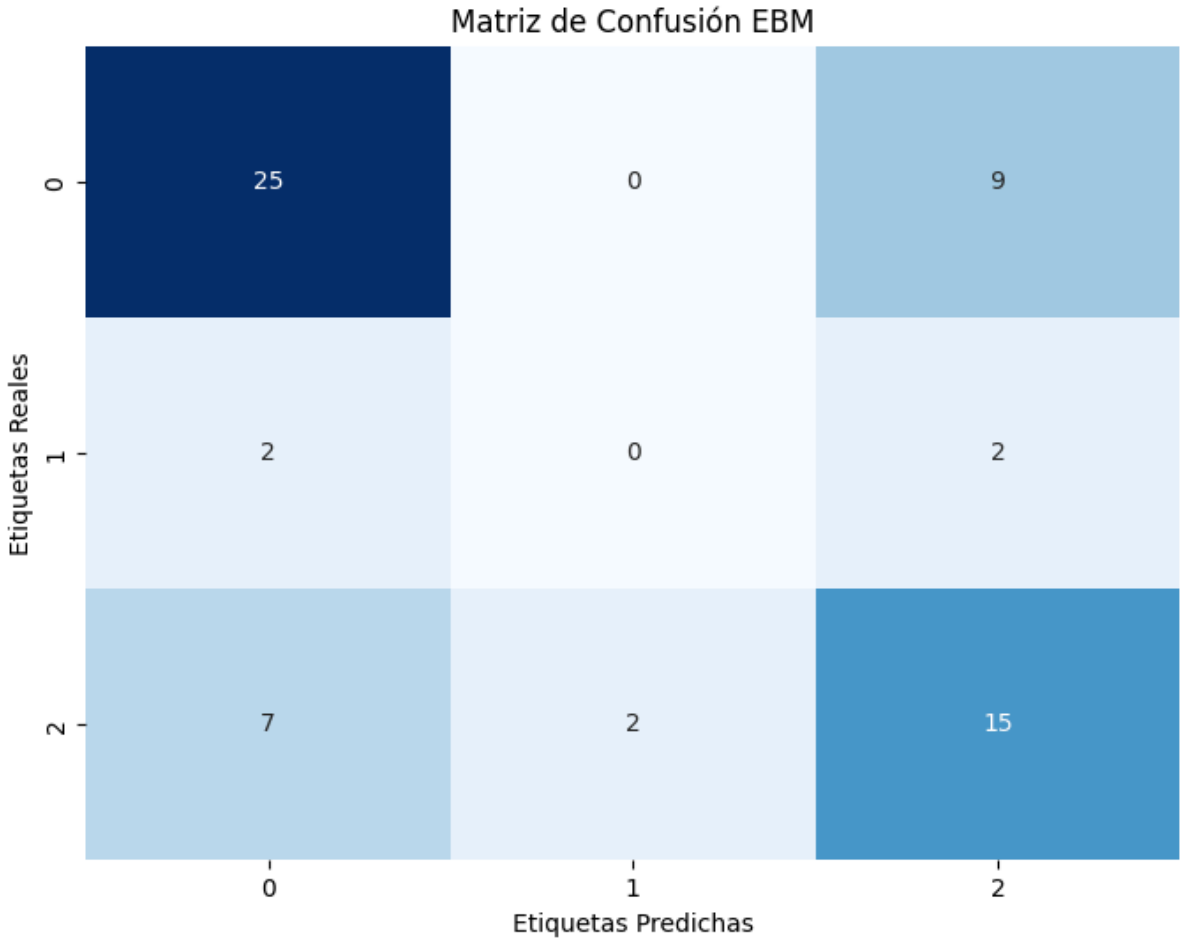
MODEL	**PRECISIÓ**
KNN	83.52%
Decision Tree	96.02%
SVM	97.015%
EBM	97.512

Amb els resultats de totes 3 taules obtenim la idea de que SVM i EBM són els millors models a l'hora de predir. El problema que té SVM és que no es interpretable, mentre que el problema que té el EBM es que es car d'entrenar computacionalment. Tot i així, ens decantem per escollir el EBM degut a la interpretabilitat que té. Ara procedirem a probar el model triat en la partició de test:

Python

```
predictEBM_test = fit_EBM.predict(X_test)
accuracyEBM_test = accuracy_score(y_test, predictEBM_test)
print(f"Accuracy del EBM con los mejores hyperparametros: ,
{round((accuracyEBM_test*100), 3)}%")

conf_matrix_EBM_test = confusion_matrix(y_test, predictEBM_test)
print("Matriz de Confusión:")
print(conf_matrix_EBM_test)

print(classification_report(y_test, predictEBM_test))
```

Un cop el provem en el test obtenim una precisió d'un 64,5% i uns F1-Score de 0.74, 0 i 0.60. A continuació estudiem la matriu de confusió i el classification report:



Com podem veure en la matriu de confusió, el principal problema del model es que gairebé no prediu cap transplant de fetge (CL). Tot i així obté pocs falsos positius en predir la gent que morirà (‘C’) [‘C’ = 0, ‘CL’ = 1, ‘D’ = 2]. A continuació veiem el classification report del model i, per últim, la model card.

	precision	recall	f1-score	support
C	0.74	0.74	0.74	34
CL	0.00	0.00	0.00	4
D	0.58	0.62	0.60	24
accuracy			0.65	62
macro avg	0.44	0.45	0.45	62
weighted avg	0.63	0.65	0.64	62

11. Model Card

Model Card for Cirrhosis Patient Survival Prediction Dataset

Model Details

Overview

Aquest model prediu l'estat final d'un pacient de cirrosi hepàtica basant-se en les dades clíniques d'altres pacients.
Aquest model està entrenat amb un algorisme de EBM. És un algorisme paramètric i que crea diversos arbres de decisions i els combina per predir una resposta. Els hiperparàmetres que s'han hagut de controlar són: interaccions =0 , ratio d'aprenentatge = 0.5, numero màxim de bins = 512, màximes interaccions entre bins=8 i mínim de mostres per fulles=5.

Version

name: 7e62d32e-c4c5-40ce-9450-92ca2f41e17d
date: 2023-12-28

Owners

César Elías Mejía Rota (IAA Student), cesar.elias.mejia@estudiantat.upc.edu

References

- https://archive.ics.uci.edu/dataset/878/cirrhosis+patient+survival+prediction+dataset-1

Considerations

Intended Users

- Professors de IAA

Use Cases

- L'intenció d'ús d'aquest model és evaluar els coneixements obtinguts en l'assignatura de IAA impartida en el GIA de la UPC creant un model que predigui l'estat final d'un pacient de cirrosi hepàtica segons dades clíniques d'altres pacients. Aquest model no té cap intenció en ser usat per a realitzar cap diagnòstic real o negoci. L'usuari que ha de fer servir aquest model és el professorat de l'assignatura.

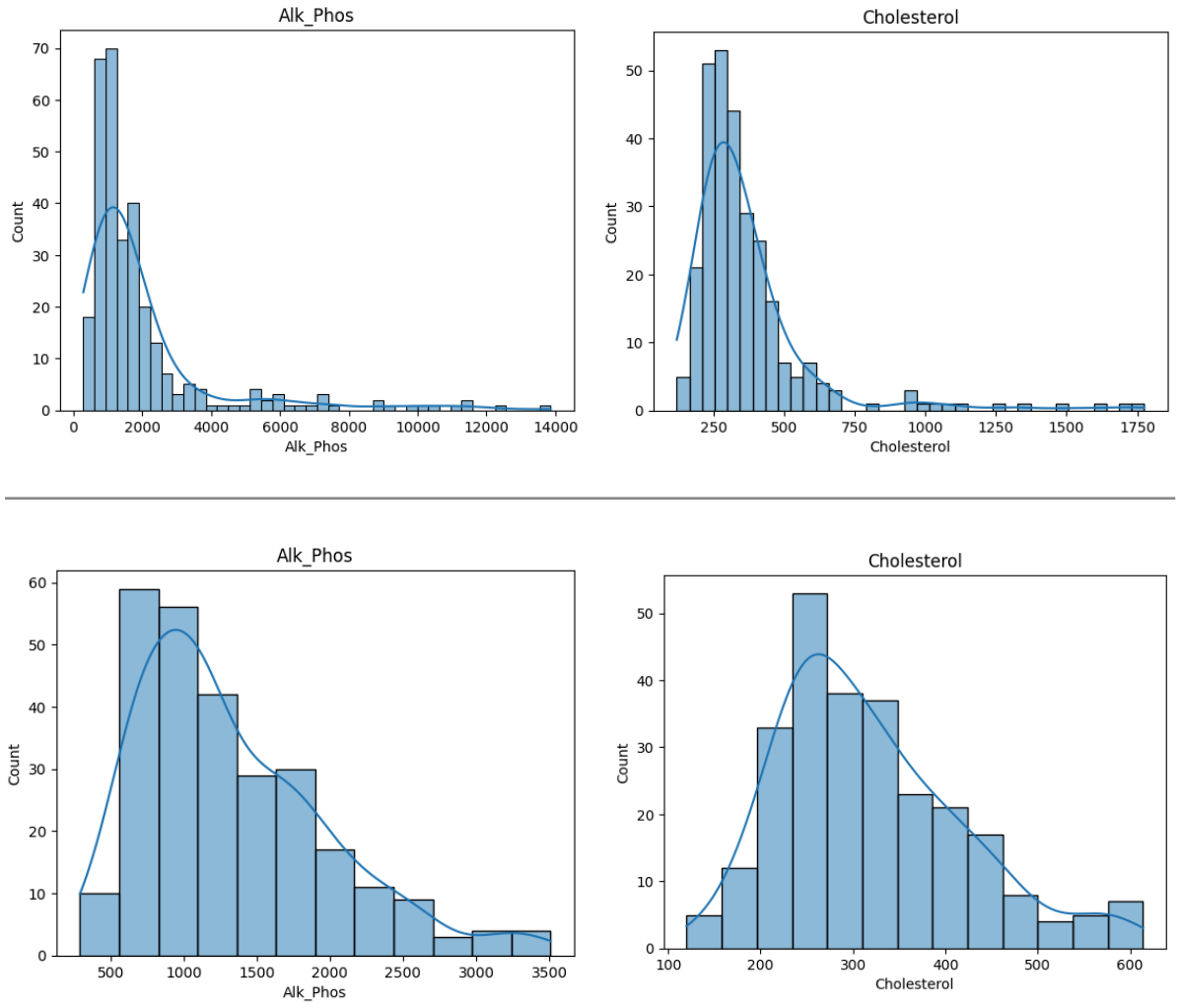
Limitacions

- El model té un rendiment molt pobre a l'hora de predir pacients que reben un transplant de fetge.
- El model triga significativament més temps a ser entrenat que els seus models germans

Ethical Considerations

- El model pot arribar a no predir cap transplantament, i té una precisió relativament baixa. Mitigation Stategy: Mai prendre com a resultat final d'un estudi els resultats obtinguts pel model.

Datasets



Quantitative Analysis

Confusion matrix on X_train and X_test