**Normalidad de los datos**

Prueba de shapiro- wilk: Hace una prueba t de student entre la distribución que queremos evaluar y una distribución normal genérica. Solo valores numéricos.

 the p-value > 0.05 implying that the distribution of the data are not significantly different from normal distribution. In other words, we can assume the normality.

Variables:

Edad *p-value 1.52e-18*

Medu *p-value 7.90e-23*

Fedu *p-value 1.52e-22*

Traveltime *p-value 1.52e-31*

Studytime *p-value 1.52e-26*

Failures *p-value 1.52e-41*

Famrel *p-value 1.52e-26*

Freetime *p-value 1.52e-19*

Goout *p-value 1.52e-19*

Dalc *p-value 1.52e-36*

Walc *p-value 1.52e-24*

Health *p-value 1.52e-25*

Absences *p-value 1.52e-29*

G1 *p-value 1.52e-07*

G2 *p-value 1.52e-12*

G3 *p-value 1.52e-17*

**Ninguna corresponde a una distribución normal**

**Método NNMF (non negative matrix factorizacion)**

Es para dividir una matriz nxm en dos, una nxk y otra kxm.

Como son dos matrices diferentes, y queres que se mantenga el peso de los valores, intentas minimizar la distancia entre los valores originales de la matriz X y los de las dos matrices nuevas, llámense H y W (mientras X, H y W sean todas positivas)

* Min((X-HW)2)

No se puede resolver matemáticamente, ya que el hessiano no es definido positivo para todos los valores de W, H y X (aunque todos estos sean positivos). Esto quiere decir que no se pueden encontrar minimos globales, pero si locales. Si aproximas X a HW, podemos buscar minimos locales.

Algoritmo *multiplicative update* busca encontrar minimizar estos minimos locales

**def** random\_initialization(A,rank): number\_of\_documents = A.shape[0] number\_of\_terms = A.shape[1] W = np.random.uniform(1,2,(number\_of\_documents,rank)) H = np.random.uniform(1,2,(rank,number\_of\_terms)) **return** W,H **def** nndsvd\_initialization(A,rank): u,s,v=np.linalg.svd(A,full\_matrices=**False**) v=v.T w=np.zeros((A.shape[0],rank)) h=np.zeros((rank,A.shape[1])) w[:,0]=np.sqrt(s[0])\*np.abs(u[:,0]) h[0,:]=np.sqrt(s[0])\*np.abs(v[:,0].T) **for** i **in** range(1,rank): ui=u[:,i] vi=v[:,i] ui\_pos=(ui>=0)\*ui ui\_neg=(ui<0)\*-ui vi\_pos=(vi>=0)\*vi vi\_neg=(vi<0)\*-vi ui\_pos\_norm=np.linalg.norm(ui\_pos,2) ui\_neg\_norm=np.linalg.norm(ui\_neg,2) vi\_pos\_norm=np.linalg.norm(vi\_pos,2) vi\_neg\_norm=np.linalg.norm(vi\_neg,2) norm\_pos=ui\_pos\_norm\*vi\_pos\_norm norm\_neg=ui\_neg\_norm\*vi\_neg\_norm **if** norm\_pos>=norm\_neg: w[:,i]=np.sqrt(s[i]\*norm\_pos)/ui\_pos\_norm\*ui\_pos h[i,:]=np.sqrt(s[i]\*norm\_pos)/vi\_pos\_norm\*vi\_pos.T **else**: w[:,i]=np.sqrt(s[i]\*norm\_neg)/ui\_neg\_norm\*ui\_neg h[i,:]=np.sqrt(s[i]\*norm\_neg)/vi\_neg\_norm\*vi\_neg.T **return** w,h **def** mu\_method(A,k,max\_iter,init\_mode='random'): **if** init\_mode == 'random': W ,H = random\_initialization(A,k) **elif** init\_mode == 'nndsvd': W ,H = nndsvd\_initialization(A,k) norms = [] e = 1.0e-10 **for** n **in** range(max\_iter): *# Update H* W\_TA = W.T@A W\_TWH = W.T@W@H+e **for** i **in** range(np.size(H, 0)): **for** j **in** range(np.size(H, 1)): H[i, j] = H[i, j] \* W\_TA[i, j] / W\_TWH[i, j] *# Update W* AH\_T = A@H.T WHH\_T = W@H@H.T+ e **for** i **in** range(np.size(W, 0)): **for** j **in** range(np.size(W, 1)): W[i, j] = W[i, j] \* AH\_T[i, j] / WHH\_T[i, j] norm = np.linalg.norm(A - W@H, 'fro') norms.append(norm) **return** W ,H ,norms

**Método de Regresión Forward, Backward y Stepwise**

La forward inicia con un modelo sin variables y en cada paso le agrega una nueva. Se agregan variables siempre y cuando estas sean significativas con un alfa del 10%, y tenga el R2 mas alto

La backward inicia con el modelo completo, y va quitando las menos significativas. El alfa q se usa para ver si permanece es también del 10% (el minimo)

Es una combinación de las anteriores, e inicia sin variables. En cada paso, me fijo en las variables q tengo a ver si hay alguna q ahora no es significativa, y si la hay al 10% la saca (y repite el paso).