Clasificacion

Ignacio Vellido

11/24/2020

Cargamos los datos

```
-- Column specification --------------------------------------------------------
cols(
  Age = col_double(),
  Year = col_double(),
  Positive = col_double(),
  Survival = col_character()
)
```

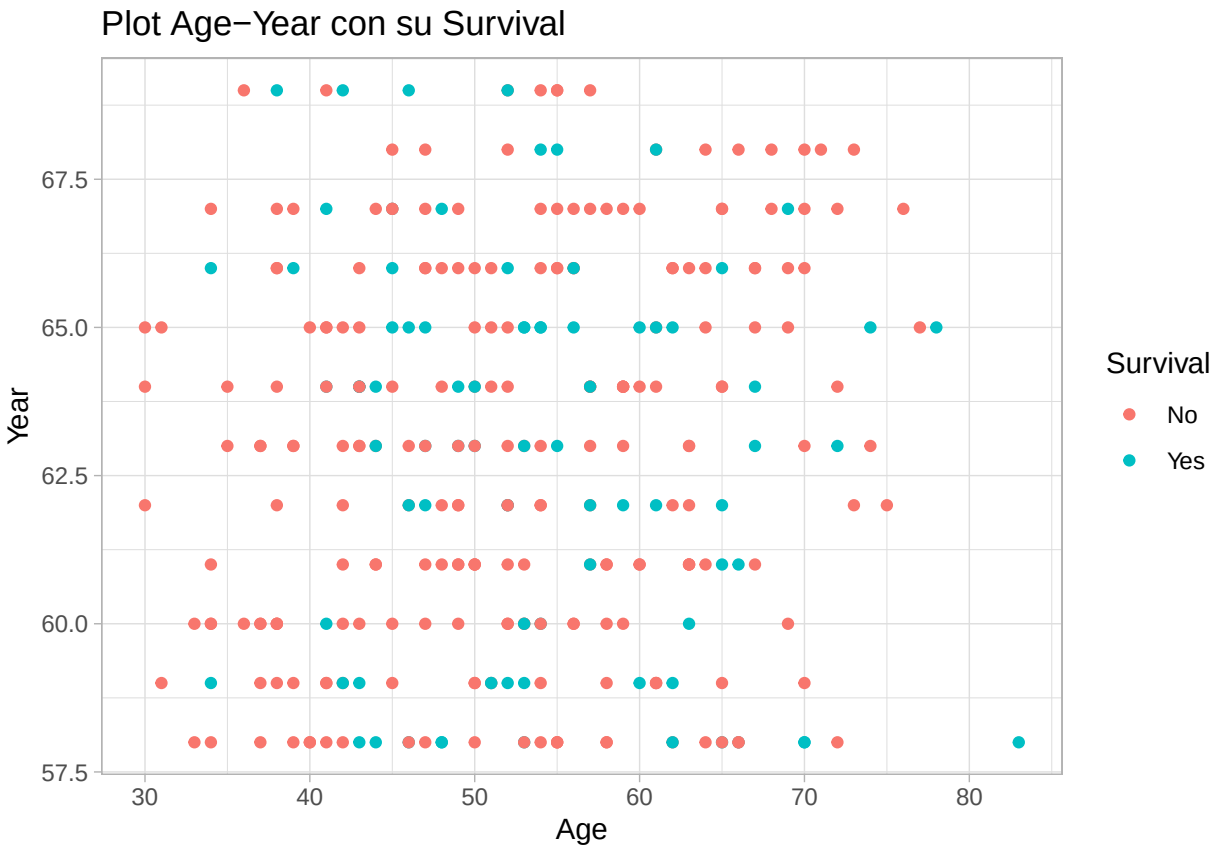
Los preprocesamos (estandarización)

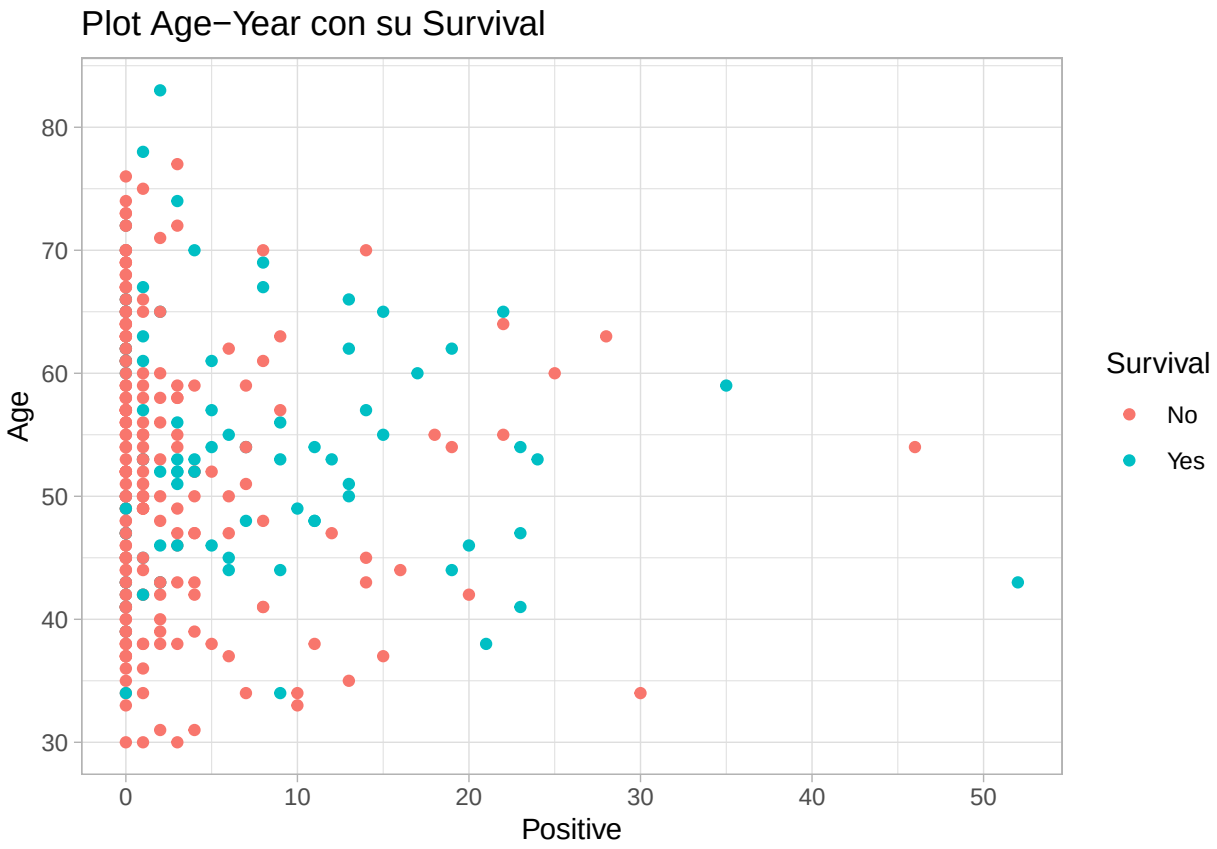
Creamos holdout del 90%

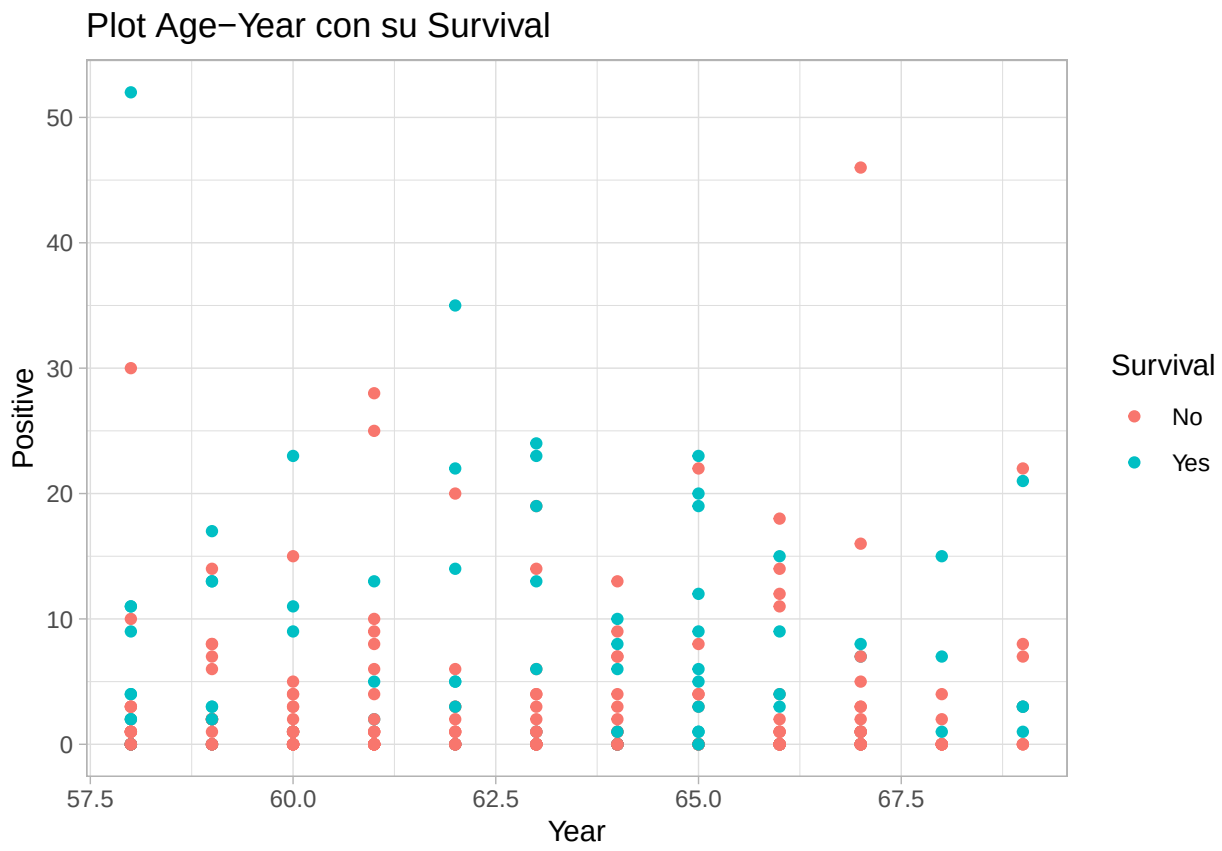
Separamos los datos de las etiquetas

Utilizar el algoritmo k-NN probando con diferentes valores de k. Elegir el que considere más adecuado para su conjunto de datos. Analice qué ocurre en los valores de precisión en training y test con los diferentes valores de k.

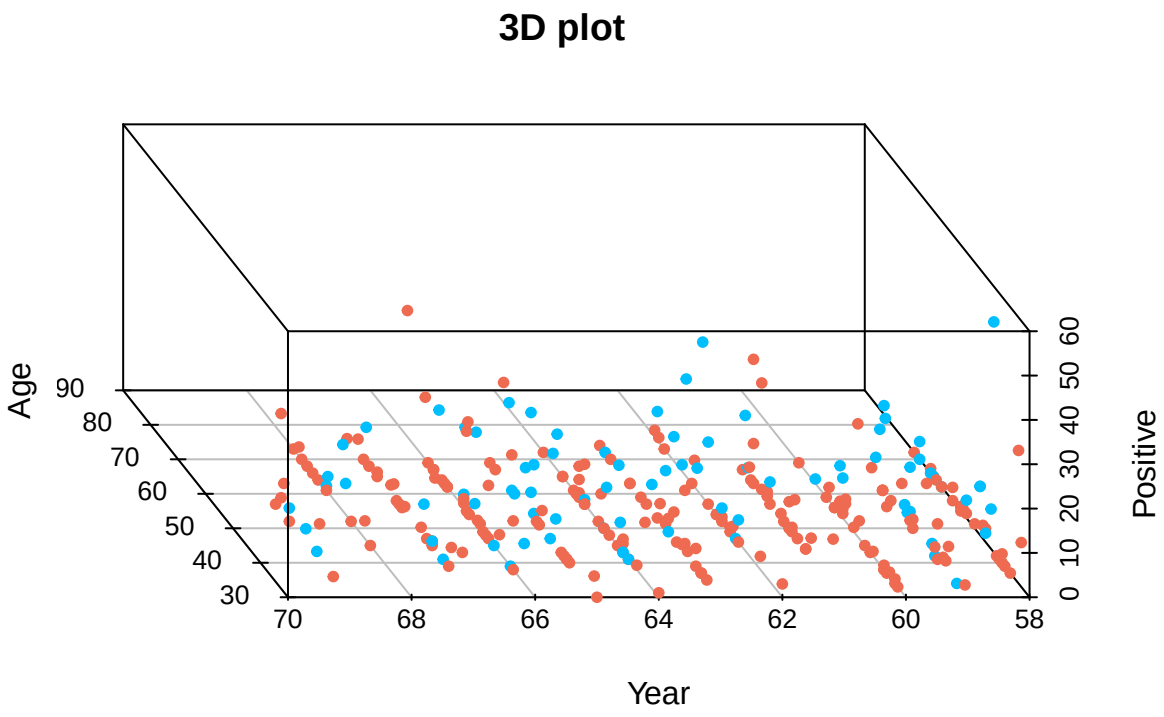
Recordamos los gráficos 1-1 con las clasificaciones, vistos en el EDA.


Plot Age−Year con su Survival
Year
67.5
65.0
62.5
60.0
57.5
30
40
50
60
70
80
Age
Survival
No
Yes

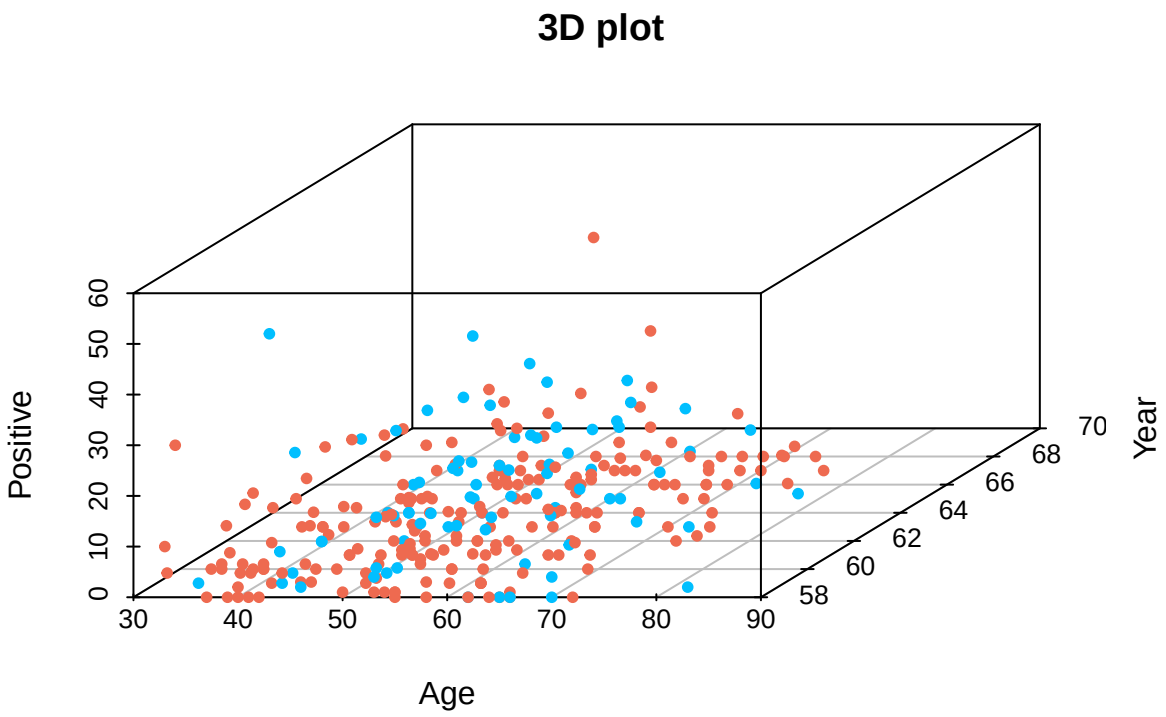

Plot Age−Year con su Survival
Age
80
70
60
50
40
30
0
10
20
30
40
50
Positive
Survival
No
Yes




```
Warning: Unknown or uninitialised column: `color`.
```




```
Warning: Unknown or uninitialised column: `color`.
```



De cara a un algoritmo KNN, apreciamos los datos muy entremezclados, con mayor tendencia a agruparse los no supervivientes que los que sí, pero nada que nos llame la atención.

Debido a esto vamos a empezar con un valor de K relativamente bajo y vamos a ir aumentándolo poco a poco. Tenemos que tener cuidado con el overfitting, eso sí.

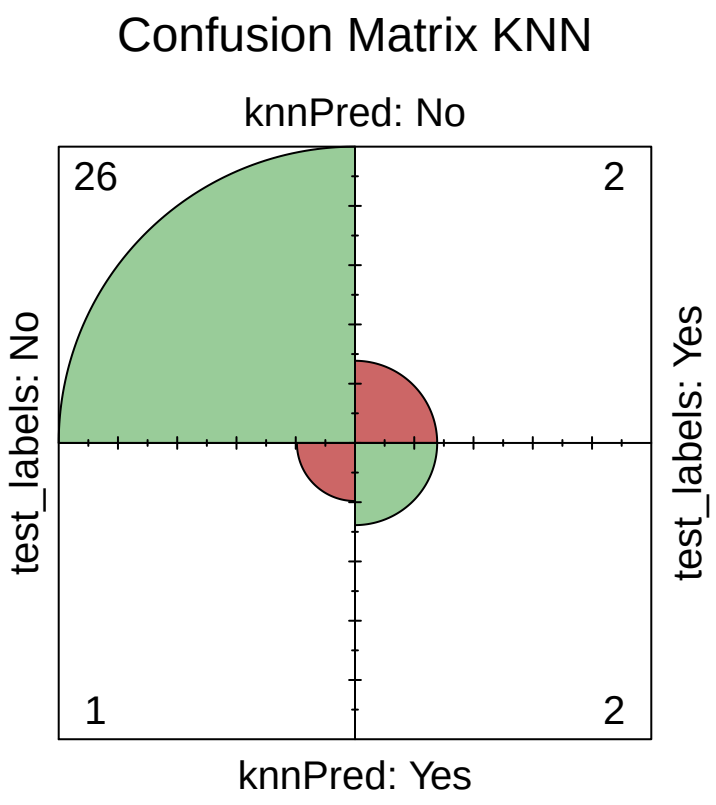
```
k-Nearest Neighbors

275 samples
  3 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 248, 247, 247, 247, 247, 247, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   3  0.6947599  0.2034939
   4  0.6832418  0.1615034
   5  0.6884412  0.1100164
   6  0.6879121  0.1426382
   7  0.6950549  0.0983539
   8  0.7165954  0.1496547
   9  0.7240130  0.1881672
  10  0.7164632  0.1470534
  11  0.7277269  0.1753847
```

```
  12  0.7203195  0.1708068
  13  0.7241656  0.1767408
  14  0.7313085  0.2015352
  15  0.7422873  0.2320214

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 15.
```

Recordamos que los datos ya estaban previamente preprocesados (estandarizados, concretamente)

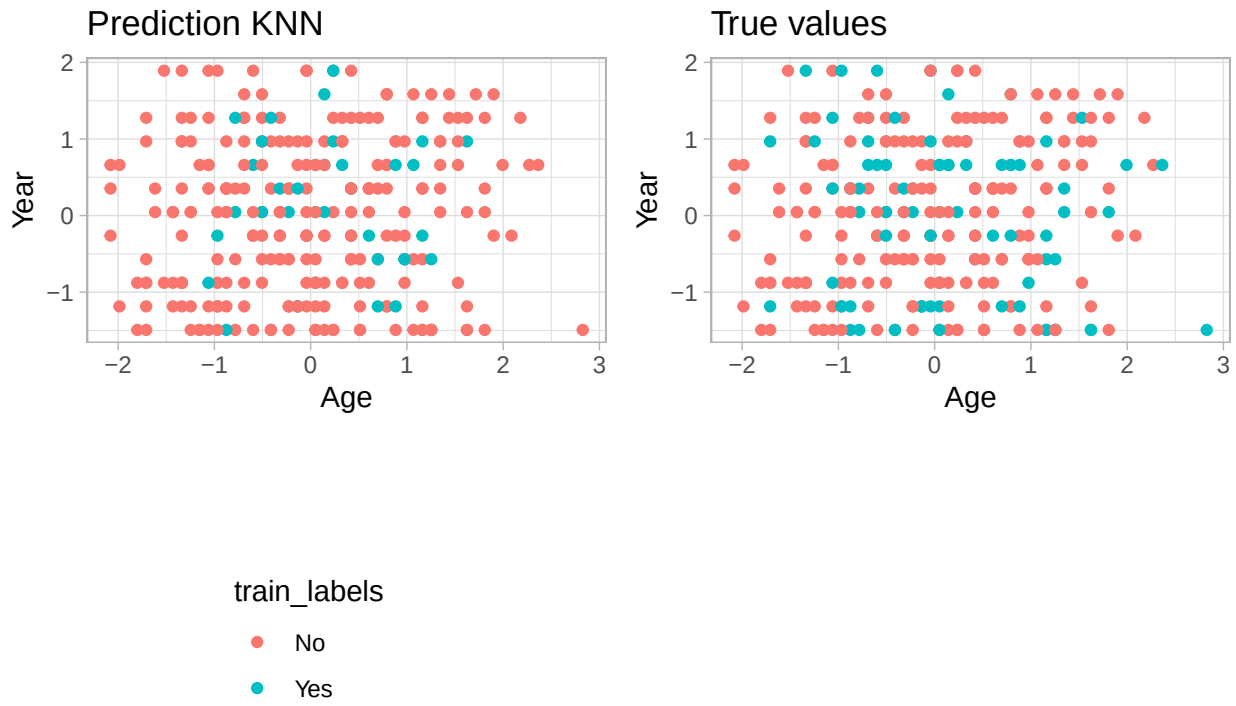
```
       test_labels
knnPred No Yes
    No  26   2
    Yes  1   2
 Accuracy     Kappa
0.9032258 0.5181347
```

Se aprecia que la accuracy en test es ligeramente inferior (~6 centésimas) a la de training.



Vemos que al estar los datos tan entremezclados ni siquiera con un K pequeño sobreaprende, es ya con un K medianamente alto (= 13) donde obtiene mayor accuracy en train.

Probablemente esto se deba a la gran mezcla de los datos, de forma que necesite la "opinión" de un gran número de vecinos para poder predecir con mayor confianza el nuevo valor.



Vemos que el uso de un K alto hace que perdamos puntos de Yes. Probablemente al ser minoría el error cometido clasificándolos como No es menor y por eso obtiene mejor accuracy.

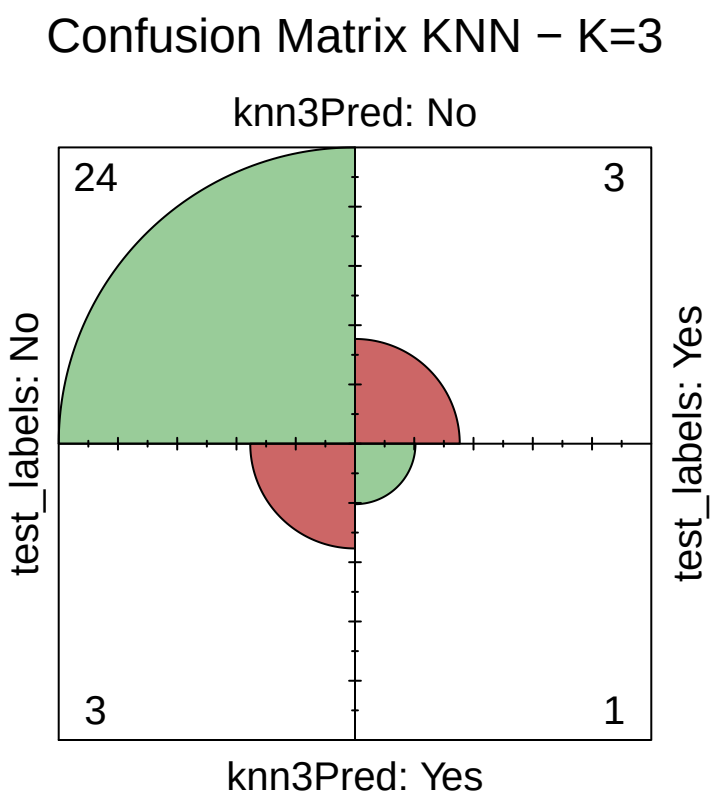
En principio (a falta de evaluar con CV) creemos que en comparación con otros algoritmos el hecho de que el dataset este desbalanceado y los datos muy entremezclados la calidad real (fuera del entrenamiento) no sea muy buena. Podría ser que en la población se mantega este desbalanceo en los datos y funcione bien, pero en caso de que no lo sea, el valor de K tan alto haría que probablemente se tienda a devolver una predicción de No.

Como habíamos comentado al principio, para el problema que nos atañe quizás esto podría ser incluso un hecho positivo, ya que los falsos positivos sería algo que querríamos evitar a toda costa.

Podemos también coger otros valores de K en test. Puesto que hemos obtenido los mejores resultados en training con un K de 13, que es un valor relativamente alto, podemo probar con uno bajo y uno intermedio (3 y 7)

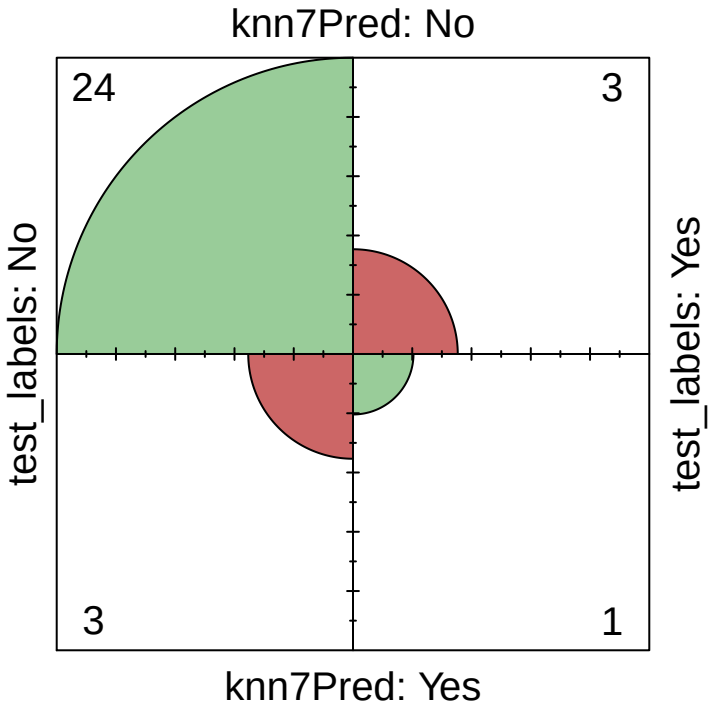
```
k-Nearest Neighbors

275 samples
  3 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 248, 247, 248, 248, 247, 247, ...
Resampling results:
```

```
  Accuracy  Kappa
  0.697619  0.1988554

Tuning parameter 'k' was held constant at a value of 3
```




```
        test_labels
knn3Pred No Yes
     No  24   3
     Yes  3   1
 Accuracy     Kappa
0.8064516 0.1388889
```

```
k-Nearest Neighbors

275 samples
  3 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 248, 247, 247, 248, 248, 247, ...
Resampling results:

  Accuracy   Kappa
  0.6761905  0.08141897

Tuning parameter 'k' was held constant at a value of 7
```




```
        test_labels
knn7Pred No Yes
     No  24   3
     Yes  3   1
 Accuracy     Kappa
0.8064516 0.1388889
```

K=7 vemos que es el que más sufre al evaluar en test, y ambos (tal y como nos había indicado CV) tienen una calidad bastante inferior a un K=13

Utilizar el algoritmo LDA para clasificar. No olvide comprobar las asunciones.

Comprobamos asunciones:

1- Distribución aleatoria: No nos queda más remedio que creer que sí.

2- Cada predictor sigue una distribución normal: Ya vimos en el EDA que esto no era cierto. El test de Shapiro nos demostraba que no y los QQ-plots nos lo hacían ver claramente. Técnicamente sabiendo esto no deberíamos usar LDA, pero puesto que esto es un proyecto seguimos igualmente.

Aún así, las variables Age y Year no parecen seguir una distribución demasiado "rara" (en comparación con una normal), por lo que es posible que obtengamos resultados decentes.

3- Las clases siguen la misma matriz de covarianza: Solo nos interesa la diagonal

```
New names:
* NA -> ...4
New names:
```

```
* NA -> ...4
```

```
Para clase Yes:
      Age      Year  Positive
0.9176155 1.0113109 1.6788033
Para clase No:
      Age      Year  Positive
1.0756415 0.9752581 0.5448553
```

Las variables Age y Positive parecen seguir distintas varianzas, lo aseguramos con un test estadístico.

Puesto que nuestras variables no siguen una distribución normal, no podemos hacer el test de homogeneidad de Barlett. Utilizamos por tanto el de Levene

`Age:`

	Df	F value	Pr(>F)
group	1	1.799898	0.1807261
	304	NA	NA

`Year:`

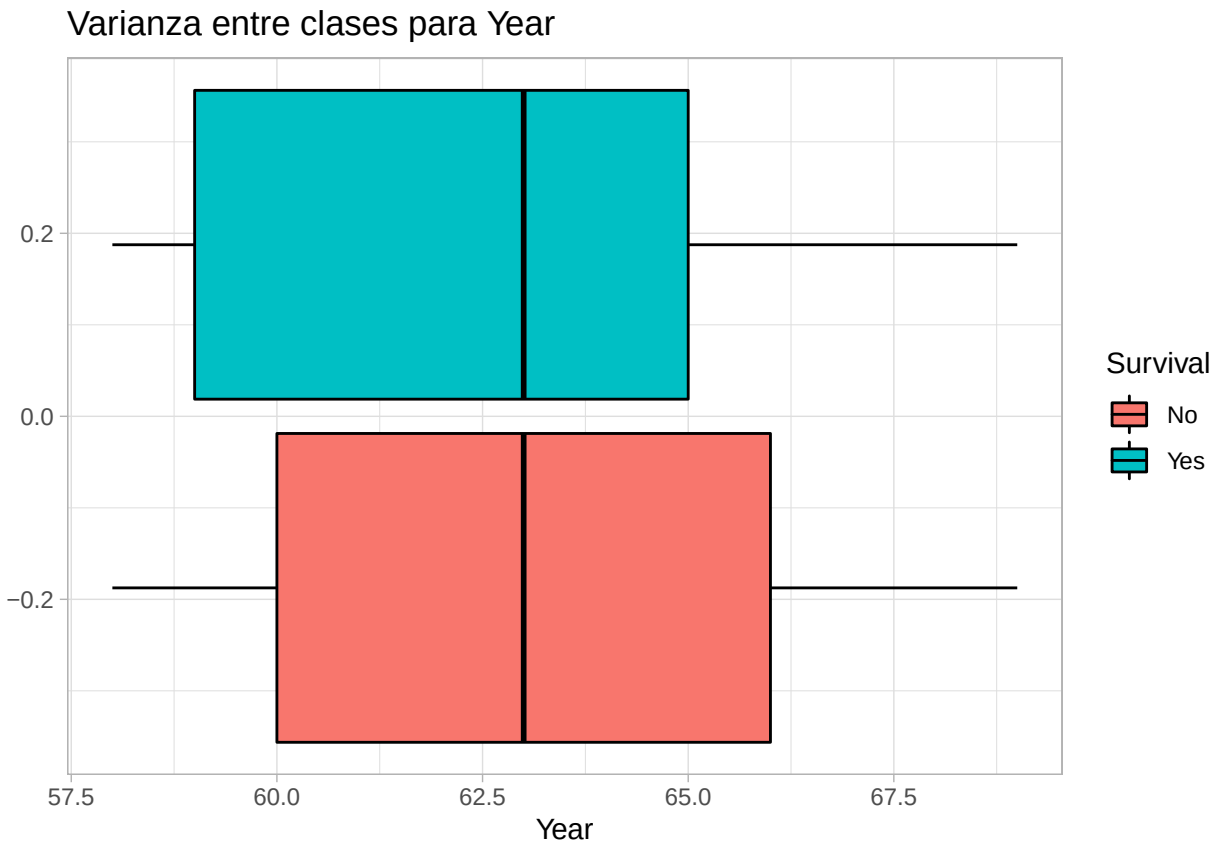
	Df	F value	Pr(>F)
group	1	0.0624405	0.8028481
	304	NA	NA

`Positive:`

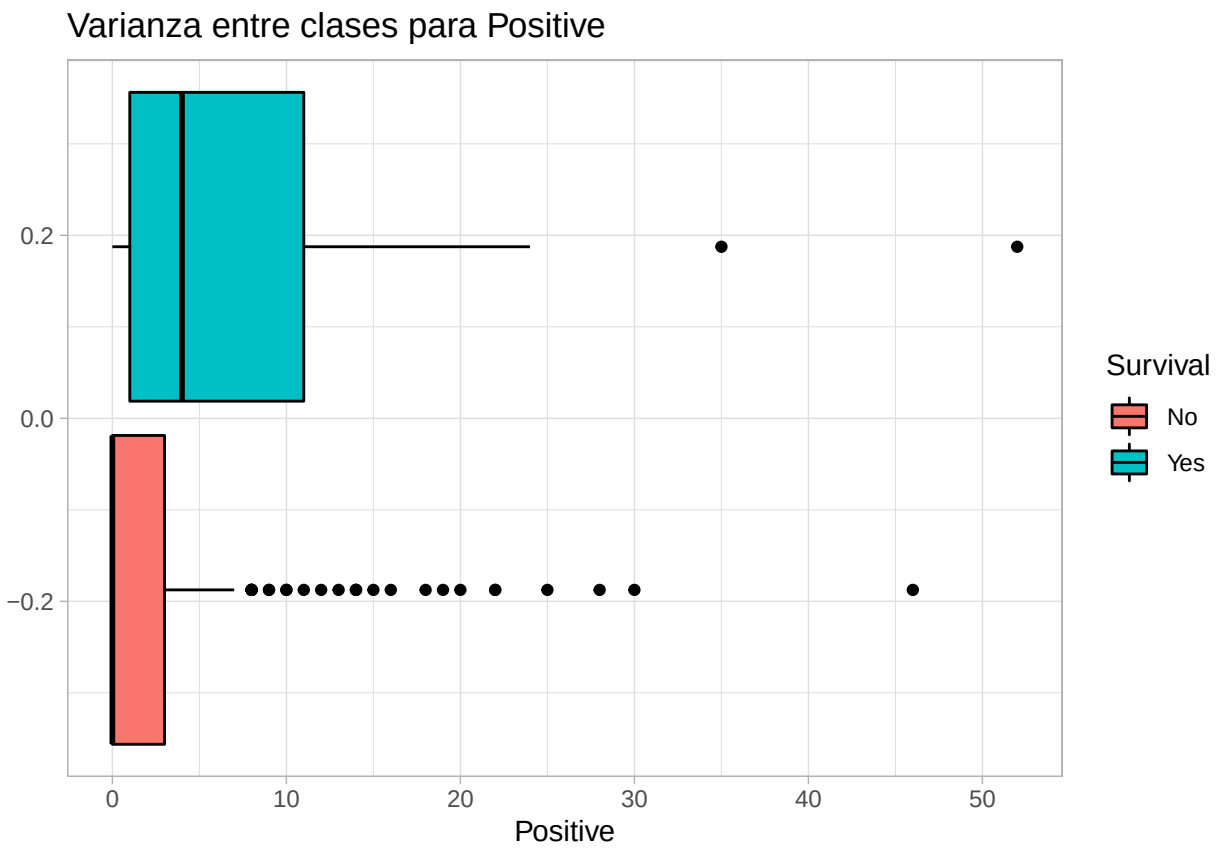
	Df	F value	Pr(>F)
group	1	18.78912	1.99e-05
	304	NA	NA

Indicándonos que solo se puede asegurar que la variable Positive no tiene homogeneidad entre clases diferentes.

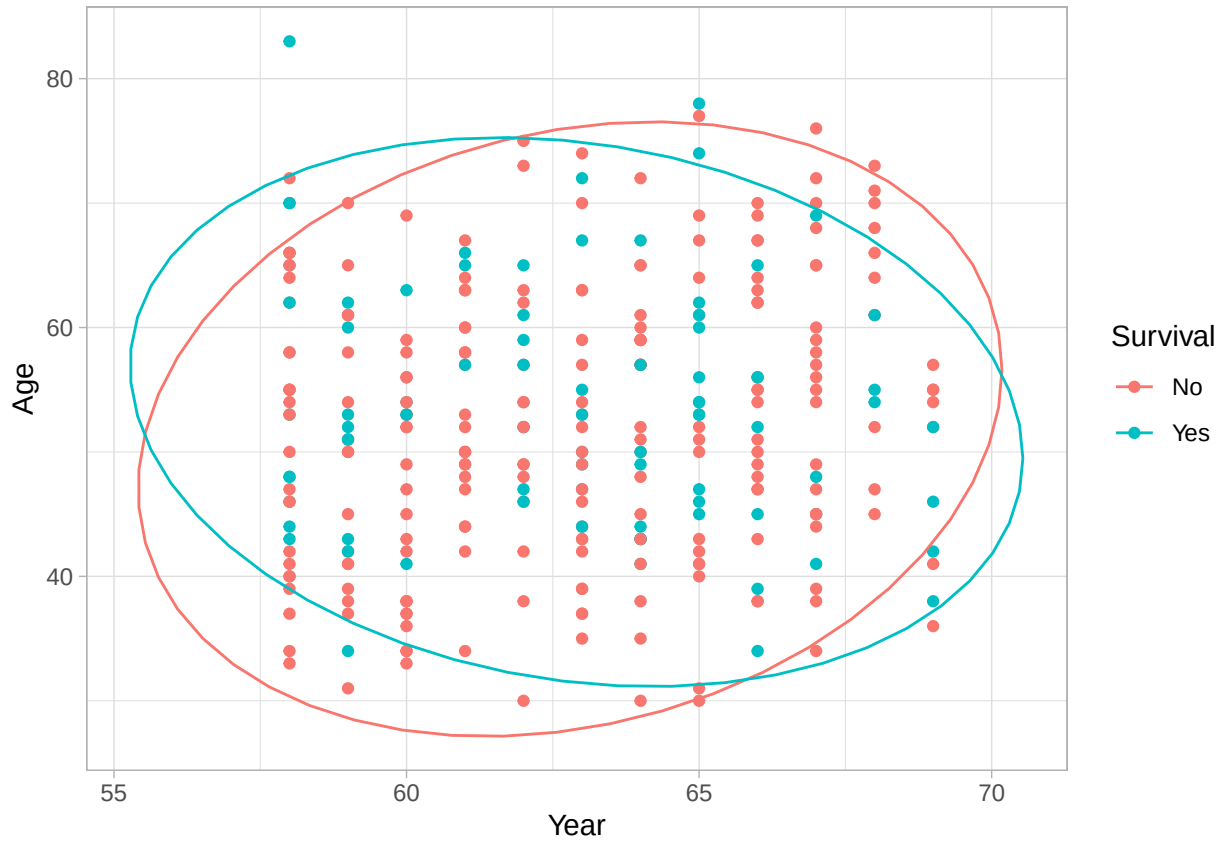
Podemos verlo más claro gráficamente

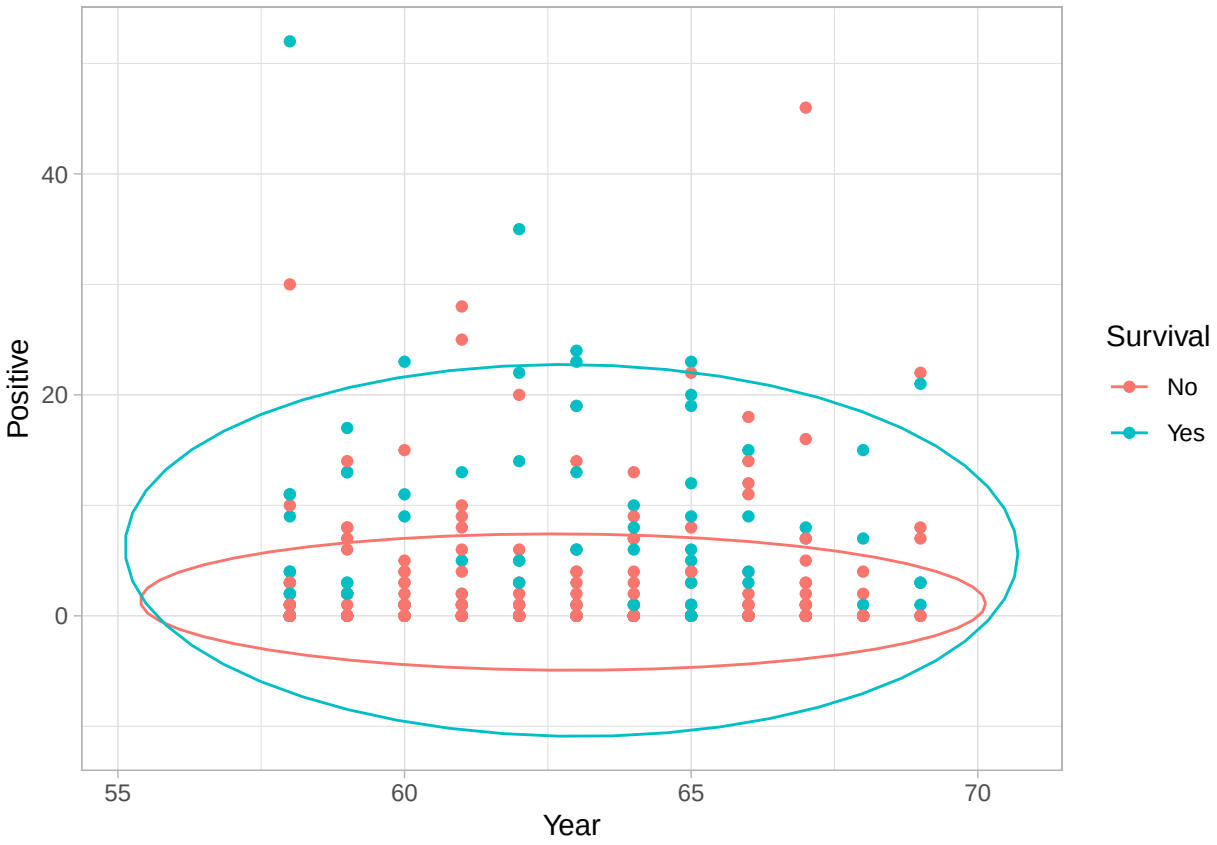

Varianza entre clases para Year
0.2
0.0
−0.2
57.5
60.0
62.5
65.0
67.5
Year
Survival
No
Yes

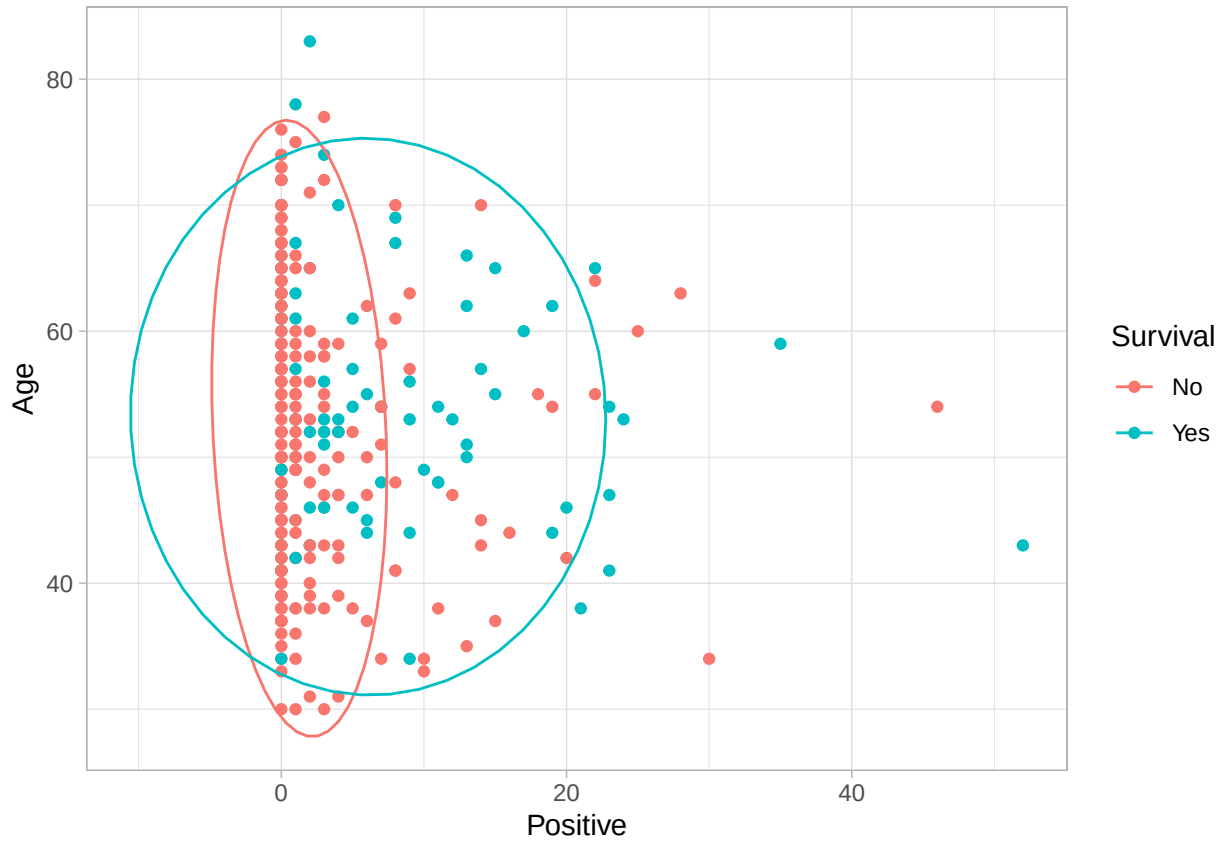

Varianza entre clases para Age
0.2
0.0
−0.2
30
40
50
60
70
80
Age
Survival
No
Yes



Podemos mostrar las varianzas en grafos de elipses:


80
60
40
Age
55
60
65
70
Year
Survival
No
Yes


Positive
40
20
0
55
60
65
70
Year
Survival
No
Yes



Se nota que la causa de que no se rechace el test para esta variable es la gran cantidad de datos con Positive=0

Por tanto para LDA no podemos hacer uso de la variable Positive, pero sí de las otras dos.

Aunque solo es recomendable, y no son cualidades necesarias para obtener solución en LDA: - Tenemos más instancias que predictores, por varios órdenes de magnitud. - Los predictores son independientes. - No tenemos varianza cercana a cero

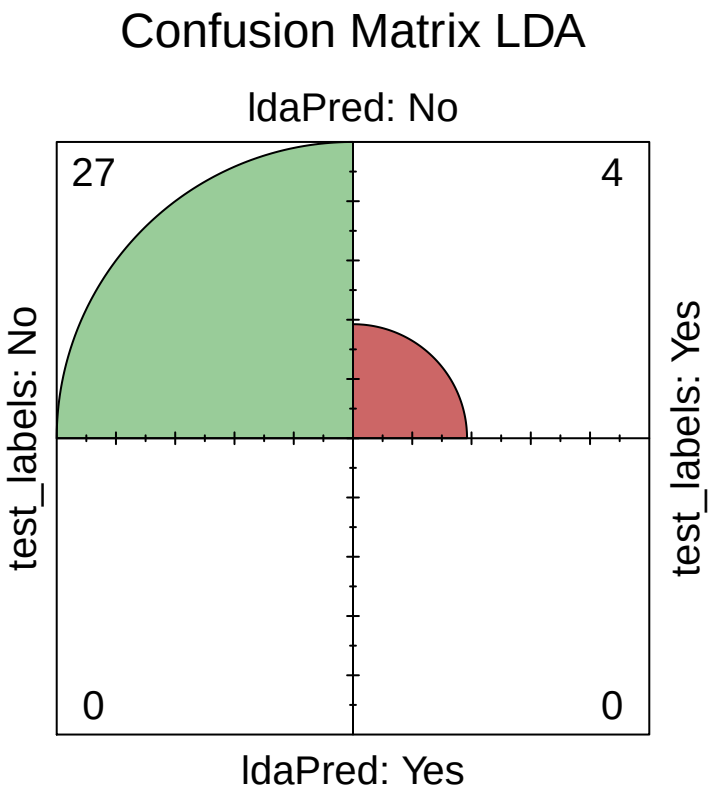
Aplicando LDA

```
Call:
lda(x, y)

Prior probabilities of groups:
  No  Yes
0.72 0.28

Group means:
             Age        Year
No  -0.04141411 0.004845716
Yes  0.11031976 0.021276742

Coefficients of linear discriminants:
            LD1
Age  0.98140853
Year 0.03411981
Cross-Validated (10 fold) Confusion Matrix
```

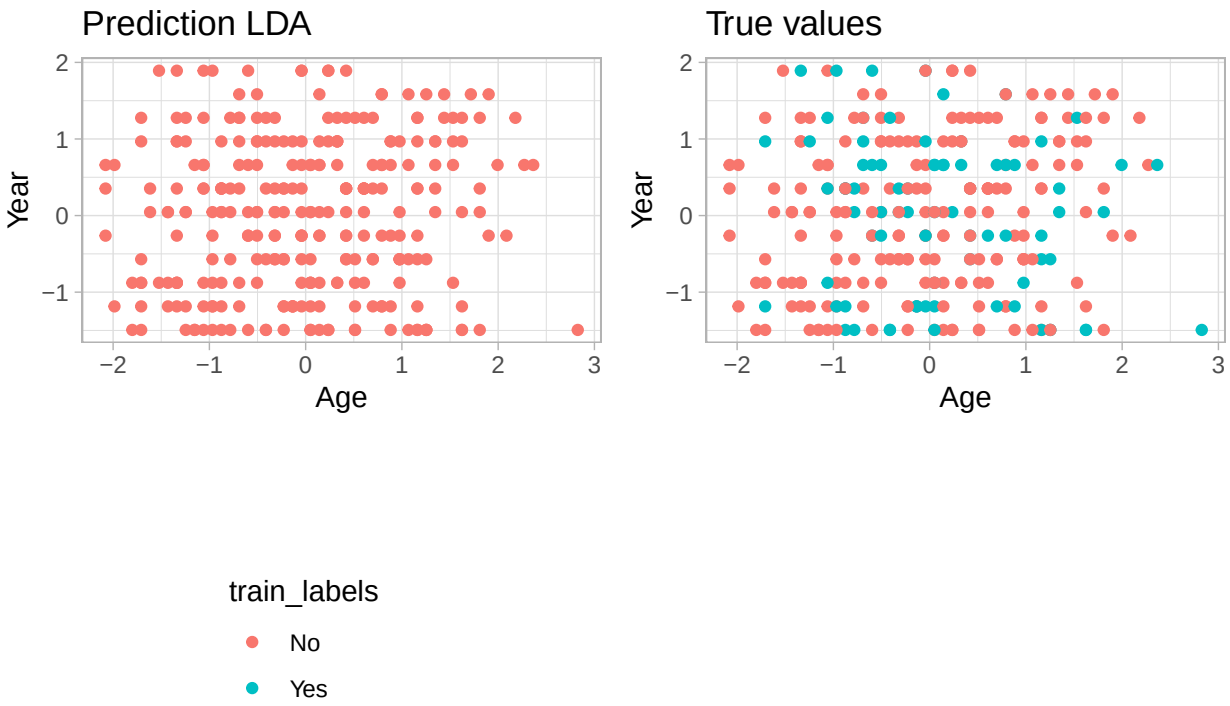
```
(entries are percentual average cell counts across resamples)

          Reference
Prediction No Yes
       No  72  28
       Yes  0   0

 Accuracy (average) : 0.72
```

Predecimos en test

```
       test_labels
ldaPred No Yes
    No  27   4
    Yes  0   0
 Accuracy     Kappa
0.8709677 0.0000000
```

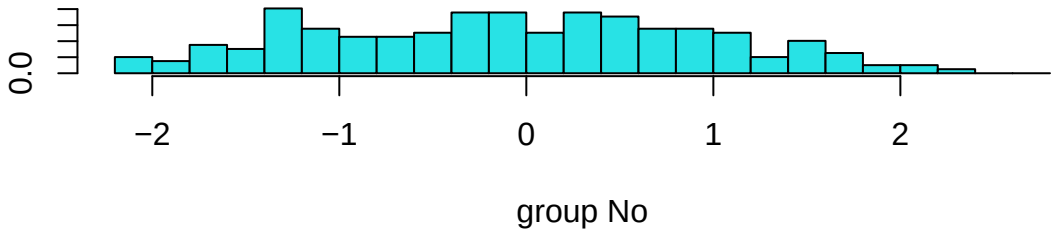


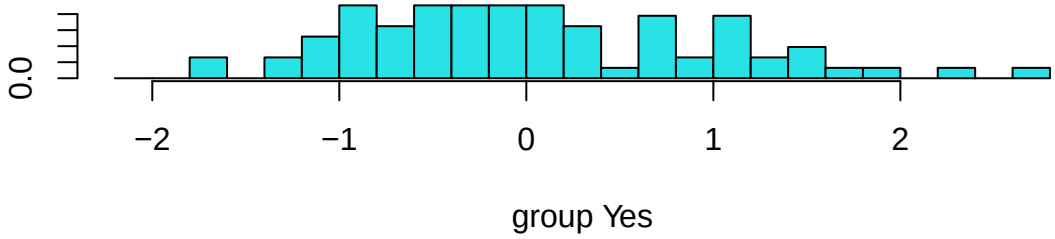
En training



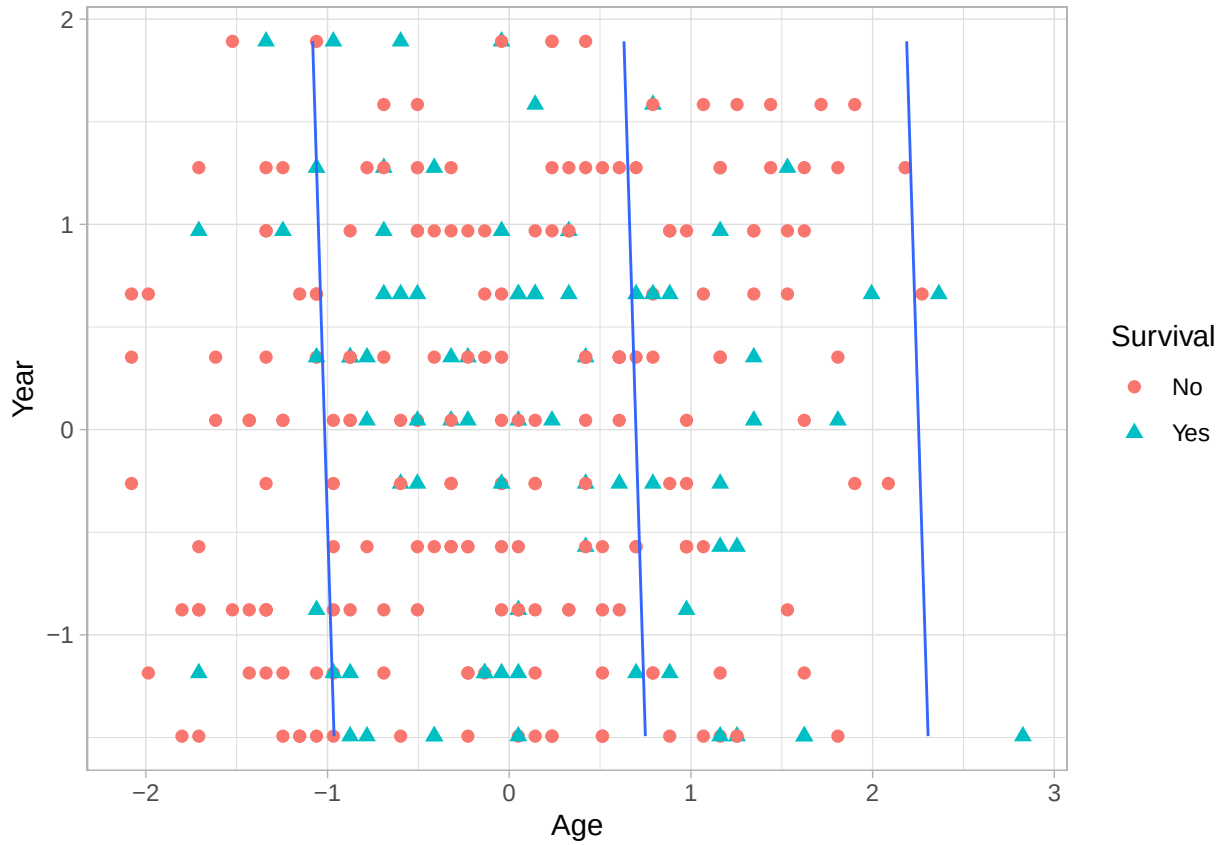
Hacemos un plot del ajuste

```
New names:
* NA -> ...4
```






```
NULL
```



(Estamos pintando un contorno 3D y por eso nos salen múltiples líneas en el gráfico)

Tenemos un dataset bastante desbalanceado, y LDA no predice para la clase Yes. Con esto se asegura un alto accuracy en nuestro entrenamiento, pero no asegura de que para datos externos vaya a ser así. Pese a ello, no nos queda más remedio que suponer que nuestros datos vienen de la misma muestra aleatoria y por tanto son releventes para la clasificación.

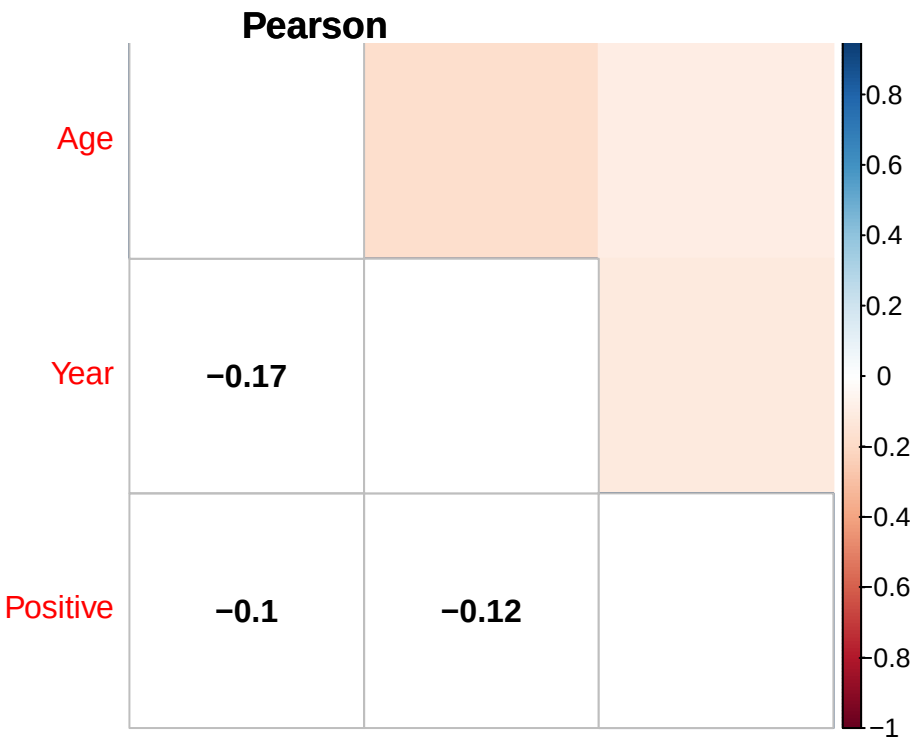
Utilizar el algoritmo QDA para clasificar. No olvide comprobar las asunciones.

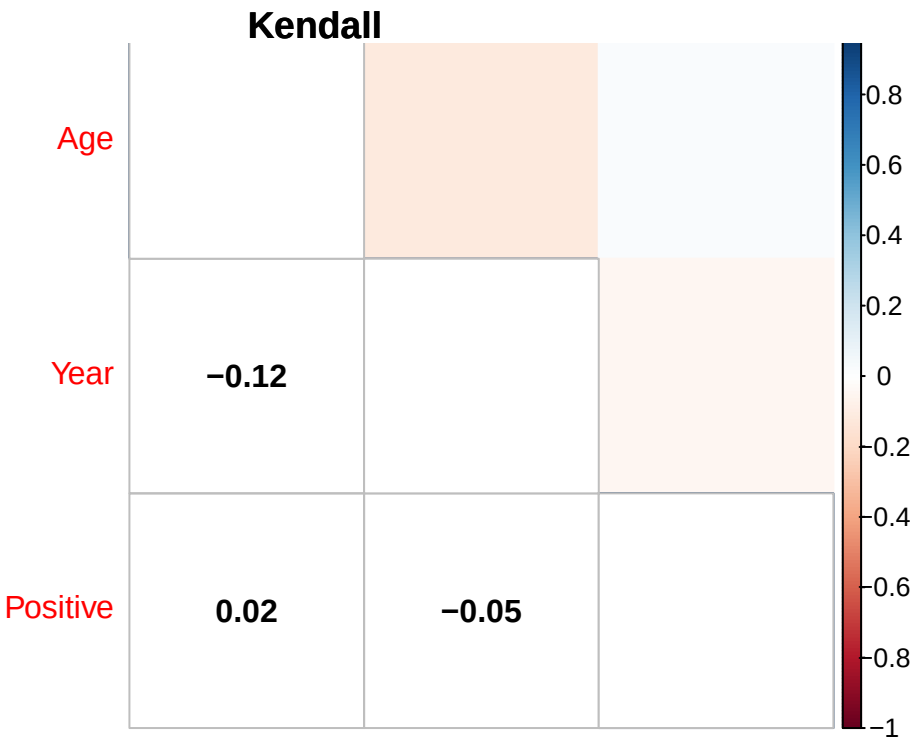
QDA tiene las mismas asunciones de LDA salvo que relaja la norma de que las clases tengan igual covarianza. Esto nos permite usar la variable Positive que habíamos descartado en LDA.

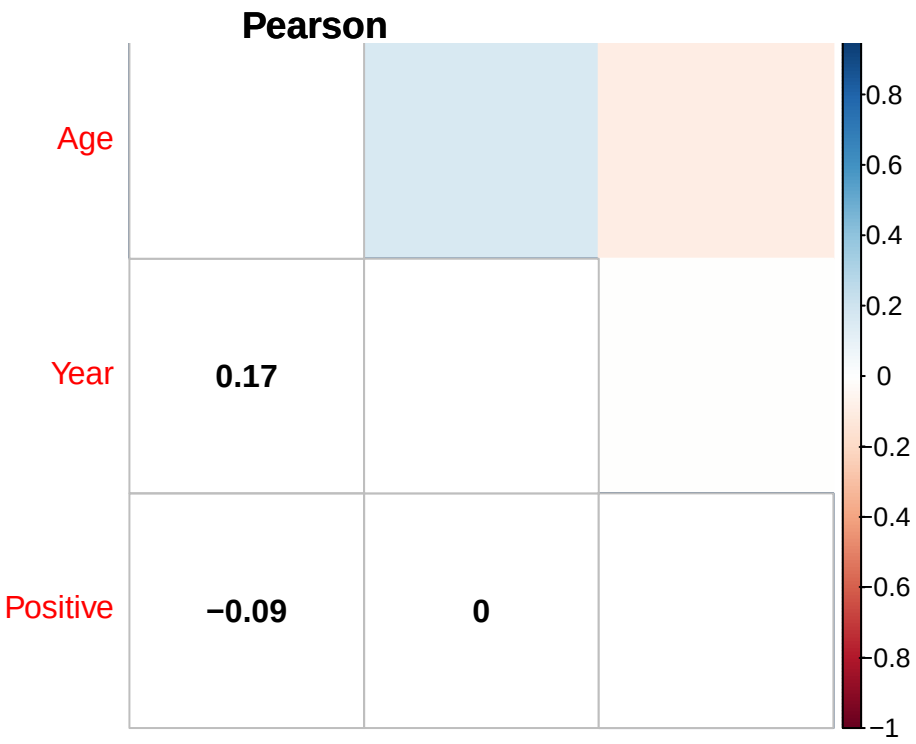
Por tanto tenemos los requisitos de: - Distribución aleatoria - Distribución normal

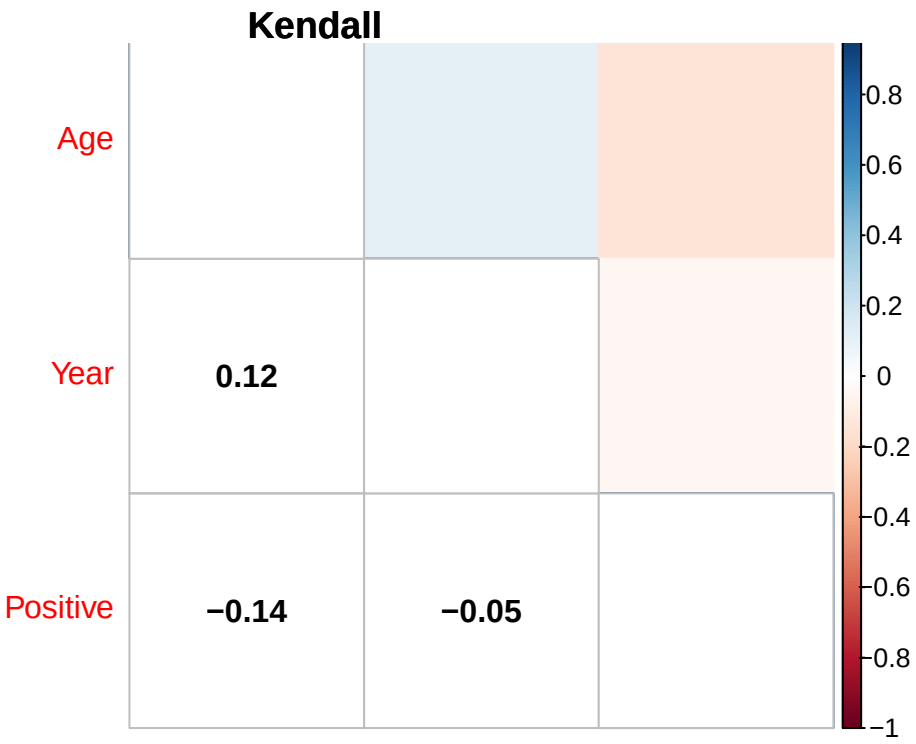
Técnicamente el no cumplir normalidad no imposibilita que se encuentre solución, pero ya no nos lo asegura.

Además tenemos de forma recomendada que: - El número de predictores debe ser menor que el número de instancias de cada clase. Cosa que sabemos que sí por la tabla Yes/No - Los predictores dentro de cada clase no deben estar correlacionados.


Pearson
Age
Year
Positive
−0.17
−0.1
−0.12
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
−1


Kendall
Age
Year
Positive
−0.12
0.02
−0.05
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
−1


Pearson
Age
Year
Positive
0.17
−0.09
0
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
−1

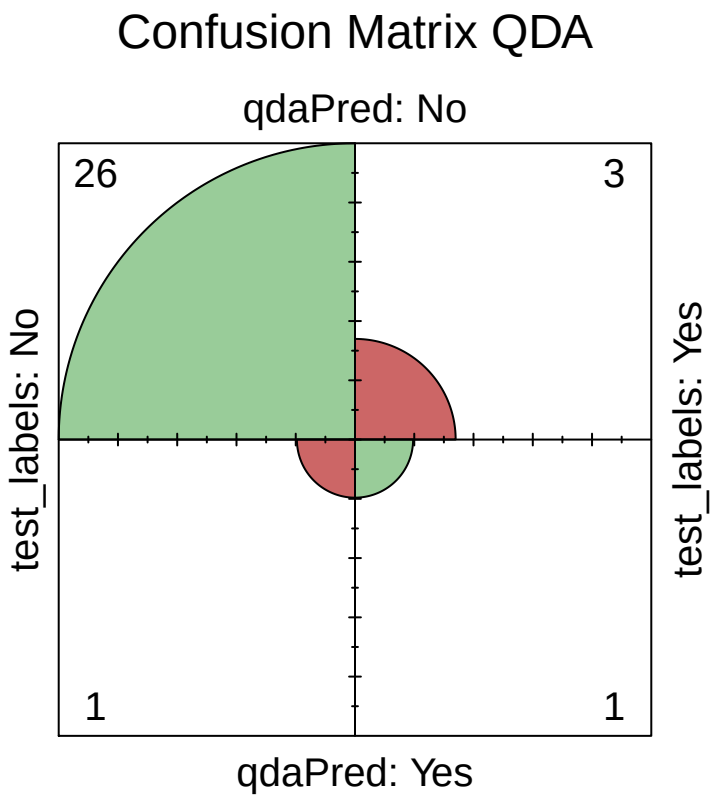


No tenemos predictores correlacionados dentro de cada clase.

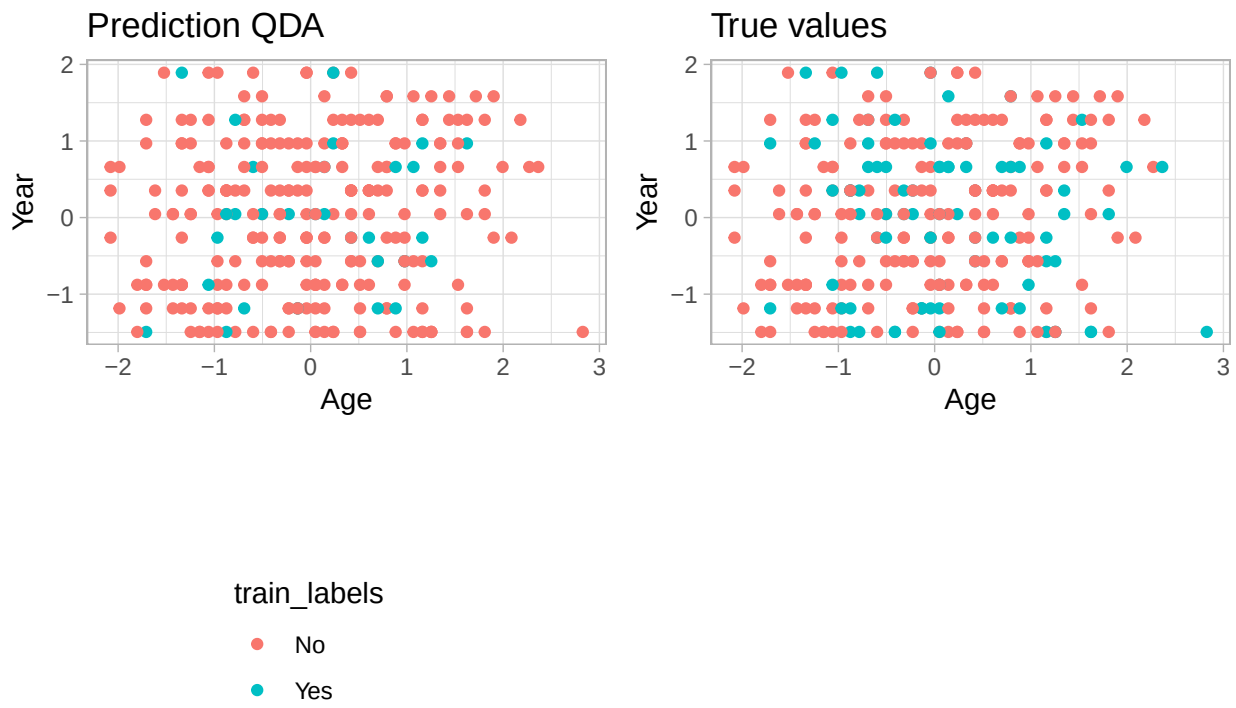
Aplicando QDA

```
Call:
qda(x, y)

Prior probabilities of groups:
  No  Yes
0.72 0.28

Group means:
            Age        Year   Positive
No  -0.04141411 0.004845716 -0.1792534
Yes  0.11031976 0.021276742  0.4804648
Cross-Validated (10 fold) Confusion Matrix

(entries are percentual average cell counts across resamples)

          Reference
Prediction   No  Yes
       No  67.3 21.5
       Yes  4.7  6.5

 Accuracy (average) : 0.7382
```

```
       test_labels
qdaPred No Yes
    No  26   3
    Yes  1   1
 Accuracy     Kappa
0.8709677 0.2705882
```



En training



Obtenemos resultados extremadamente similares a LDA, pero en este caso vemos que sí se predice la clase Yes.

Comparar los resultados de los tres algoritmos

Si nos fijamos únicamente en los resultados obtenidos para este problema, los tres algoritmos obtienen el mismo accuracy en nuestro conjunto de test. Aunque las etiquetas de este conjunto contienen elementos de ambas clases, podemos ver que se predice mayoritariamente la clase No. Como se había mencionado nuestro dataset está bastante desbalanceado, por lo que era más probable que se predijera esa clase con mayor facilidad.

(Recordamos que las medidas devueltas en cada algoritmo provienen de un CV de 10-fold usando el paquete caret)

```
Predicciones LDA:
 [1] No No No No No No No No No No No No No No No No No No No No No No No No No
[26] No No No No No No
Levels: No Yes
Predicciones QDA:
 [1] No  No  Yes No  No  No  No  Yes No  No  No  No  No  No  No  No  No  No  No
[20] No  No  No  No  No  No  No  No  No  No  No  No
Levels: No Yes
Predicciones KNN:
 [1] No  No  Yes No  No  No  No  Yes No  No  No  No  No  No  No  No  No  Yes No
[20] No  No  No  No  No  No  No  No  No  No  No  No
Levels: No Yes
```

```
Etiquetas:
 [1] No  No  Yes No  No  No  No  No  Yes No  No  No  No  No  No  No  No  Yes No
[20] No  No  No  No  No  No  No  No  Yes No  No  No
Levels: No Yes
```

```
Accuracy KNN:
 Accuracy     Kappa
0.9032258 0.5181347

Accuracy LDA:
 Accuracy     Kappa
0.8709677 0.0000000

Accuracy QDA:
 Accuracy     Kappa
0.8709677 0.2705882
```

Tenemos la misma accuracy, pero diferentes valores de Kappa, siendo en general todos bajos.

Pese a esto, puesto que no cumplimos las asunciones necesarias para asegurarnos resultados en LDA y QDA, para este problema obtaríamos por usar el algoritmo KNN.

Por otro lado, podemos hacer una comparativa general de la calidad de los algoritmos haciendo uso de las tablas proporcionadas para la práctica. Para ello, primeramente...

Podríamos leer los de training pero para comparar necesitamos los de test

```
Warning: Missing column names filled in: 'X1' [1]


-- Column specification --------------------------------------------------------
cols(
  X1 = col_character(),
  out_test_knn = col_double(),
  out_test_lda = col_double(),
  out_test_qda = col_double()
)
```

X1	out_test_knn	out_test_lda	out_test_qda
appendicitis	0.8966667	0.8690909	0.8109091
australian	0.6838235	0.8579710	0.8028986
balance	0.9024546	0.8624101	0.9167905
bupa	0.6865775	0.6837924	0.5991759
contraceptive	0.5448653	0.5091561	0.5173102
haberman	0.7462069	0.7481720	0.7512903
hayes-roth	0.5666667	0.5500000	0.5875000
heart	0.6692308	0.8481481	0.8296296
iris	0.9642857	0.9800000	0.9733333
led7digit	0.7510204	0.7420000	0.6975000
mammographic	0.7977698	0.8241269	0.8194042
monk-2	0.9743632	0.7703433	0.9235535
newthyroid	0.9071429	0.9164502	0.9629870
pima	0.7348861	0.7709930	0.7412403
tae	0.3838095	0.5245833	0.5425000
titanic	0.7850353	0.7760304	0.7733032
vehicle	0.6291452	0.7813305	0.8522409
vowel	0.6428571	0.6030303	0.9191919
wine	0.6959559	0.9944444	0.9888889
wisconsin	0.9735023	0.9592185	0.9519476

Comparativas 1-1 con Wilconxon

LDA vs QDA

```
LDA vs QDA:
    Wilcoxon signed rank exact test

data:  res_norm[, 2] and res_norm[, 1]
V = 96, p-value = 0.7562
alternative hypothesis: true location shift is not equal to 0

R+:   V
114
R-:  V
96
p-value: [1] 0.7561665
```

Obtenemos un ranking de 144 para LDA y 96 para QDA, con un p-valor de 0.75 (o nivel de confianza del 25%).

Esto nos dice que LDA obtiene mejores resultados pero puesto que el p-value es extremadamente grande no podemos afirmar con garantía estadística que las diferencias entre los tests sean notorias.

LDA vs KNN

```
LDA vs KNN:
    Wilcoxon signed rank exact test

data:  res_norm[, 2] and res_norm[, 1]
V = 120, p-value = 0.5958
alternative hypothesis: true location shift is not equal to 0

R+:  V
```

```
90
R-:   V
120
p-value: [1] 0.5958195
```

Ahora obtenemos un ranking de 90 para LDA y 120 para QDA, con un p-valor de 0.59 (o nivel de confianza del 41%).

Seguimos teniendo un p-valor demasiado grande para poder asegurar con significación la diferencia.

QDA vs KNN

```
QDA vs KNN:
    Wilcoxon signed rank exact test

data:  res_norm[, 2] and res_norm[, 1]
V = 141, p-value = 0.1893
alternative hypothesis: true location shift is not equal to 0

R+:  V
69
R-:   V
141
p-value: [1] 0.1893482
```

Por último tenemos un ranking de 69 para LDA y 141 para KNN, con un p-valor de 0.18 (o nivel de confianza del 82%).

Ya ahora podemos afirmar al 82% que los resultados de ambos algoritmos sí son significativamente diferentes, pero para hacerlo con bastante seguridad realmente buscaríamos al menos un 95% de confianza.

Comparativa múltiple con Friedman

```
    Friedman rank sum test

data:  as.matrix(results[, 2:4])
Friedman chi-squared = 0.7, df = 2, p-value = 0.7047
```

El p-value es >0.05 por lo que no podemos concluir que haya al menos un par de algoritmos de calidad diferente.

Análisis post-hoc con Holm

El resultado del test de Friedman ya nos indica que es post-hoc es inútil. Los resultados que se obtengan no nos van a asegurar la diferencia en la calidad de los algoritmos.

Aún así, por completitud en la memoria, aplicamos el test estadístico

```
1 = KNN, 2 = LDA, 3 = QDA
    Pairwise comparisons using Wilcoxon signed rank exact test

data:  as.matrix(results[, 2:4]) and groups

  1    2
```

```
2 1.00 -
3 0.53 1.00

P value adjustment method: holm
```

Vemos que los p-value son lo más altos posibles, carece de sentido intentar diferenciar los algoritmos. Aunque podemos notar, tal y como habíamos visto en los test de Wilcoxon, que la diferencia KNN-QDA probablemente sea mayor que el resto de parejas.