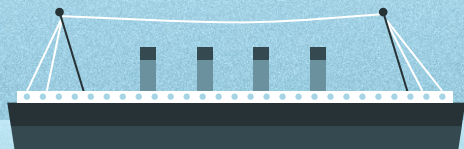


— Inteligencia artificial avanzada para la ciencia de datos I —
Equipo 6

TITANIC

Machine Learning from Disaster



— Reto —

1.

Utilizar machine learning para crear un modelo que prediga qué pasajeros sobrevivieron al choque del Titanic.

Probar las predicciones en Kaggle para medir la precisión.

2.

— Table of contents —

1.

Exploración

Overview del dataset para entender el output esperado.

2.

Limpieza

Preparar los datos de acuerdo al análisis previo.

3.

Construcción del modelo

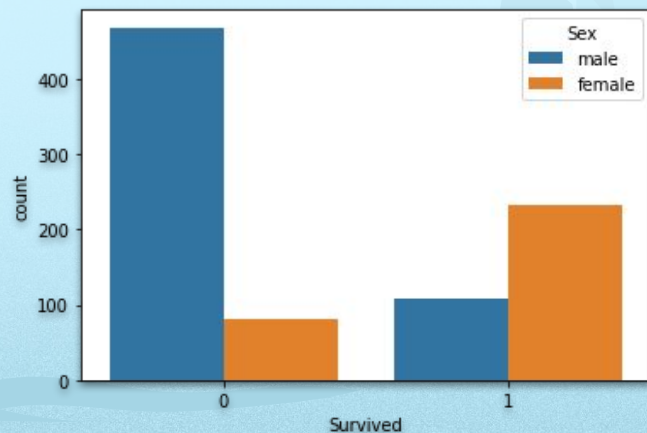
Seleccionar y poner a prueba modelos.

4.

Pruebas

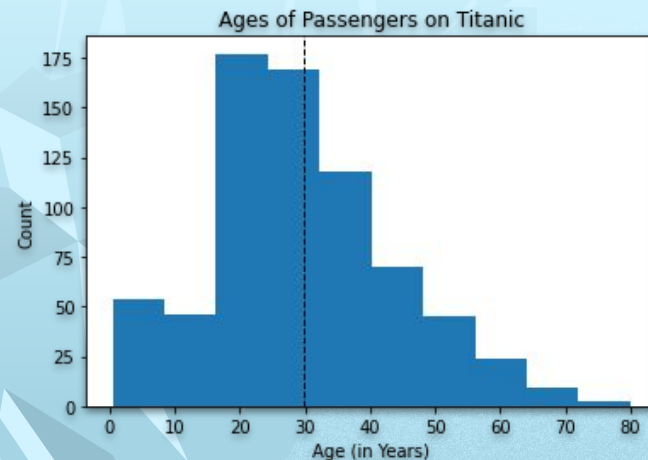
Realizar pruebas y mejorar el modelo.

— Exploración —



```
female    0.543351  
male      -0.543351
```

Correlación entre survived y sex



```
1    71.751412  
0    28.248588
```

Probabilidad de sobrevivir o morir
para mujeres y niños < 16

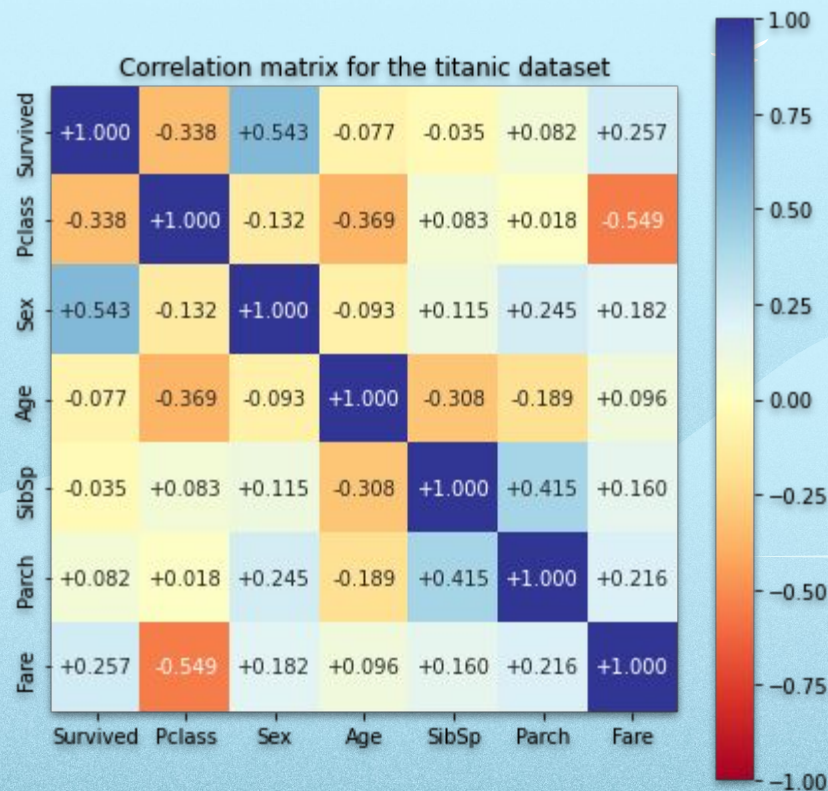
— Exploración —

Probabilidad de sobrevivir o morir
para hombres edad > 16

```
0      82.338308
1      17.661692
```

Media por columna de sobrevivientes y
fallecidos

	Pclass	Age	SibSp	Parch	Fare
Survived					
0	2.531876	30.626179	0.553734	0.329690	22.117887
1	1.950292	28.343690	0.473684	0.464912	48.395408



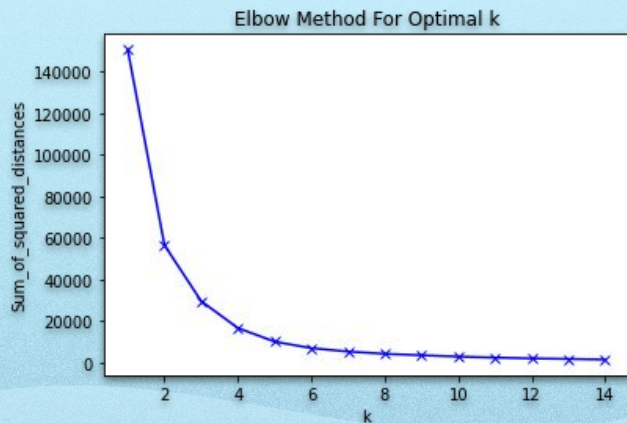
— Limpieza —

1. Checar valores nulos y seleccionar variables relevantes.

	No. NaN	%
Pclass	0	0.000000
Name	0	0.000000
Sex	0	0.000000
Age	177	0.198653
SibSp	0	0.000000
Parch	0	0.000000
Ticket	0	0.000000
Fare	0	0.000000
Cabin	687	0.771044
Embarked	2	0.002245

- Convertir 'Cabin' a booleano.
- Convertir 'Sex' a booleano.
- Variable dummy para 'Embarked'.
- Promedio para 'Fare'.
- Predicción para 'Age' mediante agrupamiento de edades.
- Eliminar las columnas de:
 - Ticket.
 - Name

— Age Groups —



- Uso de *KMeans* para obtener el número óptimo de clusters (age groups) = 4
- Uso de funciones de numpy + *Kmeans* para obtener rangos
- *Age Groups* Finales:

```
[ 0.   13.16 27.67 43.52  inf]
```

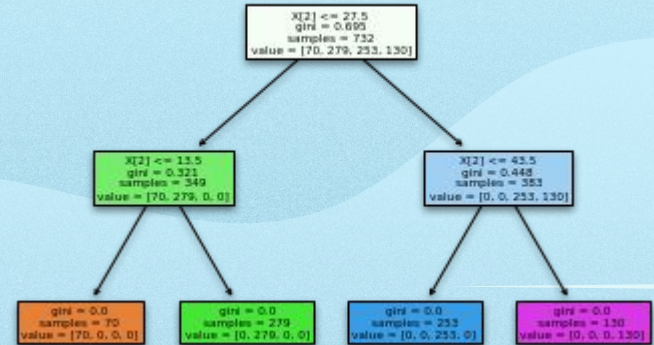
- Categorizamos los que tienen Age (*pd.cut*).

— Age Groups —

2. Predicción de edades.

- Combinamos datasets de *train.csv* y *test.csv* de *Kaggle* para más datos y mejor modelo (para ageGroups).
- Utilizamos todos los datos que tuvieran *Age* y entrenamos nuestro modelo.
- Dividimos entre conjunto de prueba y entrenamiento (70-30).
- Usamos el Decision Tree Classifier para clasificar edades.

```
withAgesFilled = bothDf[bothDf['Age'].isnull()==False]  
withoutAgesFilled = bothDf[bothDf['Age'].isnull()==True]
```



— Age Groups Métricas —

Se utilizaron las siguientes métricas para medir el éxito del modelo:

- *Accuracy_score*
- *Confusion_matrix*

Estas son las mejores métricas para medir rendimiento de modelos con variables *output* categóricas.

Resultados:

- 100% accuracy en *train subset*
- 100% accuracy en *test subset*
- Cuando se predice para los datos que no cuentan con edad, no se puede obtener puntaje de accuracy debido a que se desconocen los datos reales

```
[[ 70   0   0   0]
 [  0 279   0   0]
 [  0   0 253   0]
 [  0   0   0 130]]
1.0
[[ 29   0   0   0]
 [  0 126   0   0]
 [  0   0 103   0]
 [  0   0   0  56]]
1.0
```

— Age Groups Métricas —

Convertimos a variable *dummy* ya con los resultados (lo pudimos mantener de 1-4 porque es nominal).

Después de la limpieza, separamos de nuevo los *datasets* de *train.csv* y *test.csv* pero con *ageGroups* y sólo las columnas relevantes para el modelo final.

PassengerId	Survived	Pclass	Sex	SibSp	Parch	Fare	Has_Cabin	emb_C	emb_Q	emb_S	ag_age1	ag_age2	ag_age3	ag_age4	
0	1	0.0	3	0	1	0	7.2500	0	0	0	1	0	1	0	0
1	2	1.0	1	1	1	0	71.2833	1	1	0	0	0	0	1	0
2	3	1.0	3	1	0	0	7.9250	0	0	0	1	0	1	0	0
3	4	1.0	1	1	1	0	53.1000	1	0	0	1	0	0	1	0
4	5	0.0	3	0	0	0	8.0500	0	0	0	1	0	0	1	0

— Construcción del modelo —

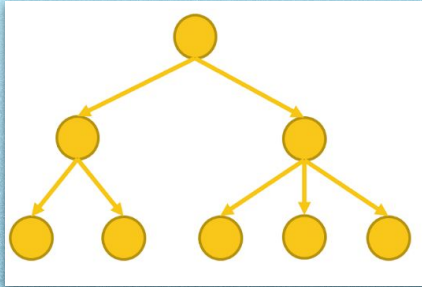
- ❑ Como nuestra variable dependiente es categórica, debemos escoger modelos con estas características.
- ❑ Como variables independientes tenemos numéricas y categóricas convertidas a *dummy*.
- ❑ Para nosotros fue importante explorar diferentes modelos antes de escoger y optimizar uno.
- ❑ Utilizamos las mismas métricas que en nuestro modelo de *ageGroup* para medir el rendimiento de cada uno de nuestros modelos a probar.

	Pclass	Sex	SibSp	Parch	Fare	Has_Cabin	emb_C	emb_Q	emb_S
0	3	0	1	0	7.2500	0	0	0	1
1	1	1	1	0	71.2833	1	1	0	0
2	3	1	0	0	7.9250	0	0	0	1
3	1	1	1	0	53.1000	1	0	0	1
4	3	0	0	0	8.0500	0	0	0	1

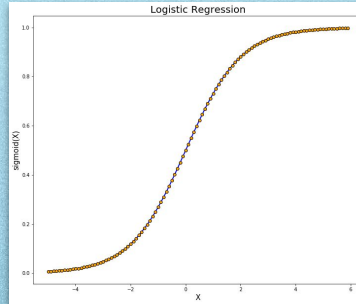
	ag_age1	ag_age2	ag_age3	ag_age4
0	0	1	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
0	0.0			

— Construcción del modelo —

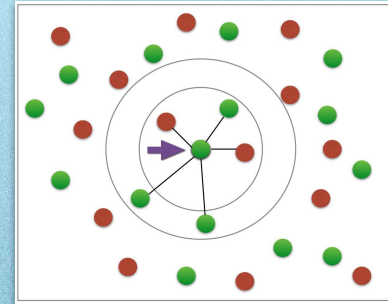
- ❑ Modelos a probar: KNeighbors Classifier, Logistic Regression, Decision Tree Classifier.
- ❑ Utilizamos el mismo parámetro *random_state* y training split para mantener la consistencia y poder comparar.



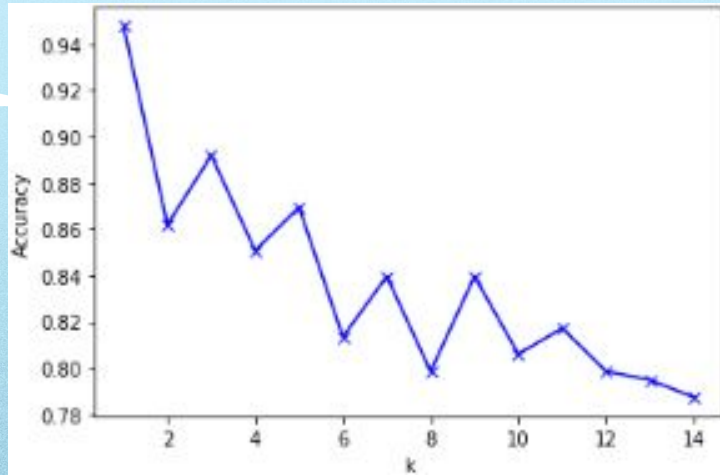
VS



VS



— KNeighbors Classifier —




Accuracy para todos los *KNeighbors*

```
[[343  49]
 [ 93 138]]
0.7720706260032103
[[140  17]
 [ 37  74]]
0.7985074626865671
```

Buen modelo, no
tanta accuracy
como quisiéramos

— Logistic Regression —



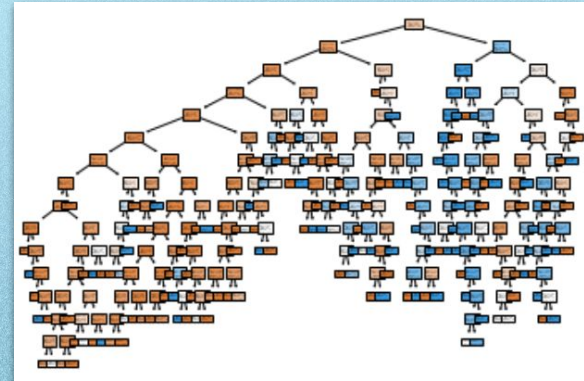
```
[[337  55]  
 [ 69 162]]  
0.8009630818619583  
[[135  22]  
 [ 30  81]]  
0.8059701492537313
```

Buen modelo, no muestra señales de
sobreajuste

— Decision Tree —

- Mejor modelo hasta ahora
- Muestra señales de sobreajuste: *accuracy* muy alto en *training* y no muy alto en el *test subset*, igualmente el árbol de decisiones muestra muchas ramas y profundidad
- Con la optimización podemos incrementar el *accuracy*.

```
> [[387  5]
    [ 21 210]]
0.9582664526484751
[[127  30]
 [ 35  76]]
0.7574626865671642
```

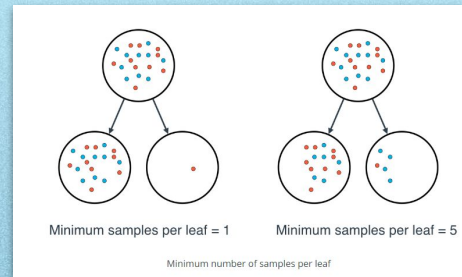
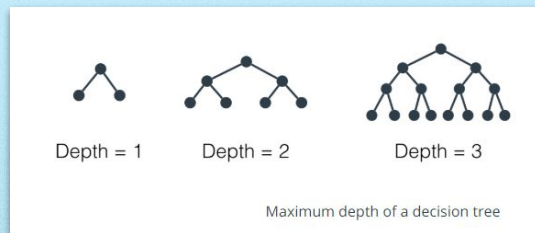


— Optimización —

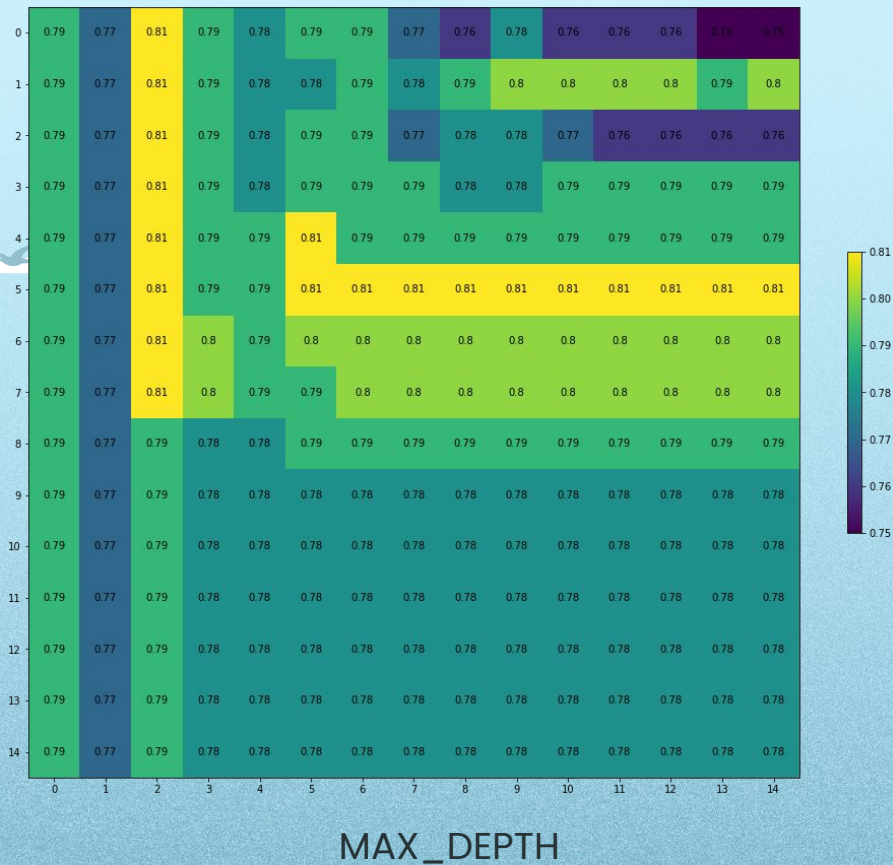
En cuanto a Decision Tree Classifier:

Contamos con **sobreajuste**, para esto hay dos hiper parámetros que se pueden modificar para tener un mejor modelo: **Min_Sample_Leaf** y **Max_Depth**.

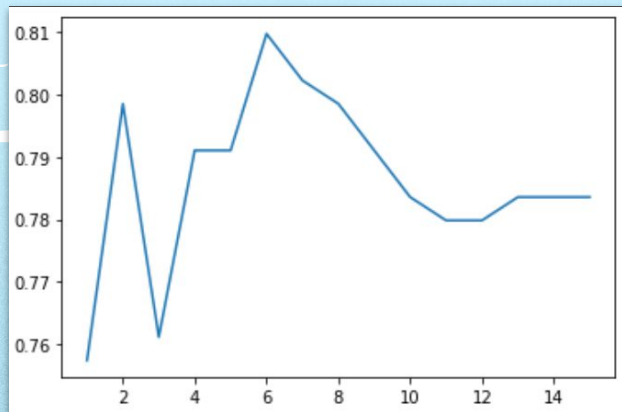
- **Min_Sample_Leaf** nos dice que tantos ejemplos una rama debe de tener para generarla. Esto es bueno tenerlo a un nivel bajo para no terminar con ramas con solo 1-3 situaciones.
- Lo mismo va para nuestro hiper parámetro **Max_Depth**, el cual limita la profundidad de las ramas que igualmente puede hacer mucho ruido y causar sobreajuste.



MIN_SAMPLE_LEAF



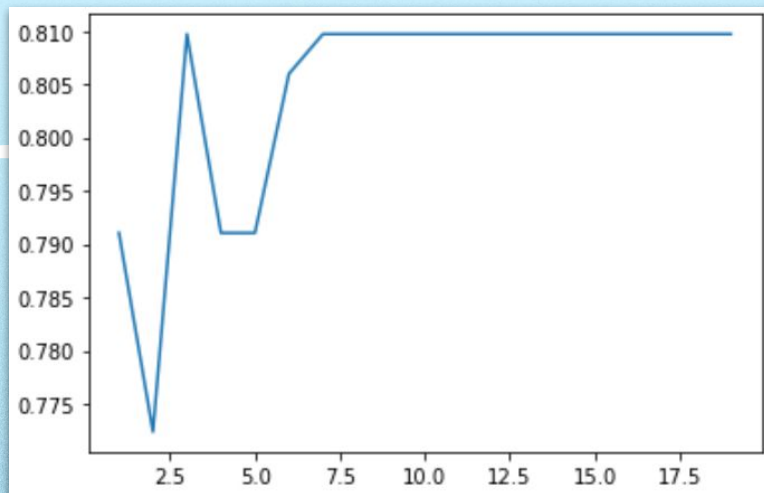
— Optimización Min Sample Leaf —



```
{1: 0.7574626865671642,  
2: 0.7985074626865671,  
3: 0.7611940298507462,  
4: 0.7910447761194029,  
5: 0.7910447761194029,  
6: 0.8097014925373134,  
7: 0.8022388059701493,  
8: 0.7985074626865671,  
9: 0.7910447761194029,  
10: 0.7835820895522388,  
11: 0.7798507462686567,  
12: 0.7798507462686567,  
13: 0.7835820895522388,  
14: 0.7835820895522388,  
15: 0.7835820895522388}
```

Vemos que nuestro mejor *accuracy* es con 6 de *min_sample_leaf*. Lo que significa que necesita mínimo 6 ejemplos de ese caso para generar una nueva rama.

— Optimizacion Max Depth —



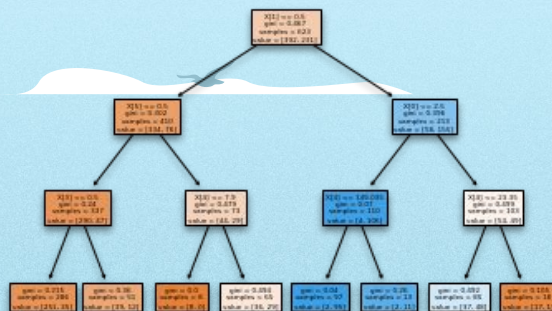
```
{1: 0.7910447761194029,  
2: 0.7723880597014925,  
3: 0.8097014925373134,  
4: 0.7910447761194029,  
5: 0.7910447761194029,  
6: 0.8059701492537313,  
7: 0.8097014925373134,  
8: 0.8097014925373134,  
9: 0.8097014925373134,  
10: 0.8097014925373134,  
11: 0.8097014925373134,  
12: 0.8097014925373134,  
13: 0.8097014925373134,  
14: 0.8097014925373134,  
15: 0.8097014925373134,  
16: 0.8097014925373134,  
17: 0.8097014925373134,  
18: 0.8097014925373134,  
19: 0.8097014925373134}
```

- Vemos que sube el *accuracy*, después baja y vuelve a subir
- Tratando de optimizar es el sobreajuste: el mínimo *depth* que nos de el mejor *accuracy score* = 3.

— Modelo Final —

#FINAL MODEL

```
survivedOrNot = DecisionTreeClassifier(random_state=42, min_samples_leaf = 6, max_depth=3)  
survivedOrNot.fit(x_train, y_train)
```



```
[[141  16]  
 [ 35  76]]  
0.8097014925373134  
[[351  41]  
 [ 77 154]]  
0.8105939004815409
```

- Terminamos con un modelo más preciso y sin sobreajuste. Como podemos ver en el árbol y los *accuracy_scores*
- Mejor que cualquiera de los otros modelos que comparamos.

— Pruebas Kaggle —

- Preparamos predicciones y subimos a Kaggle
- Enviamos la de todos nuestros modelos que comparamos
- Nuestro modelo final es el que tiene mejor score

```
forTestModelx = test.drop(['PassengerId','Survived'],axis=1)

predictions = survivedOrNot.predict(forTestModelx)

predictDf = pd.DataFrame(predictions,columns=['Survived'])

finalCsv = pd.concat([test.drop(['Survived'],axis=1), predictDf], axis=1)
finalCsv.head(20)

finalCsv['Survived'] = finalCsv['Survived'].astype(int)
finalCsv[['PassengerId','Survived']].to_csv('export.csv',index=False)
files.download('export.csv')
```

YOUR RECENT SUBMISSION



export (8).csv

Submitted by Nicolas CV · Submitted 2 days ago

Score: 0.77990

↓ Jump to your leaderboard position

Conclusion

1. Analizamos los datos
2. Limpiamos
3. Analizamos variables relevantes
4. Construimos y comparamos tres modelos
5. A partir de esos tres modelos escogimos el mejor y lo optimizamos para mejor resultados
6. **Exito!**

