# Práctica 2

# Limpieza y validación de los datos

Gisele Guadalupe Almeida dos Santos Maia

Diciembre 2019.

Universitat Oberta de Catalunya.

Máster de Ciencia de Datos (Data Science).

Tipología y ciclo de Vida de los datos

**Siguiendo las principales etapas de un proyecto analítico, las diferentes tareas a realizar (y justificar) son las siguientes:**

**1. Descripción del dataset. ¿Por qué es importante y qué pregunta/problema pretende responder?**

*Respuesta: He elegido el data set Titanic: Machine Learning from Disaster disponible en Kaggle: (https://www.kaggle.com/c/titanic).*

*Tenemos 2 data set principales, el de teste (**test.csv**) y el de entrenamiento (**train.csv**).*

*El archivo train.csv tenemos 12 variables (columnas) y 891 registros (filas) y en archivo test.csv tenemos 11 variables y 419 registros. La columna de nombre Survived es la diferencia entre las columnas de los dos archivos.*

*La descripción de las variables:*

*PassengerId: el número, la clave de identificación de la persona;*

*Survived: es la sobrevivencia, se el pasajero ha sobrevivido o no. Se = 1 el pasajero ha sobrevivido, se = 0 no ha sobrevivido;*

*Pclass: es la clase del billete: 1º*, 2º o 3º donde se igual a 1 es de la 1º clase, se igual a 2 es de la 2º clase y se igual a 3 es de la 3º clase;

*Name: es el nombre del pasajero, ejemplo: Harris, Mr. Walter, se es mujer y esposa de algun pasajero, tiene el nombre de los dos. Ejemplo: Braf, Miss. Elin Ester Maria;*

*Sex: el sexo del pasajero: female = mujer o male=hombre;*

*Age: la edad del pasajero: 26 años;*

*SibSp: número de hermanos o cónyuge a bordo en Titanic;*

*Parch: número de parientes (niños) a bordo en Titanic;*

*Ticket: el número del billete. Ejemplo: 330911;*

*Fare: el precio pago por el billete;*

*Cabin: el número de la cabina en Titanic;*

*Embarked: el nombre del puerto de embarcación. C = Cherboung, Q = Queenstorwn, S = Southampton.*

*Cómo en el archivo de teste no hay la variable de Survived, la idea es intentar hacer la previsibilidad de las personas del archivo de teste sobrevivieran o no. Y para esto vamos a llevar en consideración características como: género, clase social, edad y las variables que sean posible o que sean importantes en el facto de sobrevivir o no al desastre, o sea, identificar las variables que más influyen en la sobrevivencia. Para esto es importante hacer un análisis de las variables para identificar si es posible o no su utilización.*

**2. Integración y selección de los datos de interés a analizar.**

*Para el análisis de los datos, vamos a utilizar solamente el archivo train donde contiene las 12 variables.*

*Hacemos la importación de las librarías que vamos a utilizar, miramos la cantidad de líneas y columnas:*

```
# Leer lo archivo
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
pd.options.display.max_columns = None
train = pd.read_csv('train.csv')
```

```
train.shape
```

```
(891, 12)
```

*Las variables que tenemos y sus formatos:*

```
train.head()
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| **1** | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| **2** | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| **3** | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| **4** | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

```
train.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

*Miramos un resumen de las variables numéricas:*

```
train.describe().transpose()
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **PassengerId** | 891.0 | 446.000000 | 257.353842 | 1.00 | 223.5000 | 446.0000 | 668.5 | 891.0000 |
| **Survived** | 891.0 | 0.383838 | 0.486592 | 0.00 | 0.0000 | 0.0000 | 1.0 | 1.0000 |
| **Pclass** | 891.0 | 2.308642 | 0.836071 | 1.00 | 2.0000 | 3.0000 | 3.0 | 3.0000 |
| **Age** | 714.0 | 29.699118 | 14.526497 | 0.42 | 20.1250 | 28.0000 | 38.0 | 80.0000 |
| **SibSp** | 891.0 | 0.523008 | 1.102743 | 0.00 | 0.0000 | 0.0000 | 1.0 | 8.0000 |
| **Parch** | 891.0 | 0.381594 | 0.806057 | 0.00 | 0.0000 | 0.0000 | 0.0 | 6.0000 |
| **Fare** | 891.0 | 32.204208 | 49.693429 | 0.00 | 7.9104 | 14.4542 | 31.0 | 512.3292 |

*Así, podemos comprobar que la variable PassengerId no hay valores vacíos, la variable Survived tampoco hay valores vacíos y tenemos registros de 0 hasta 1. La variable Pclass tampoco hay valores vacíos, tenemos valores de 1 hasta 3. Ya la variable Age hay 177 (19,9%) registros nulos con personas de 0 hasta 80 años, a variable SibSp (hermanos o cónyuge) hay registros de 0 hasta 8 pero no hay registros vacíos, así como la variable Parch (número de niños a bordo) y Fare (precio) que tampoco hay registros vacíos. Pero, en Parch hay registros de 0 hasta 6 y el precio entre 0 y 512.329200.*

```
# Miramos se hay valores nulos también en las variables de texto
train.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

*La variable Cabin hay 687 registros vacíos y variable Embarked 2. Cómo la variable cabin hay 77,1% de las variables vacías, no vas a ser una variable muy útil ya que tenemos pocos registros, así podemos borrarla.*

**3. Limpieza de los datos.**

**3.1. ¿Los datos contienen ceros o elementos vacíos? ¿Cómo gestionarías cada uno de estos casos?**

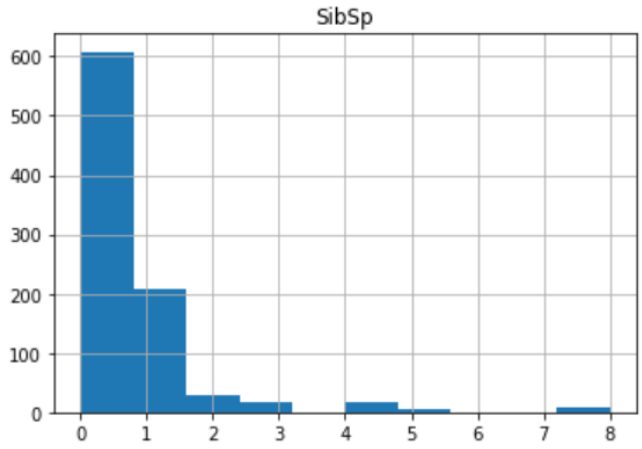
*La variable SibSp (hermanos o cónyuge), hay la siguiente distribución:*

```
SibSp = pd.crosstab(index=train["SibSp"],
                    columns="count")
SibSp/SibSp.sum()
```

| col_0 | count |
|---|---|
| SibSp | |
| 0 | 0.682379 |
| 1 | 0.234568 |
| 2 | 0.031425 |
| 3 | 0.017957 |
| 4 | 0.020202 |
| 5 | 0.005612 |
| 8 | 0.007856 |

```
train.hist(column='SibSp')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x
000001E0160775F8>]],
      dtype=object)
```




*68% de los registros están cómo 0 que no sabemos se es porque no son casados o no tiene hermanos o porque no tenía su marido o esposa o hermanos a bordo.*

*Así, vamos a hacer una integración donde 0 es: no tenía hermanos o cónyuge a bordo y > 0 con hermanos o cónyuge a bordo.*

```
def g(x):
  if x['SibSp'] > 0: return 'Con hermanos o cónyuge'
  else: return 'Sin hermanos o cónyuge'

train['SibSp_2'] = train.apply(g, axis=1)
```

```
SibSp_2 = pd.crosstab(index=train["SibSp_2"],
                              columns="count")
SibSp_2/SibSp_2.sum()
```

| col_0 | count |
|---|---|
| SibSp_2 | |
| Con hermanos o cónyuge | 0.317621 |
| Sin hermanos o cónyuge | 0.682379 |

*Miramos la distribución de la variable Parch donde tenemos el número de niños a bordo*

```
# Miramos el percentual de niños a bordo
Parch = pd.crosstab(index=train['Parch'],
                          columns='count')
Parch/Parch.sum()
```

| col_0 | count |
|---|---|
| Parch | |
| 0 | 0.760943 |
| 1 | 0.132435 |
| 2 | 0.089787 |
| 3 | 0.005612 |
| 4 | 0.004489 |
| 5 | 0.005612 |
| 6 | 0.001122 |

*76% de los pasajeros no tienen hijos o no están con ellos a bordo.*

*Por lo tanto, vamos a hacer también una integración, donde 0 no están con hijos a bordo y > 0 están con hijos a bordo. Ya que tampoco sabemos si 0 es que no tiene hijos o no están con ellos.*

```
def h(x):
  if x['Parch'] > 0: return 'Con hijos a bordo'
  else: return 'Sin hijos a bordo'

train['Parch_2'] = train.apply(h, axis=1)
```

```
Parch_2 = pd.crosstab(index=train["Parch_2"],
                      columns="count")
Parch_2/Parch_2.sum()
```

| col_0 | count |
|---|---|
| Parch_2 | |
| Con hijos a bordo | 0.239057 |
| Sin hijos a bordo | 0.760943 |

*Los datos vacíos en la Age (177), vamos a imputar los valores perdidos de Age así como la variable Embarked (2 vacíos).*

*Para la variable Embarked que tenemos solamente 2 registros vacíos, vamos a mirar cuál de los 3 puertos hay más pasajeros y así sustituir los vacíos por el puerto con un % más grande:*

```
Embarked_2 = pd.crosstab(index=train['Embarked'],
                         columns='count')
Embarked_2
```

| col_0 | count |
|---|---|
| Embarked | |
| C | 168 |
| Q | 77 |
| S | 644 |

```
Embarked_2/Embarked_2.sum()
```

| col_0 | count |
|---|---|
| Embarked | |
| C | 0.188976 |
| Q | 0.086614 |
| S | 0.724409 |

*Así, remplazamos los registros vacíos en Embarked por "S" que contiene 72,4% de los registros.*

```
# Remplazar null por 'S' en la columna Emparked
train = train.fillna({'Embarked': 'S'})
```

```
Embarked_2 = pd.crosstab(index=train['Embarked'],
                         columns='count')
Embarked_2
```

| col_0 | count |
|---|---|
| Embarked | |
| C | 168 |
| Q | 77 |
| S | 646 |

*Ahora vamos a imputar los datos vacíos de la edad. Utilizamos el RandomForestRegressor que es una metodología de clasificación y regresión para predecir valores pero, el valor elegido tiene relación con los otros. Así, tenemos el input que tenga relación con las edades de los otros pasajeros.*

```
# Para imputar los vacíos de la edad, vamos a utilizar el RandomForestRegressor de la librería sklearn
from sklearn.ensemble import RandomForestRegressor
```

```
# Separamos los datos de los que tienen la edad y los que no
train_edad = train[pd.isnull(train['Age']) == False]
train_sin_edad = train[pd.isnull(train['Age'])]
```

```
# Aquí se hace las dumies de algunas variables que vamos a utilizar para el input
puerto_edad = pd.get_dummies(train_edad['Embarked'])
puerto_sin_edad = pd.get_dummies(train_sin_edad['Embarked'])
```

```
SibSp_2_edad = pd.get_dummies(train_edad['SibSp_2'])
SibSp_2_sin_edad = pd.get_dummies(train_sin_edad['SibSp_2'])
```

```
Parch_2_edad = pd.get_dummies(train_edad['Parch_2'])
Parch_2_sin_edad = pd.get_dummies(train_sin_edad['Parch_2'])
```

```
Sex_edad = pd.get_dummies(train_edad['Sex'])
Sex_sin_edad = pd.get_dummies(train_sin_edad['Sex'])
```

```
x = ['Pclass', 'Fare', 'Age']
```

```
train_edad = train_edad[x]
train_sin_edad = train_sin_edad[x]
```

```
train_edad = pd.concat([train_edad, puerto_edad, SibSp_2_edad, Parch_2_edad, Sex_edad], axis = 1)
train_sin_edad = pd.concat([train_sin_edad, puerto_sin_edad, SibSp_2_sin_edad, Parch_2_sin_edad, Sex_sin_edad], axis = 1)
```

*Ahora vamos a hacer la clasificación con los datos que tenemos la edad con las variables que pueden hacer diferencia, que son las variables que están en el dataframe train_edad y también train_sin_edad.*

```
# Variables para acer la clasificación
variables = ['Pclass', 'Fare', 'C', 'Q', 'S', 'Con hermanos o cónyuge', 'Sin hermanos o cónyuge',
             'Con hijos a bordo', 'Sin hijos a abordo', 'female', 'male']
```

```
edad_model = RandomForestRegressor()
```

```
edad_model.fit(train_edad[variables], train_edad['Age'])
```

```
C:\Users\gisel\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators wil
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

```
predict_age = edad_model.predict(X = train_sin_edad[variables])
```

```
predict_age
```

```
array([34.37916667, 31.39711783, 16.85714286, 33.94702381, 19.39      ,
       27.1948029 , 22.825     , 24.30583333, 25.11      , 31.84      ,
       31.45694711, 30.05833333, 24.30583333, 24.        , 40.13333333,
       37.9       , 19.95833333, 27.1948029 , 31.45694711, 21.29916667,
       31.45694711, 31.45694711, 27.1948029 , 29.19018759, 22.2       ,
       31.45694711, 43.08154762,  7.53809524, 24.62      , 30.6442674 ,
       26.98931097, 18.5       , 23.        , 56.425     ,  7.35      ,
       26.88333333, 29.8       , 39.2       , 26.4       , 43.08154762,
       24.30583333, 18.5       , 39.72083333, 27.1948029 , 21.6       ,
       21.9       , 15.01666667, 26.4       , 30.6442674 , 35.86666667,
       43.08154762, 24.30583333, 50.445     , 24.30583333, 34.56635309,
       61.38333333, 37.9       , 46.435     , 24.30583333, 28.29166667,
       38.1       , 31.45694711, 27.8       , 18.5       , 22.2       ,
       40.64166667, 27.1948029 , 27.2       , 44.43333333, 33.94702381,
       19.39      , 19.39      , 30.05833333, 14.425     , 24.30583333,
       31.8       , 27.1948029 , 28.29821429, 21.6       , 27.1948029 ,
       25.275     , 34.56635309, 24.46666667, 31.84      , 30.6442674 ,
       43.08154762, 27.2       , 20.41851852, 19.21666667, 31.45694711,
       39.2       , 43.08154762, 31.45694711, 34.56635309, 28.29821429,
       30.6442674 , 46.3       , 34.56635309, 21.6       , 19.21666667,
```

```
# Como quedamos con números en el formato float, vamos cambiarlos para números enteros
train_sin_edad['Age'] = predict_age.astype(int)
```

```
# Y luego juntamos los dos dataframes
train2 = train_edad.append(train_sin_edad)
```

```
train2.head()
```

| | Pclass | Fare | Age | C | Q | S | Con hermanos o cónyuge | Sin hermanos o cónyuge | Con hijos a bordo | Sin hijos a abordo | female | male |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 7.2500 | 22.0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| **1** | 1 | 71.2833 | 38.0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| **2** | 3 | 7.9250 | 26.0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **3** | 1 | 53.1000 | 35.0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **4** | 3 | 8.0500 | 35.0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

```
# Como quedamos con números en el formato float, vamos cambiarlos para números enteros
train_sin_edad['Age'] = predict_age.astype(int)
```

```
# Y luego juntamos los dos dataframes
train2 = train_edad.append(train_sin_edad)
```

```
train2.reset_index(inplace=True)
train2.drop('index',inplace=True,axis=1)
```

```
train2.head()
```

| | Pclass | Fare | Age | C | Q | S | Con hermanos o cónyuge | Sin hermanos o cónyuge | Con hijos a bordo | Sin hijos a abordo | female | male |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 7.2500 | 22.0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| **1** | 1 | 71.2833 | 38.0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| **2** | 3 | 7.9250 | 26.0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **3** | 1 | 53.1000 | 35.0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **4** | 3 | 8.0500 | 35.0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

```
# Unir los dos dataframes para quedar también con la edad que hicimos el input (age_fin)
train3 = train.join(train2, lsuffix='_inicial', rsuffix='_fin')
train3.head()
```

| | PassengerId | Survived | Pclass_inicial | Name | Sex | Age_inicial | SibSp | Parch | Ticket | Fare_inicial | Cabin | Embarked | SibSp_2 | Parch_2 | Pclass_fi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S | Con hermanos o cónyuge | Sin hijos a abordo | |
| **1** | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C | Con hermanos o cónyuge | Sin hijos a abordo | |
| **2** | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S | Sin hermanos o cónyuge | Sin hijos a abordo | |
| **3** | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S | Con hermanos o cónyuge | Sin hijos a abordo | |

```
# Borramos la columna Fare_fin y Pclass_fin ya que no hay diferencias entre los valores con la Fare_inicial y la Pclass_inicial
train3 = train3.drop(columns=['Fare_fin', 'Pclass_fin'])
```

```
# Renombrando columnas que no tenemos más iguales
train3 = train3.rename(columns={"Pclass_inicial": "Pclass", "Fare_inicial": "Fare"})
```

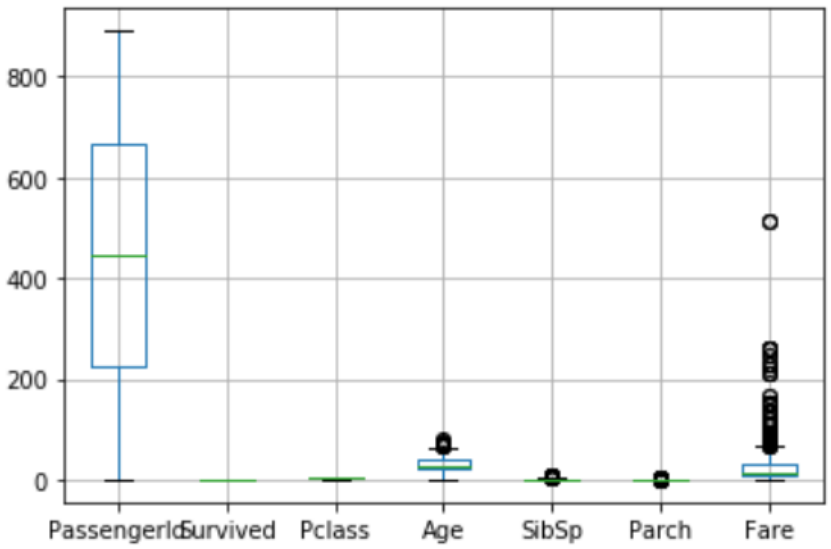
```
train3.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age_inicial',
       'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'SibSp_2',
       'Parch_2', 'Age_fin', 'C', 'Q', 'S', 'Con hermanos o cónyuge',
       'Sin hermanos o cónyuge', 'Con hijos a bordo', 'Sin hijos a abordo',
       'female', 'male'],
      dtype='object')
```

### 3.2. Identificación y tratamiento de valores extremos.

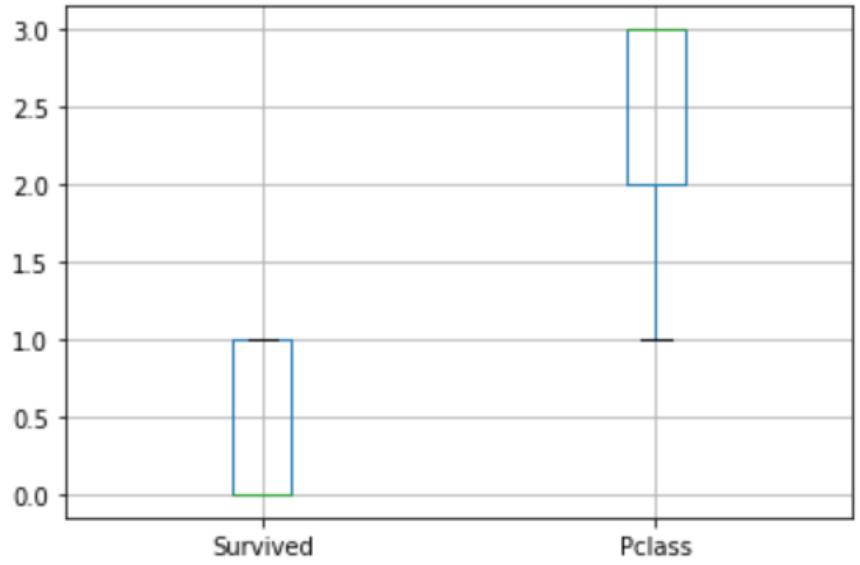
*Para las variables numéricas, vamos a comprobar los valores y los valores máximos y mínimos: con boxplot de todas las variables numéricas de los datos originales:*

```
# Comprobando outliers
train.boxplot()
plt.means=True
```



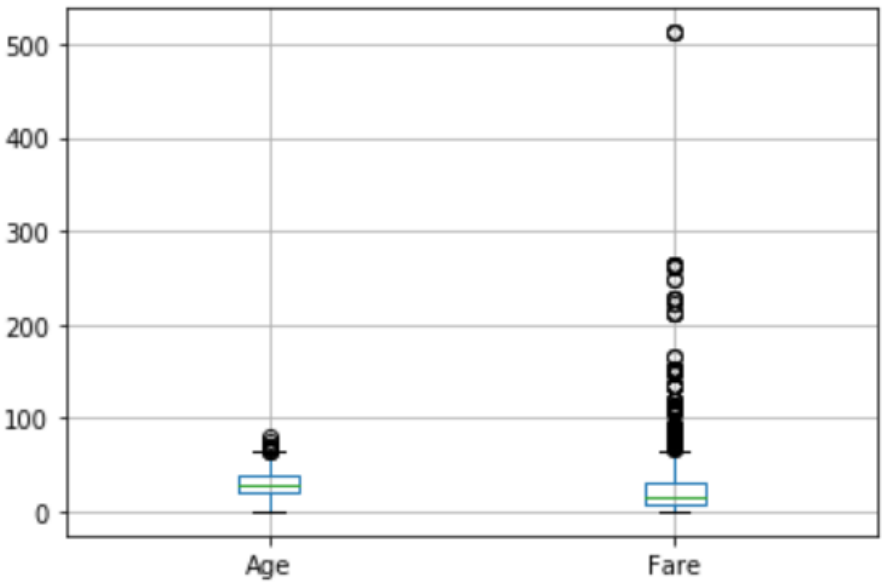

*Luego, cómo los rangos de las variables son distintos de la variable PassengerId, vamos a mirar 2 a 2 con los rangos más parecidos:*

```
# Comprobando outliers
train.boxplot(column=['Survived', 'Pclass'])
plt.means=True
```



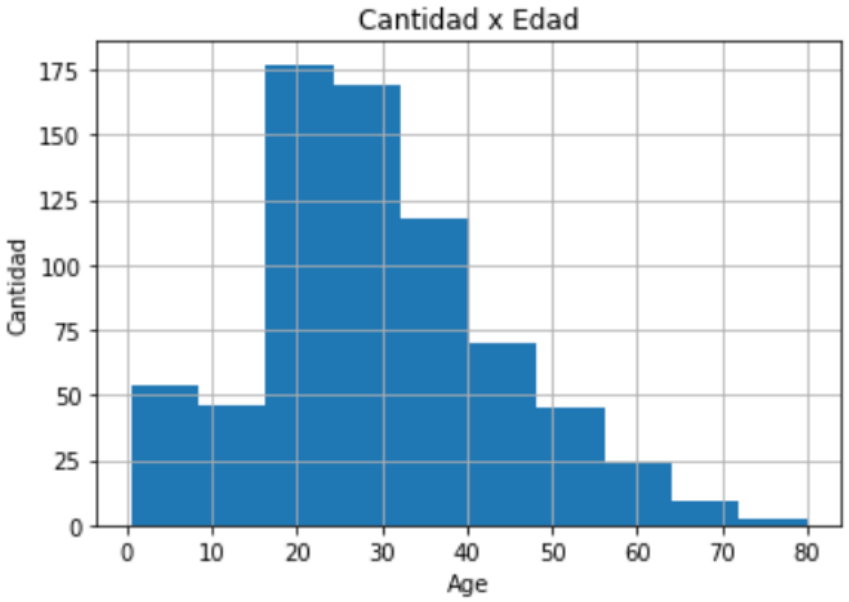

*En Survived (0 o 1) y Pclass (0 o 1 o 2), no tenemos outliers ya que las variables están en un rango específico.*

```
# Comprobando outliers
train.boxplot(column=['Age', 'Fare'])
plt.means=True
```



```
# Distribuición normal o no de l edad
train.hist('Age')
plt.title('Cantidad x Edad')
plt.xlabel('Age')
plt.ylabel('Cantidad')
```

```
Text(0, 0.5, 'Cantidad')
```

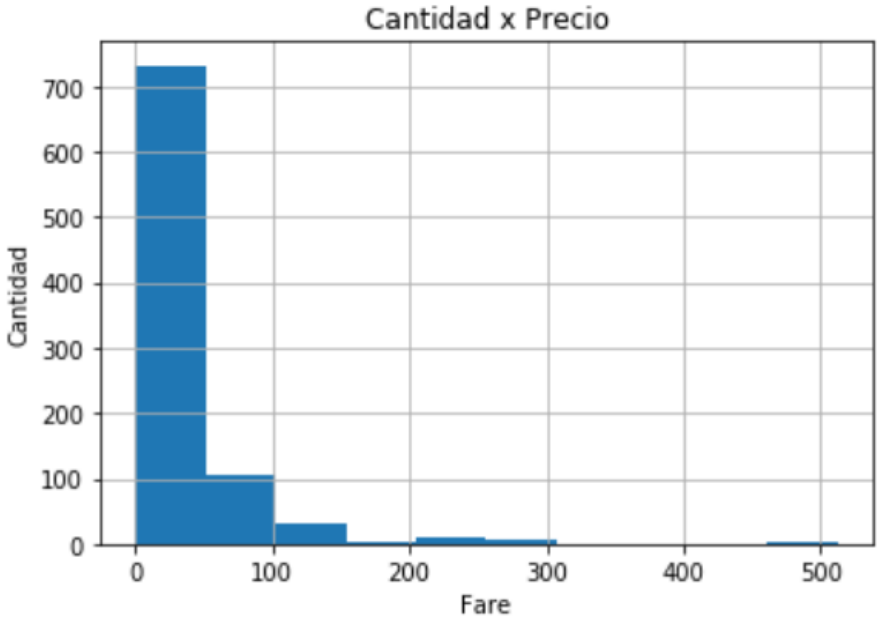



*La variable Age contiene un poco de outliers, y sigue la distribución más parecida on la gaussiana entonces no vamos a hacer ningún tipo de tratamiento, quedamos así.*

*Ya la distribución del precio es asimétrica:*

```
# Distribuición normal o no del precio
train.hist('Fare')
plt.title('Cantidad x Precio')
plt.xlabel('Fare')
plt.ylabel('Cantidad')
```

```
Text(0, 0.5, 'Cantidad')
```

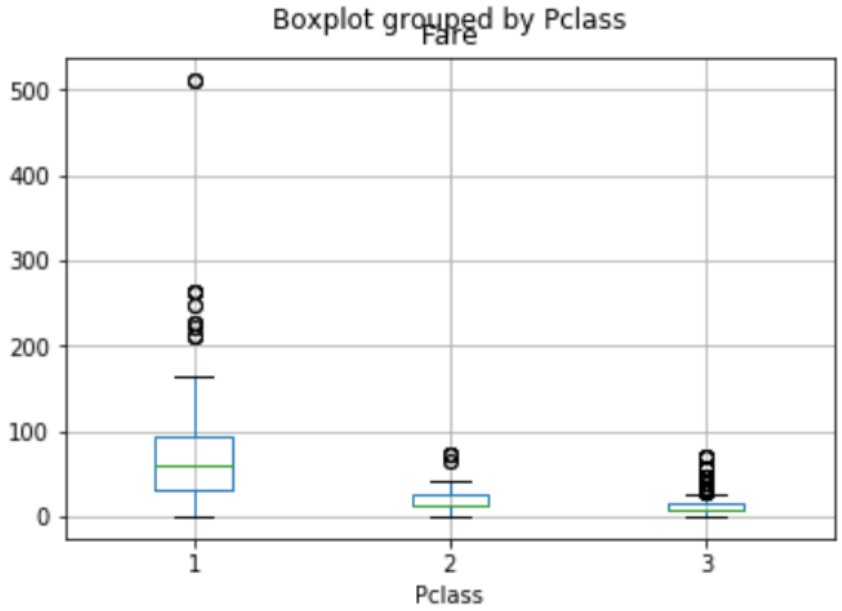



*El precio (Fare) tiene un rango muy grande, el mínimo es 0 y el máximo es 512.32992, y, el 75% de la base de datos tiene el valor de 31.000. Así, vamos a mirar su distribución y se hay diferencia de los precios por las clases:*

```
train.groupby('Pclass')['Fare'].describe()
```

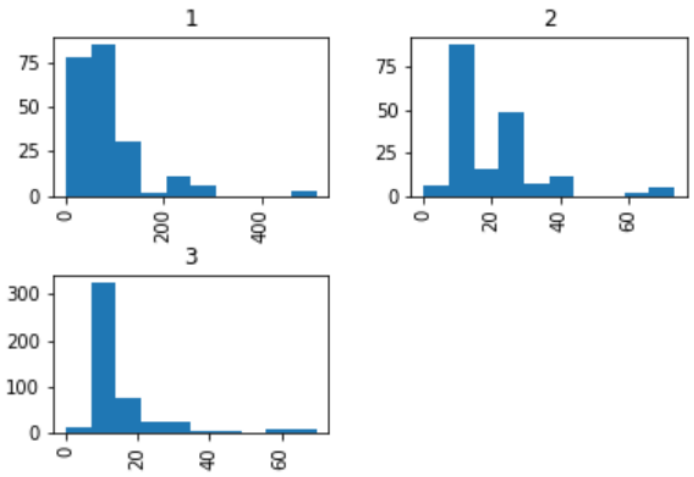
| Pclass | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| 1 | 216.0 | 84.154687 | 78.380373 | 0.0 | 30.92395 | 60.2875 | 93.5 | 512.3292 |
| 2 | 184.0 | 20.662183 | 13.417399 | 0.0 | 13.00000 | 14.2500 | 26.0 | 73.5000 |
| 3 | 491.0 | 13.675550 | 11.778142 | 0.0 | 7.75000 | 8.0500 | 15.5 | 69.5500 |

```
boxplot = train.boxplot(column=['Fare'],by='Pclass')
```




```
train['Fare'].hist(by=train['Pclass'])
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001C4D5CDC278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C4D5DA9198>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001C4D5DD5748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C4D5E05CF8>]],
      dtype=object)
```




*La primera clase es la que tiene una variabilidad más grande y con outliers muy distintos: 200 hasta 500 pero, la grande concentración está entre 20 y 100. La clase 2 y 3 los outliers están más concentrados. Y la distribución de los precios de la primera clase es asimétrica a la derecha (positiva), vamos a utilizar el método Inter cuartil para cambiaremos los outliers.*

```
# Outliers para la variable Precio (Fare), pero en el dataset ya con input de edad: train3
IQR = train3.Fare.describe()[6] - train3.Fare.describe()[4]
cuartil_sup_fare = train3.Fare.quantile(0.75) + IQR * 3
cuartil_inf_fare = train3.Fare.quantile(0.25) - IQR * 3
```

```
print('Rango del Fare con cuartil superior: {a} y cuartil inferior {b}'.format(a = cuartil_sup_fare, b = cuartil_inf_fare))
```

```
Rango del Fare con cuartil superior: 100.2688 y cuartil inferior -61.358399999999996
```

```
print('Cantidad de registros a cambiar: {}'.format(train3[train3['Fare']>100].count()[0]))
```

```
Cantidad de registros a cambiar: 53
```

*Vamos hacer el cambio ya que sabemos (por descibe() del Fare por Pclass visto arriba) que los precios más grandes que 100 son todos de la clase 1.*

*Vamos remplazar los 53 registros de outliers por la media de precios que los pasajeros de la clase 1 pagaran. Pero, quitamos no llevaremos en consideración en la media, los outliers. Visto que las medias son distintas.*

```
# Precio medio de la primera clase, no considerando los outliers
train[(train['Pclass']==1) & (train['Fare']<100)]['Fare'].mean()
```

```
50.70350306748466
```

```
train[train['Pclass']==1]['Fare'].mean()
```

```
84.15468749999992
```

*Además, es importante, para que no haya un único valor y una grande distribución concentrada en un único valor, vamos a dejar la variancia entre 45 y 60. Y los registros que están igual a 0, vamos a cambiarlos por la media del precio de acuerdo con cada clase.*

```
print(train3.groupby('Pclass')['Fare'].mean())
```

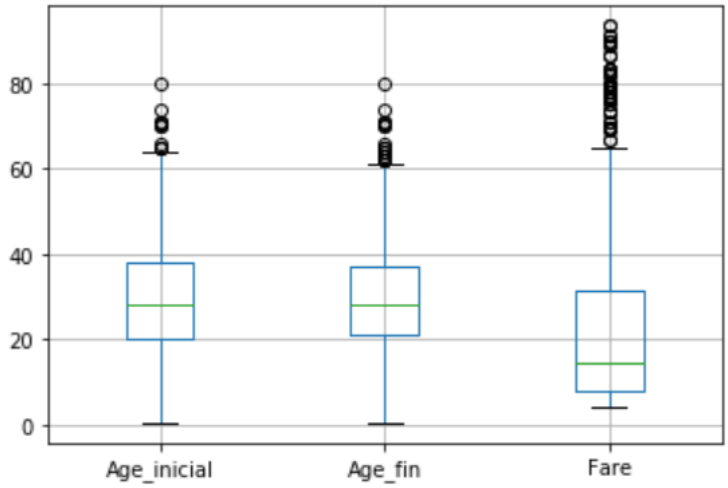
```
Pclass
1    84.154687
2    20.662183
3    13.675550
Name: Fare, dtype: float64
```

```
# Vamos a hacer una función para remplazar los outliers
def outlier_fare(df):
    df['Fare'] = np.where(df['Fare']>100, np.random.randint(45, 60,1)[0], df['Fare'])
    df['Fare'] = np.where(((df['Fare']==0) & (df['Pclass']==1)), 50, df['Fare'])
    df['Fare'] = np.where(((df['Fare']==0) & (df['Pclass']==2)), 20, df['Fare'])
    df['Fare'] = np.where(((df['Fare']==0) & (df['Pclass']==3)), 13, df['Fare'])
```

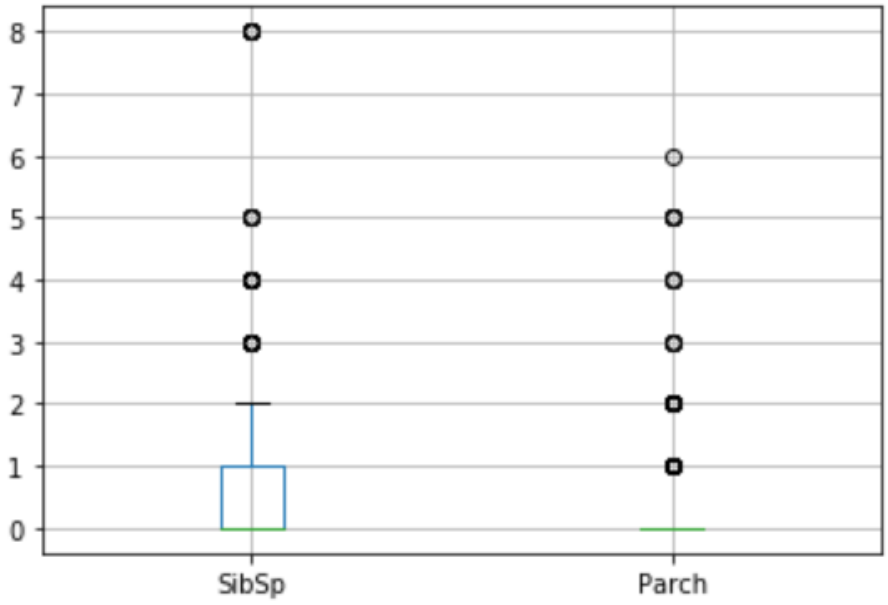
```
outlier_fare(train3)
```

*Ahora tenemos valores del precio hasta 100:*

```
# Comprobando outliers
train3.boxplot(column=['Age_inicial', 'Age_fin', 'Fare'])
plt.means=True
```




```
train.boxplot(column=['SibSp', 'Parch'])
plt.means=True
```




*La variable SibSP contiene algunos outliers de 3 hasta 8 pero, no vamos dejar solamente la variable SibSp_2 que hicimos antes. La variable Parch contiene más outliers pero, también utilizaremos la variable Parch_2 creada anteriormente.*

**4. Análisis de los datos.**

**5. Representación de los resultados a partir de tablas y gráficas.**

*Ahora vamos a quedar con las variables que vamos a analizar.*

```
train4 = train3.drop(columns=['Name', 'Age_inicial', 'SibSp', 'Parch', 'Ticket', 'Cabin'])
```

*Analizaremos cuales son las variables que contienen correlación con sobrevivir o no, y cuales tienen correlaciones entre ellas. Así, podemos elegir cuales son las variables más importantes para predecir la sobrevivencia en Titanic y así hacer la predicción en la base de datos de teste. Pero, para esto además de las variables para construcción del modelo, es necesario comprobar se las variables que vamos a utilizar se comportan de la misma manera. O sea: comprobar la normalidad y homogeneidad para que se pueda aplicar los modelos.*

```
# Miramos el % de los sobrevivientes
Survived2 = pd.crosstab(index=train['Survived'],
                        columns='%')
Survived2/Survived2.sum()
```

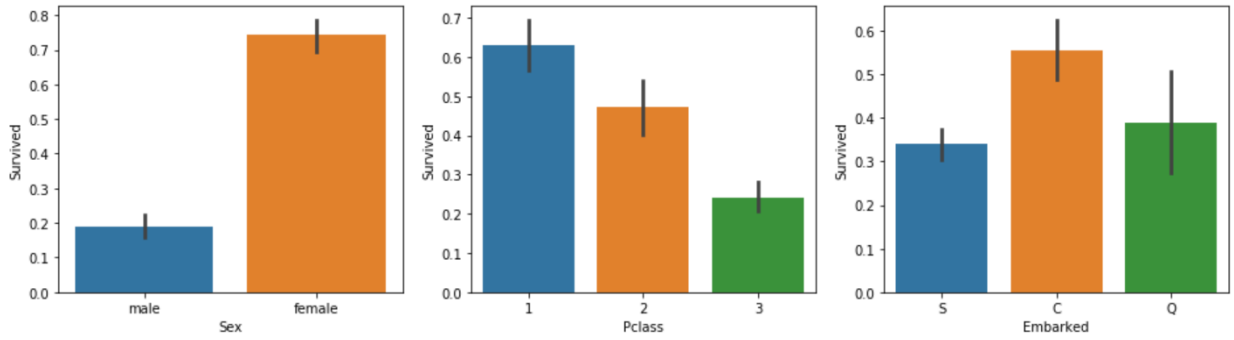
| col_0 | % |
|---|---|
| **Survived** | |
| **0** | 0.616162 |
| **1** | 0.383838 |

*Miramos ahora los grupos que contiene la probabilidad de sobrevivir más grande:*

```
# Mirando el grupo que tenía más probabilidad de subrevivir
fig, (axis1, axis2, axis3) = plt.subplots(1,3, figsize=(16,4))

sns.barplot(x='Sex', y='Survived', data=train4, ax=axis1)
sns.barplot(x='Pclass', y='Survived', data=train4, ax=axis2)
sns.barplot(x='Embarked', y='Survived', data=train4, ax=axis3)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4eae8d898>
```

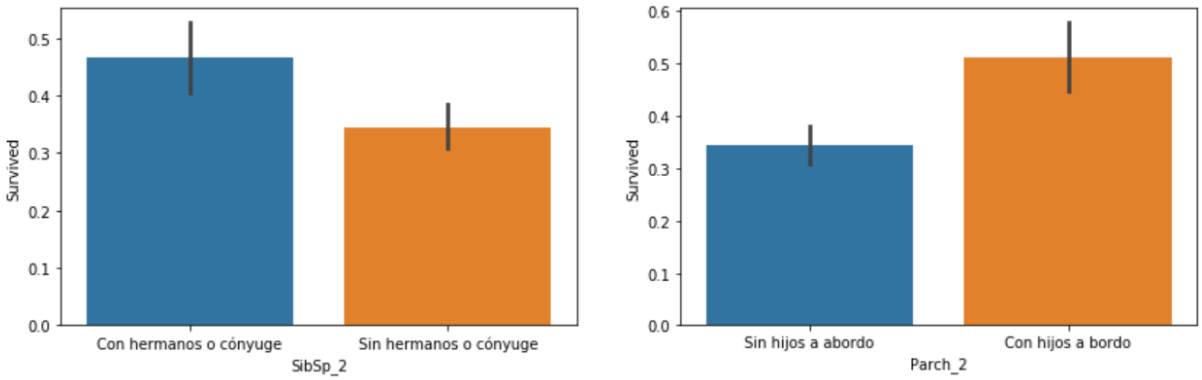



*Las mujeres tienen más oportunidad de sobrevivir que los hombres (74% x 19%), los pajareros de la primera clase también son los que tienen más oportunidad de sobrevivir (63% - primera clase, 47% - segunda clase y 24% tercera clase), así cómo los pasajeros que embarcaran por Cherboung (55%), los de Queenstorwn con 39% y de Southampton 34%.*

```
fig, (axis4, axis5) = plt.subplots(1,2, figsize=(14,4))

sns.barplot(x='SibSp_2', y='Survived', data=train4, ax=axis4)
sns.barplot(x='Parch_2', y='Survived', data=train4, ax=axis5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4eb0c1e48>
```



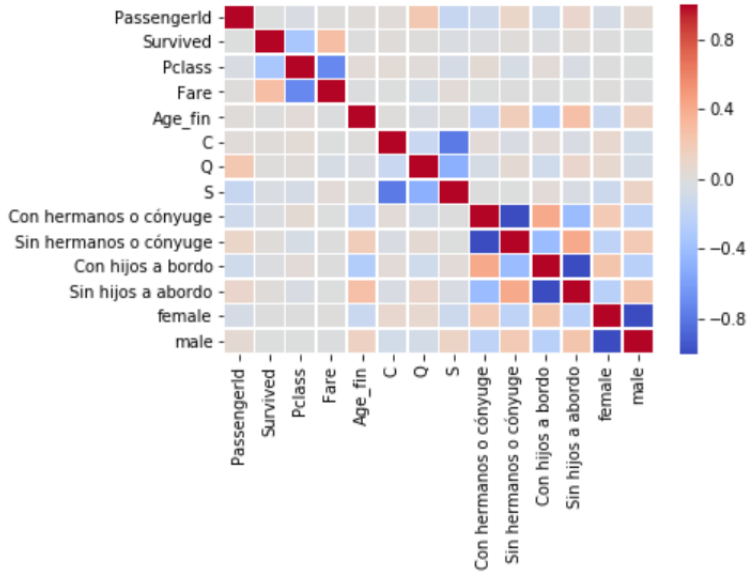

*También tienen una oportunidad de sobrevivir los pasajeros con hermanos o cónyuge (aproximadamente 48%) y los pasajeros con hijos a bordo (aproximadamente 51%).*

*Luego, el grupo con más probabilidad de sobrevivir son: las mujeres, la primera clase, los que embarcaran por Cherboung, los que tienen hermanos, o conyugue o hijos que también estaban a bordo.*

*Hacemos también una correlación entre las variables numéricas, a ver cuáles son las variables con correlaciones:*

```
# Correlación de las variables numéricas
sns.heatmap(train4.corr(), cmap='coolwarm', fmt='.2f', linewidths=0.5, vmax=1.0, linecolor='white')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4eaf7c400>
```



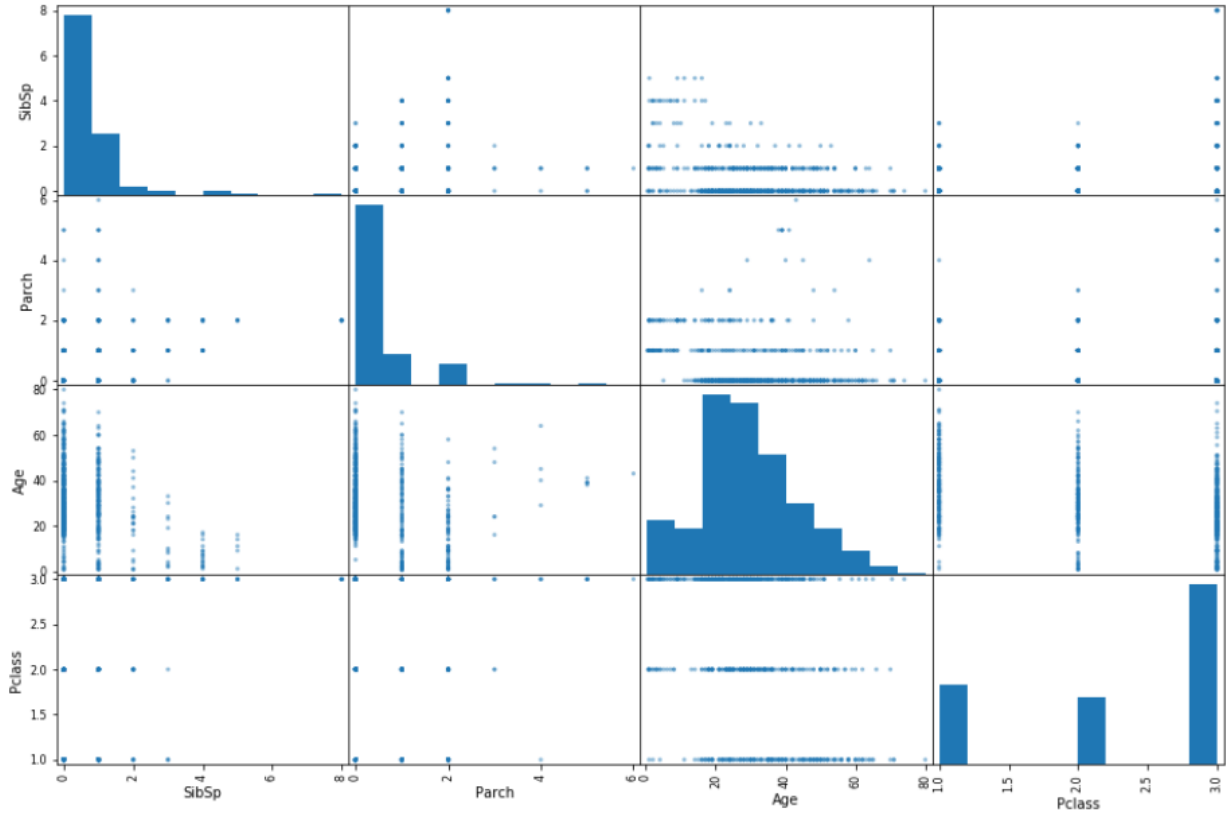

*Aquí miramos que la variable Survived hay una correlación inversamente proporcional al Pclass y proporcional al precio (Fare).*

*También podemos mirar que la correlación inversamente proporcional al Pclass y Fare, o sea, el precio es más alto para las menores clases (cómo ya sabemos: la primera clase es más cara) y la edad (Age_fin) tiene correlación también con el fato de no tener hijos a bordo y de no tener cónyuge*

*o hermanos a bordo, así como la correlación entre: sin hijos a bordo con sin cónyuge o hermanos a bordo, lo que hace sentido.*

*Vamos a mirar las variables numéricas en el dataframe original, haciendo un cruce dos a dos:*

```
# Haciendo un cruze con las variables
columns=['SibSp','Parch','Age', 'Pclass']
pd.plotting.scatter_matrix(train[columns], figsize=(15, 10))
```

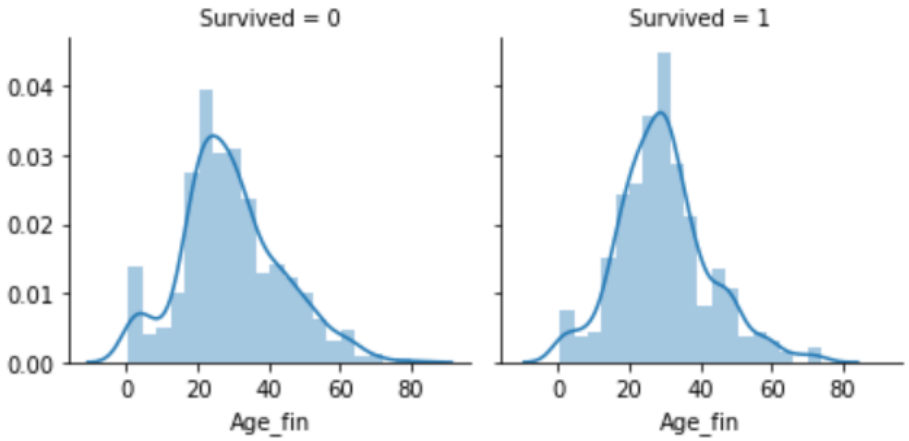



*Aquí podemos mirar que hay una concentración de las personas más viejas en la clase 1 y de las personas más jóvenes en la clase 3.*

*Vamos a mirar la relación de la edad con sobrevivir o no:*

```
# Comprobar si la edad influe en la probabilidad de subreviviencia
age_survived = sns.FacetGrid(train4, col='Survived')
age_survived.map(sns.distplot, 'Age_fin')
```

```
<seaborn.axisgrid.FacetGrid at 0x1c4eaffb080>
```




*Aquí miramos que las distribuciones son parecidas y prácticamente gaussianas. Los no sobrevivientes contienen un poco más de asimetría y con un pico de no sobreviviente con los más jóvenes (pico en los de 20 años) y un pico de sobreviviente en los 32 y 35 años.*

*Vamos también comprobar la correlación de la edad y la sobrevivencia por el método de Pearson que es indicado para variables de escala métrica y intervalar.*

```
# Correlación entre las variables edad (Age_fin) y Survived
from scipy.stats import normaltest
import scipy.stats as stats
from scipy.stats import pearsonr
```

```
(Rho, p_value) = pearsonr(train4['Age_fin'], train4['Survived'])
print('Coeficiente de Correlación - Pearson:´{:.2f}'.format(Rho))
print('P-value: {}'.format(p_value))
```

```
Coeficiente de Correlación - Pearson:´0.01
P-value: 0.8750155808538369
```

*Ahora vamos a explorar los datos también con regresión logística con el método Spearman:*

```
# Exploración de los datos con regresión logística
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

```
model_1 = sm.Logit(train4['Survived'], train4[['Pclass', 'Age_fin', 'Fare',
                                               'Con hermanos o cónyuge',
                                               'Con hijos a bordo', 'S', 'C',
                                               'Q', 'female']])
model_2 = model_1.fit(method='bfgs')
print(model_2.pvalues)
model_2.summary()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.604258
         Iterations: 35
         Function evaluations: 38
         Gradient evaluations: 38
Pclass                      4.945309e-08
Age_fin                     6.241771e-01
Fare                        2.182494e-02
Con hermanos o cónyuge      9.675020e-01
Con hijos a bordo           8.626623e-01
S                           1.455160e-01
C                           6.638382e-02
Q                           7.642544e-02
female                      8.486693e-01
dtype: float64
```

Logit Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Survived | **No. Observations:** | 891 |
| **Model:** | Logit | **Df Residuals:** | 882 |
| **Method:** | MLE | **Df Model:** | 8 |
| **Date:** | Mon, 06 Jan 2020 | **Pseudo R-squ.:** | 0.09259 |
| **Time:** | 20:00:12 | **Log-Likelihood:** | -538.39 |
| **converged:** | False | **LL-Null:** | -593.33 |
| **Covariance Type:** | nonrobust | **LLR p-value:** | 4.053e-20 |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Pclass** | -0.6636 | 0.122 | -5.453 | 0.000 | -0.902 | -0.425 |
| **Age_fin** | 0.0027 | 0.006 | 0.490 | 0.624 | -0.008 | 0.014 |
| **Fare** | 0.0111 | 0.005 | 2.293 | 0.022 | 0.002 | 0.021 |
| **Con hermanos o cónyuge** | -0.0072 | 0.176 | -0.041 | 0.968 | -0.353 | 0.338 |
| **Con hijos a bordo** | -0.0347 | 0.200 | -0.173 | 0.863 | -0.427 | 0.358 |
| **S** | 0.6003 | 0.412 | 1.456 | 0.146 | -0.208 | 1.409 |
| **C** | 0.8241 | 0.449 | 1.836 | 0.066 | -0.056 | 1.704 |
| **Q** | 0.8351 | 0.471 | 1.772 | 0.076 | -0.089 | 1.759 |
| **female** | 0.0309 | 0.162 | 0.191 | 0.849 | -0.286 | 0.348 |

*Así, según el Statsmodel, las variables que tienen significado para la predicción de la supervivencia son todas las utilizadas en el modelo: Pclass, Age_fin, Fare, Con hermanos o cónyuge, Con hijo a bordo, S, C y Q.*

*Ya que todas ellas quedaran abajo del threshold 0.05.*

*Como ya tenemos las variables que vamos a utilizar, quedaremos con un dataframe de las variables para intentar un modelo de input en la variable Survived de la base de datos test. Quedamos con las variables que vamos a utilizar:*

```
# Quedamos con las variables a utilizar:
train5 = train4.drop(columns=['PassengerId', 'Sex', 'Embarked', 'SibSp_2', 'Parch_2',
                              'Sin hermanos o cónyuge', 'Sin hijos a abordo', 'male'])
```

```
train5.head()
```

| | Survived | Pclass | Fare | Age_fin | C | Q | S | Con hermanos o cónyuge | Con hijos a bordo | female |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | 7.2500 | 22.0 | 0 | 0 | 1 | 1 | 0 | 0 |
| **1** | 1 | 1 | 71.2833 | 38.0 | 1 | 0 | 0 | 1 | 0 | 1 |
| **2** | 1 | 3 | 7.9250 | 26.0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 3 | 1 | 1 | 53.1000 | 35.0 | 0 | 0 | 1 | 1 | 0 | 1 |
| **4** | 0 | 3 | 8.0500 | 35.0 | 0 | 0 | 1 | 0 | 0 | 0 |

*Vamos a utilizar el método de KNeighbors para hacer la clasificación:*

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
x_train, x_test, y_train, y_test = train_test_split(train5.drop('Survived',axis=1),
                                                    train5['Survived'],
                                                    test_size=0.3,
                                                    random_state=1, stratify=y)
```

*Mantenemos stratify=y para que la base de teste contenga el porcentual de sobrevivientes igual a la base de entrenamiento.*

*Así, para n_neighbors igual a 2, tenemos una precisión de 61,9% en el modelo KNeigghbors*

```
knn = KNeighborsClassifier (n_neighbors = 2)
```

```
knn.fit (x_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=2, p=2,
                     weights='uniform')
```

```
# Miramos la precisión del modelo:
knn.score(x_test, y_test)
```

```
0.6194029850746269
```

*Vamos a comprobar el modelo GradientBoostingClassifier que es un modelo basado en árboles de decisión:*

```
# Modelo: GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
clf = GradientBoostingClassifier(random_state = 0)
clf.fit(x_train, y_train)
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=0, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

```
# Precisión del modelo:
clf.score(x_test, y_test)
```

```
0.6716417910447762
```

*Así podríamos aplicar el modelo GradientBoostingClassifier en la base de datos test de Titanic y así tendríamos la perdición de los sobreviviente, ya que el modelo KNeigghbors tiene una precisión más alta que el KNeigghbors.*

***Tabla de Contribuciones:***

| ***Contribuciones*** | ***Firma*** |
|---|---|
| *Investigación previa* | *Gisele Guadalupe Almeida dos Santos Maia* |
| *Redacción de las respuestas* | *Gisele Guadalupe Almeida dos Santos Maia* |
| *Desarrollo código* | *Gisele Guadalupe Almeida dos Santos Maia* |