DESCRIPCIÓN DE LA PRÁCTICA

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

El dataset utilizado para la práctica es el archiconocido **TITANIC – *Machine Learning From Disaster***, empleado en muchos ejemplos de aprendizaje automático y repositorios como Kaggle. El dataset contiene información sobre la supervivencia de los pasajeros a bordo del Titanic (si consiguieron sobrevivir o no lo consiguieron); sólo se contempla la información relativa a los pasajeros, ignorando los datos de la tripulación. El enlace de donde se ha descargado es el siguiente:

Titanic - Machine Learning from Disaster | Kaggle

Cabe mencionar que el dataset original se compone de 2 archivos diferentes:

- **train.csv**: ~70% de los datos de entrenamiento (891 observaciones)
- **test.csv**: ~30% de los datos de prueba (418 observaciones)

Los atributos (columnas) empleados en el dataset son los siguientes:

- **`PassengerId`** → Número identificativo para cada pasajero del Titanic
- **`Survided`** → variable binaria, donde: 0 = No sobrevivió; 1 = Sí sobrevivió
  - Esta variable sólo está presente en el archivo “train.csv”, no en “test.csv”
- **`Pclass`** → Categoría en la que viajaba la persona en la embarcación
  - **1** = primera clase
  - **2** = segunda clase
  - **3** = tercera clase
- **`Name`** → nombre y apellidos de la persona que compró el billete
- **`Sex`** → variable categórica que define el género de la persona (*male* o *female*)
- **`Age`** → edad de la persona
- **`SibSp`** → número de hermanos (*siblings*) y cónyuge (*spouse*) relativos a la persona que viajaba a bordo
- **`Parch`** → número de padres (*parents*) o hijos (*children*) relativos a la persona que viajaba a bordo
- **`Ticket`** → número de ticket
- **`Fare`** → tarifa que pagó el pasajero/a por el billete
- **`Cabin`** → número de cabina del pasajero/a
- **`Embarked`** → variable categórica que define el puerto donde embarcó la persona:
  - **`C`** = Cherbourg
  - **`Q`** = Queenstown
  - **`S`** = Southampton

El objetivo del dataset es poder predecir cuáles de aquellas personas que embarcaron en el Titanic **sobrevivieron al accidente y cuáles no**, en base a los datos de entrenamiento (además de estudiar otras variables del dataset para poder llegar a conclusiones de interés). Es por eso que, en el conjunto de prueba (test.csv), no está incluida la variable objetivo "`Survived`", para poder realizar una predicción donde solo se muestren dos atributos: el *`PassengerId`* y *`Survived`*.

2. Integración y selección de los datos de interés a analizar. Puede ser el resultado de adicionar diferentes datasets o una subselección útil de los datos originales, en base al objetivo que se quiera conseguir.

Para la realización de la práctica, hemos combinado los archivos "train.csv" + "test.csv", para poder trabajar así con la totalidad de los datos (**1309 observaciones**).

Primero se han cargado los 2 datasets por separado en dos variables `train` y `test`:

```
train = pd.read_csv("../PRAC_2/titanic pec/train.csv")

test = pd.read_csv("../PRAC_2/titanic pec/test.csv")
```

Una vez cargados los dos archivos, se han juntado las dos variables, añadiendo al final de `train` las valoraciones de `test`:

```
final_df = train.append(test, ignore_index=True)
```

De esta forma, se añaden las **418 observaciones** de `test` a las **891 observaciones** de `train`, obteniendo el dataset final (`final_df`)

```
final_df.shape

(1309, 12)
```

Así pues, el dataset sobre el que trabajaremos contiene **1309 observaciones** (filas) y **12 atributos** (columnas).

Primero se echa un vistazo a los 10 primeros valores del `final_df`:

```
final_df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0.0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1.0	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0.0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0.0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0.0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0.0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1.0	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1.0	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Y luego se echa un vistazo a los 10 últimos valores del dataset:

```
final_df.tail(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1299	1300	NaN	3	Riordan, Miss. Johanna Hannah""	female	NaN	0	0	334915	7.7208	NaN	Q
1300	1301	NaN	3	Peacock, Miss. Treasteall	female	3.0	1	1	SOTON/O.Q. 3101315	13.7750	NaN	S
1301	1302	NaN	3	Naughton, Miss. Hannah	female	NaN	0	0	365237	7.7500	NaN	Q
1302	1303	NaN	1	Minahan, Mrs. William Edward (Lillian E Thorpe)	female	37.0	1	0	19928	90.0000	C78	Q
1303	1304	NaN	3	Henriksson, Miss. Jenny Lovisa	female	28.0	0	0	347086	7.7750	NaN	S
1304	1305	NaN	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S
1305	1306	NaN	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C
1306	1307	NaN	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S
1307	1308	NaN	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S
1308	1309	NaN	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C

Como ya se ha mencionado, en el archivo de prueba "test.csv" la variable objetivo "*Survived*" no existe, de modo que al juntar los dos datasets, las observaciones del archivo "test.csv" se incluirían sin ningún valor en "*Survided*", originando 418 observaciones con **valores faltantes** (NaN o null) en dicho atributo; los valores faltantes se gestionarán en el siguiente apartado 3.1.

De esta forma, se suman las 418 observaciones de `test` con las 891 observaciones de `train`, obteniendo el dataset final (`final_df`)

```
final_df.shape

(1309, 12)
```

3. Limpieza de los datos.

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

Mediante la función `info()` podemos ver de un solo vistazo qué atributos tienen variables nulos o faltantes (*NaN values*):

```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  1309 non-null   int64  
 1   Survived      891 non-null   int64  
 2   Pclass       1309 non-null   int64  
 3   Name         1309 non-null   object 
 4   Sex          1309 non-null   object 
 5   Age          1046 non-null   float64
 6   SibSp        1309 non-null   int64  
 7   Parch        1309 non-null   int64  
 8   Ticket       1309 non-null   object 
 9   Fare         1308 non-null   float64
 10  Cabin         295 non-null   object 
 11  Embarked     1307 non-null   object 
dtypes: float64(2), int64(5), object(5)
memory usage: 122.8+ KB
```

En el resultado que nos muestra por pantalla, si nos fijamos en la columna "`Non-Null Count`" observamos en qué atributos existen valores nulos o faltantes. Si pone "**`1309 non-null`**", quiere decir que están todos los valores. Si el número **es menor de 1309**, querrá decir que existen observaciones con valores nulos (se han marcado en rojo estas variables). Solo habrá que hacer la resta para averiguar cuántos valores nulos hay:

- **`Survived`**: 1309 – 891 = **418 valores nulos**
- **`Age`**: 1309 – 1046 = **263 valores nulos**
- **`Fare`**: 1309 – 1308 = **1 valor nulo**
- **`Cabin`**: 1309 – 295 = **1014 valores nulos**
- **`Embarked`**: 1309 – 1307 = **2 valores nulos**

Otra forma más directa de averiguarlo es mediante la combinación de las funciones `isnull( )` y `sum( )`:

```
final_df.isnull().sum()
```

```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
```

Se puede tratar de gestionar los valores faltantes o nulos de estos 5 atributos de formas diferentes. Los valores faltantes de `Survived` se tratarán más a fondo en el **punto 4.3**. Empezaremos con los que tienen los valores más bajos (`Embarked` y `Fare`):

- **`Embarked`**: como sólo tiene 2 valores nulos, serán fáciles de averiguar. Estas son las dos observaciones que les falta un valor en `Embarked`:

```
final_df[final_df["Embarked"].isna()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN

Vemos que las pasajeras **Amelie Icard** y **Martha Evelyn Stone**, que corresponden a las observaciones **nº 62** y **nº830** respectivamente, tienen valor `NaN` en `Embarked`, es decir, no se sabe en qué puerto llegaron a embarcar.

Con un poco de investigación, en la web **https://www.encyclopedia-titanica.org/** se puede averiguar fácilmente, y se comprueba que ambas pasajeras embarcaron en el puerto de Southampton. Así pues, mediante la función `fillna( )`, se pueden sustituir los dos valores faltantes de `Embarked` por el valor "**S**":

```
final_df["Embarked"].fillna("S", inplace=True)
```

Comprobamos que, efectivamente, se han sustituido los valores `NaN` por el valor **S**:

```
final_df.loc[61]
```

```
PassengerId                     62
Survived                         1
Pclass                           1
Name            Icard, Miss. Amelie
Sex                          female
Age                              38
SibSp                             0
Parch                             0
Ticket                       113572
Fare                             80
Cabin                           B28
Embarked                          S
Name: 61, dtype: object
```

```
final_df.loc[829]
```

```
PassengerId                                     830
Survived                                          1
Pclass                                            1
Name            Stone, Mrs. George Nelson (Martha Evelyn)
Sex                                          female
Age                                              62
SibSp                                             0
Parch                                             0
Ticket                                       113572
Fare                                             80
Cabin                                           B28
Embarked                                          S
Name: 829, dtype: object
```

- **`Fare`**: para el valor faltante en la variable `Fare` (tarifa aplicada), si consultamos en la misma página web podemos averiguar este valor:

```
final_df[final_df["Fare"].isna()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1043	1044	0	3	Storey, Mr. Thomas	male	60.5	0	0	3701	NaN	NaN	S

Consultando la web, podemos averiguar un poco de la historia que hay detrás del pasajero **Thomas Storey**[1]. Nos cuenta que era un marinero que trabajó en varias embarcaciones (en 1890 se unió a la tripulación de American Line), y llegó a ascender a la categoría de "maestro de armas" (uno de los rangos más altos dentro de la navegación británica). Fue transferido a Filadelfia, pero debido a la **Huelga Minera Inglesa**[2], que fue la primera huelga de mineros del carbón en el Reino Unido, se originaron problemas en la programación de las flotas y se canceló su viaje a Filadelfia, y tuvo que ser reubicado en el Titanic, junto a otros compañeros, como pasajeros.

Sabiendo esta historia, se puede inferir que no pagó ningún coste por el billete (ya que fue "obligado" a embarcar en el Titanic por la reprogramación de rutas causada por la huelga minera). De modo que se sustituirá el valor faltante por un 0:

```
final_df["Fare"].fillna(0, inplace=True)
```

Comprobamos que, efectivamente, se ha sustituido el valor faltante por un 0:

```
final_df.loc[1043]
```

```
PassengerId                  1044
Survived                        0
Pclass                          3
Name           Storey, Mr. Thomas
Sex                          male
Age                          60.5
SibSp                           0
Parch                           0
Ticket                       3701
Fare                            0
Cabin                         NaN
Embarked                        S
Name: 1043, dtype: object
```

Ahora vamos con las otras dos variables, que tienen un número bastante elevado de valores nulos o faltantes.

- **`Age`**: en la variable "Edad" existían 263 valores nulos. Al tratarse de una variable numérica, se puede realizar **imputación de valores**. En este caso, se pueden sustituir todos los valores faltantes por la media aritmética de todas las edades:

```
final_df["Age"] = final_df["Age"].fillna(final_df["Age"].mean())
```

- **`Cabin`**: debido al número tan elevado de valores faltantes en la variable `Cabin` (1014 observaciones, casi el 80% del total), se decide eliminar este atributo:

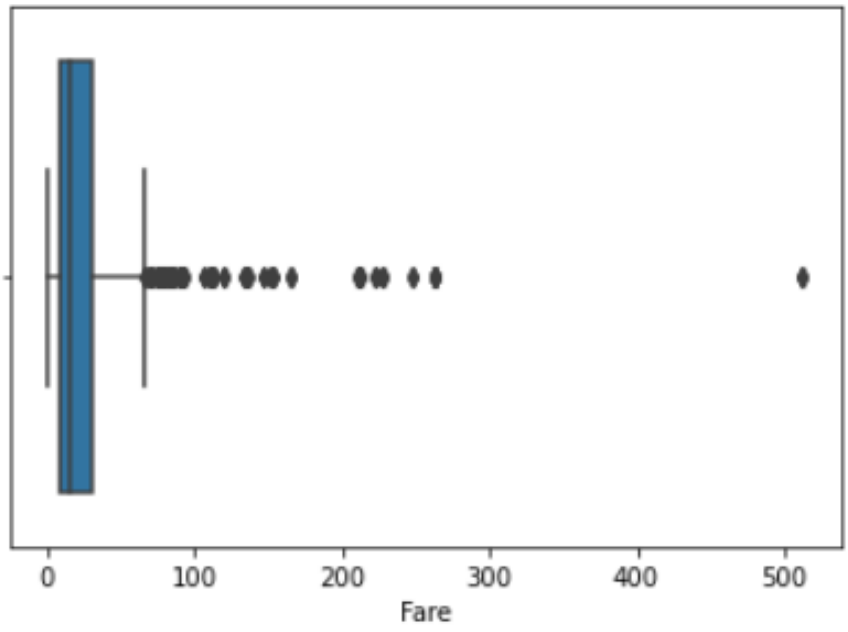
```
final_df.drop("Cabin", axis = 1, inplace = True)
```

[1] Thomas Storey : Titanic Victim (encyclopedia-titanica.org)
[2] Huelga Minera Inglesa (mites.gob.es)

3.2. Identifica y gestiona los valores extremos.

En la variable `Fare`, según el histograma representado, vemos que hay valores anómalos (*outliers*). La forma más directa de ver los outliers sería mediante un boxplot:

```
sns.boxplot(x=final_df[‘Fare’])
```



Sin embargo, se ha establecido el criterio siguiente: todos aquellos valores de `Fare` que estén alejados de la media aritmética a más del triple de su desviación típica (en ambas direcciones, es decir, a más del triple o menos del triple) serán considerados valores atípicos.

```
fare_mean = final_df[‘Fare’].mean()
fare_std = final_df[‘Fare’].std()
low = fare_mean – (3 * fare_std)
high = fare_mean + (3 * fare_std)
fare_outliers = final_df[(final_df[‘Fare’] < low | (final_df[‘Fare’] > high
fare_outliers
```

Aplicando este criterio, nos aparecen **38 observaciones** cuyo valor en la tarifa se aleja 3 veces más (o 3 veces menos) la desviación típica respecto a la mediana de la tarifa (`Fare`). De hecho, si analizamos la asimetría de la distribución normal de `Fare` respecto a su media, nos sale el siguiente valor:

```
print('skewness value of Fare: ', final_df['Fare'].skew())
skewness value of Fare:  4.368574813093145
```

Para lidiar con estos 38 *outliers*, los sustituiremos con el valor de la mediana `median()` de la variable `Fare`:

```
final_df['Fare'] = np.where((final_df['Fare'] < low ) |
                   (final_df['Fare'] > high),
                   final_df['Fare'].median(), final_df['Fare'])
```

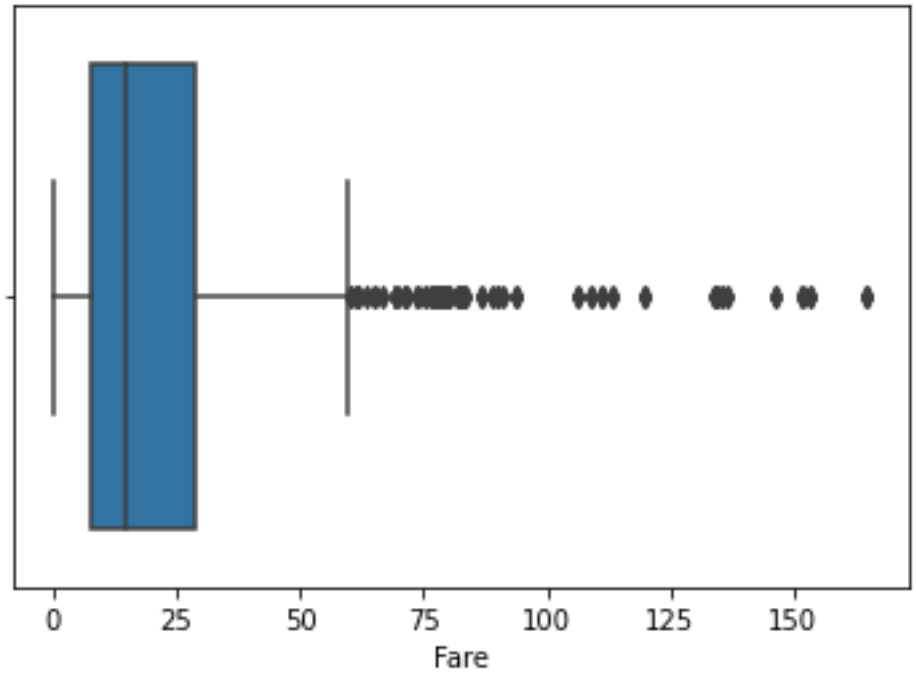
Si se vuelve a comprobar la asimetría (`skew`), se obtiene:

```
print('skewness value of Fare: ', final_df['Fare'].skew())
skewness value of Fare:  2.3641051161948408
```

Se puede apreciar que esta vez se ha obtenido un valor más bajo de asimetría, lo cual quiere decir que la variable se asemeja un poco más a la distribución normal al haber reducido los valores de sus *outliers*.

Si se vuelve a graficar el boxplot, se puede apreciar como existen menos valores extremos:

```
sns.boxplot(x=final_df['Fare'])
```



Por otra parte, en la variable Edad (`Age`) observamos una peculiaridad: es de tipo `float64`, y tiene decimales. De hecho, observamos que hay pasajeros (presumiblemente bebés) que tienen valores menores a cero:

```
final_df.loc[(final_df['Age'] < 1)].sort_values(by=['Age'])
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
1245	1246	NaN	3	Dean, Miss. Elizabeth Gladys Millvina""	female	0.17	1	2	C.A. 2315	20.5750	S
1092	1093	NaN	3	Danbom, Master. Gilbert Sigvard Emanuel	male	0.33	0	2	347080	14.4000	S
803	804	1.0	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	C
755	756	1.0	2	Hamalainen, Master. Viljo	male	0.67	1	1	250649	14.5000	S
469	470	1.0	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	2666	19.2583	C
644	645	1.0	3	Baclini, Miss. Eugenie	female	0.75	2	1	2666	19.2583	C
1172	1173	NaN	3	Peacock, Master. Alfred Edward	male	0.75	1	1	SOTON/O.Q. 3101315	13.7750	S
78	79	1.0	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.0000	S
831	832	1.0	2	Richards, Master. George Sibley	male	0.83	1	1	29106	18.7500	S
1198	1199	NaN	3	Aks, Master. Philip Frank	male	0.83	0	1	392091	9.3500	S
305	306	1.0	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	S
1141	1142	NaN	2	West, Miss. Barbara J	female	0.92	1	2	C.A. 34651	27.7500	S

Se pretende que la variable `Age` sea de tipo entero (`int`), de modo que no aceptará decimales. Así pues, todos los valores menores a 1 se van a sustituir por cero (de modo que los bebés que solo tengan unos meses de vida, tendrán 0 años):

```
final_df.loc[final_df['Age'] < 1, "Age"] = 0
```

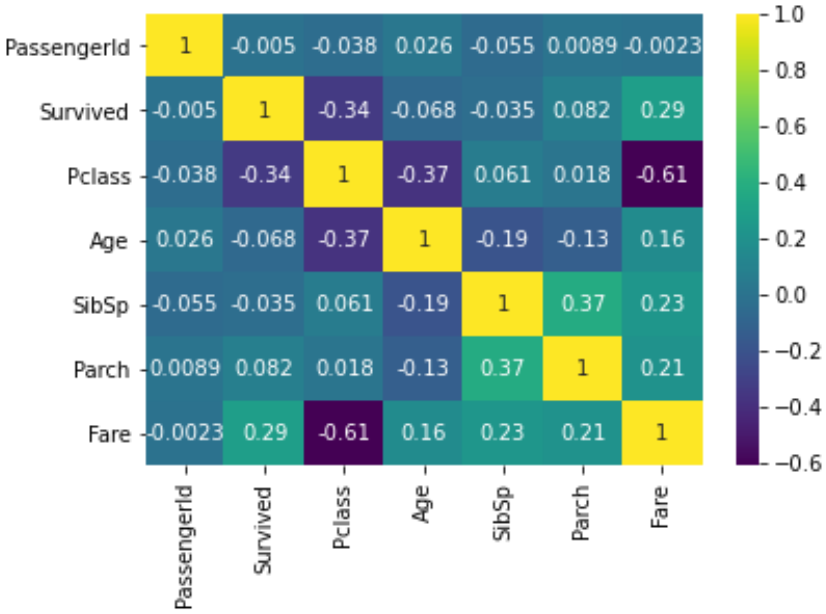
Donde `< 1` es la condición que queremos que cumpla, y `= 0` es el valor por el cual queremos reemplazar si se cumple la condición. Ahora cambiamos el tipo del atributo:

```
final_df = final_df.astype({"Age", int})
```

4. Análisis de los datos.

4.1. Selección de los grupos de datos que se quieren analizar/comparar (p. ej: si se van a comparar grupos de datos, ¿cuáles son estos grupos y qué tipo de análisis se van a aplicar?)

Primero se ha calculado la matriz de correlaciones de todas las variables del dataset para ver las relaciones entre los distintos atributos:

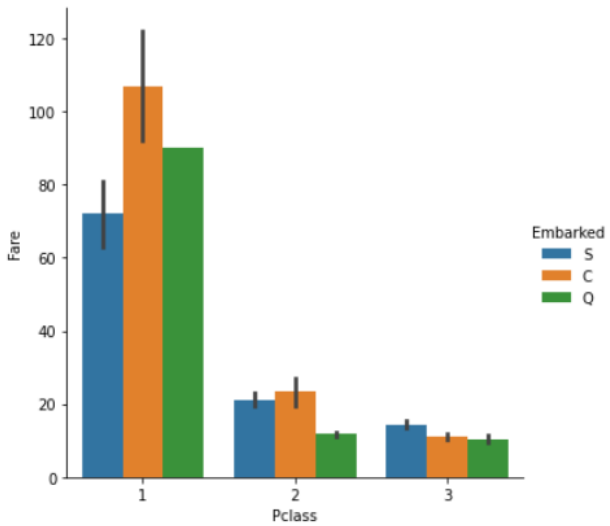


Vemos que los valores más significantes son:

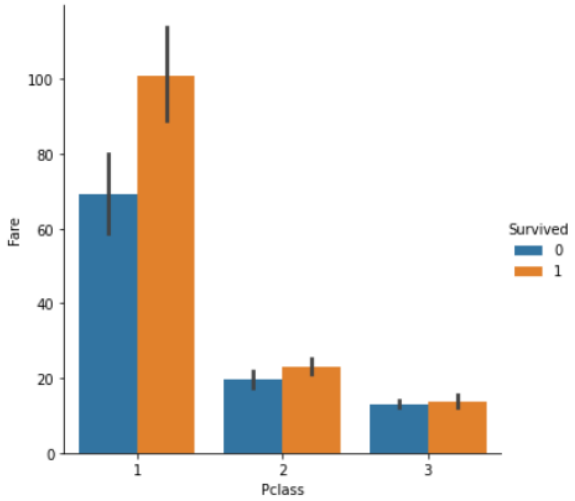
- Correlación positiva: **0.37** entre `Parch` y `SibSp`
- Correlación negativa: **-0.61** entre `Fare` y `Pclass`

Vamos a analizar la relación entre `Fare` y `Pclass`, ya que tienen el valor de correlación más alta de todos los atributos (técnicamente, la más "baja", ya que es negativa).

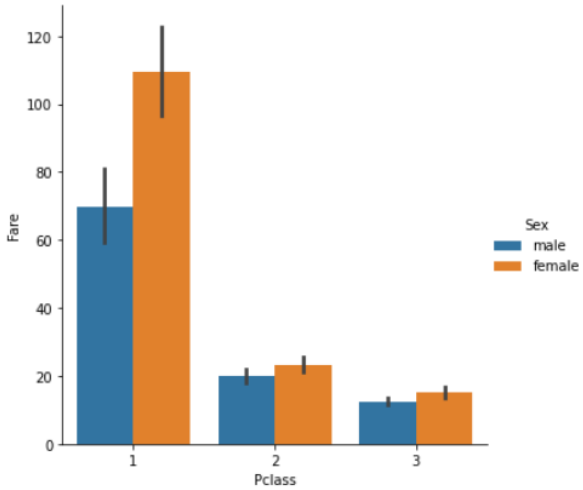
```
sns.catplot(x="Pclass", y ="Fare", hue="Embarked", kind="bar", data=final_df
```



```
sns.catplot(x="Pclass", y ="Fare", hue="Survived", kind="bar", data=final_df
```




```
sns.catplot(x="Pclass", y ="Fare", hue="Sex", kind="bar", data=final_df
```



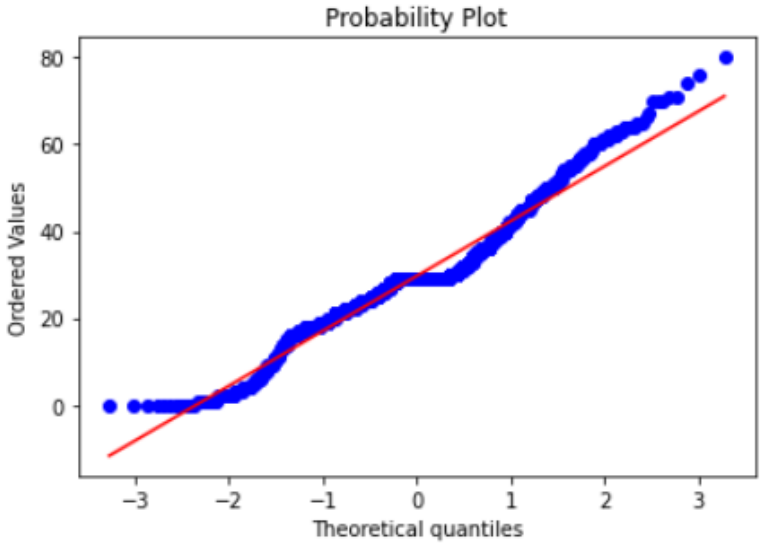
Para hacer futuras comparativas en apartados posteriores, se ha dividido el dataframe en varios grupos. Se utilizarán principalmente las variables `under18_df` y `over18_df` para el contraste de hipótesis; las demás variables se dejan para posibles análisis de interés.

```
under18_df = final_df.query("Age < 18") # People not older than 18
over18_df = final_df.query("Age >= 18") # People older than 18
firstClass_df = final_df.query("Pclass == 1") # Passengers in first class
secondClass_df = final_df.query("Pclass == 2") # Passengers in second class
thirdClass_df = final_df.query("Pclass == 3") # Passengers in third class
alive_df = final_df.query("Survived == 1") # Passengers that survived
dead_df = final_df.query("Survived == 0") # Passengers that died
test = final_df[final_df.Survived.isnull()] # Split again train and test value.isnull()
train = final_df[final_df.Survived.notnull()]
```

4.2. Comprobación de la normalidad y homogeneidad de la varianza.

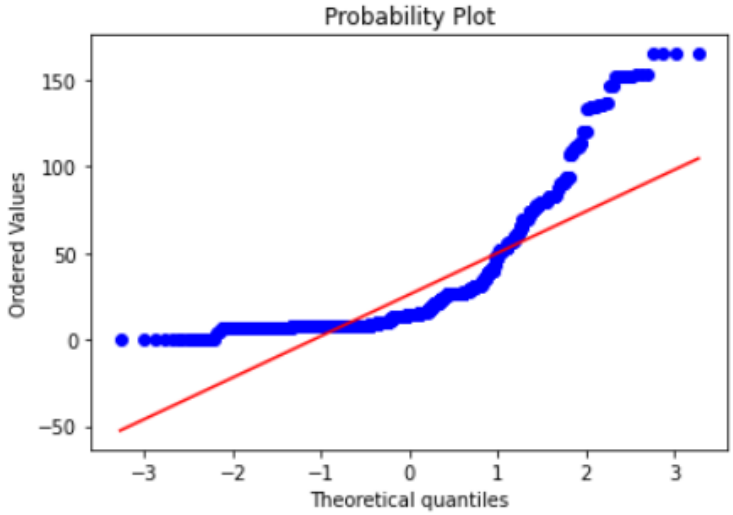
Primero vamos a comprobar la Normalidad de las variables `Age` y `Fare`.

```
import pylab
import scipy.stats as stats

stats.probplot(final_df['Age'], dist="norm", plot=pylab)
pylab.show()
```



Se puede asumir la normalidad de `Age`, puesto que sigue más o menos la recta. Ahora analicemos la variable `Fare`:

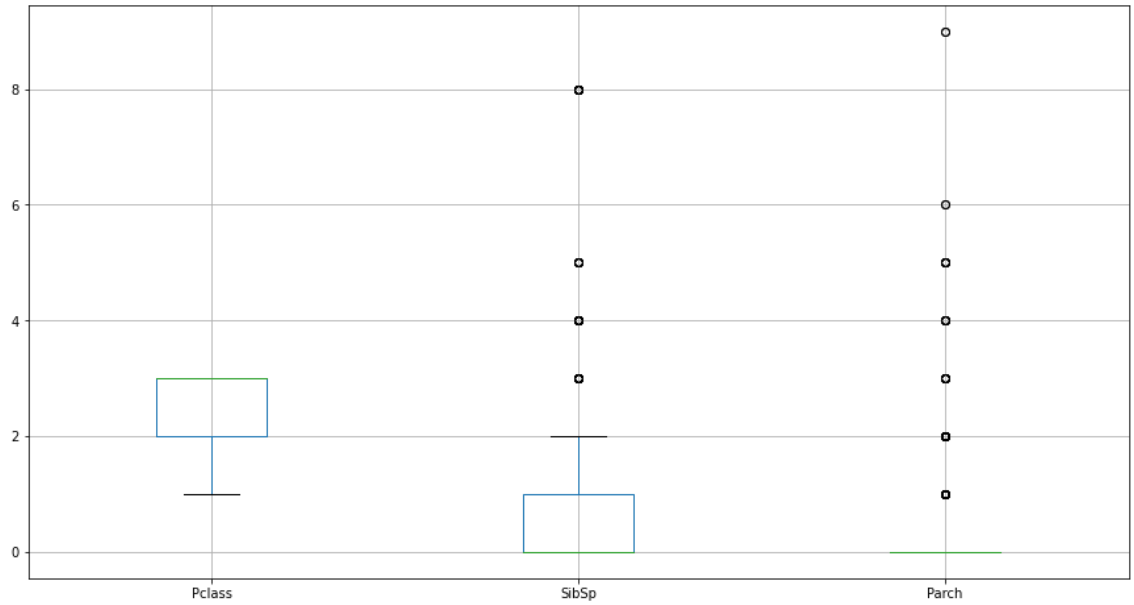
```
stats.probplot(final_df['Fare'], dist="norm", plot=pylab)
pylab.show()
```



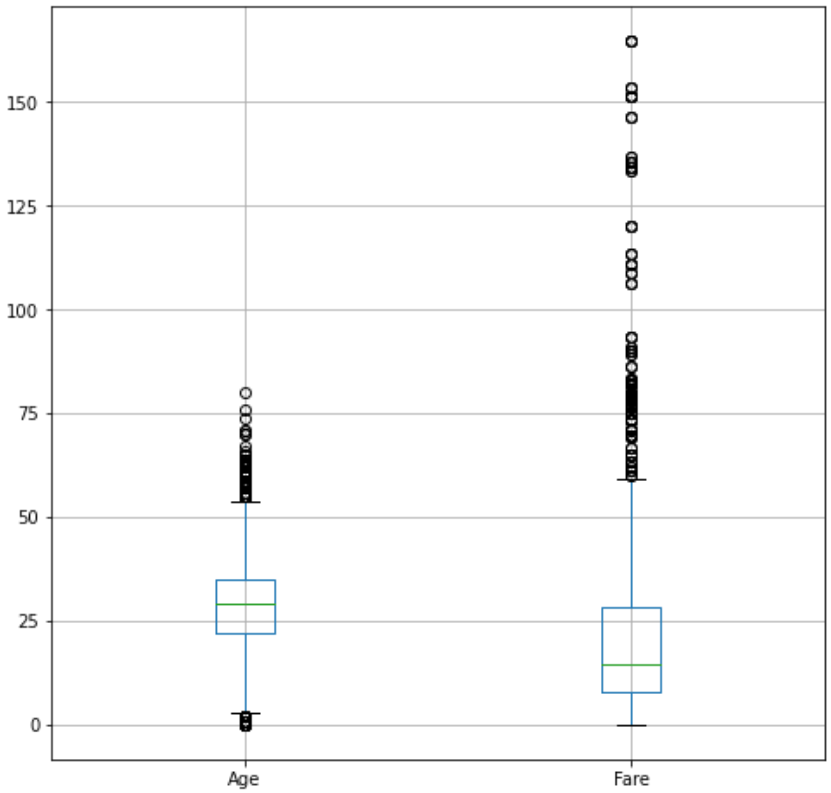
No se puede asumir normalidad en la variable `Fare`, debido a su forma irregular, ya que se desvía de la recta.

Respecto a la homogeneidad de la varianza, se puede analizar visualmente de forma rápida mediante un boxplot que represente todas las variables que sean de interés:

```
final_df[["Pclass", "Age", "SibSp", "Parch"]].boxplot()
```




```
final_df[["Age", "Fare"]].boxplot()
```



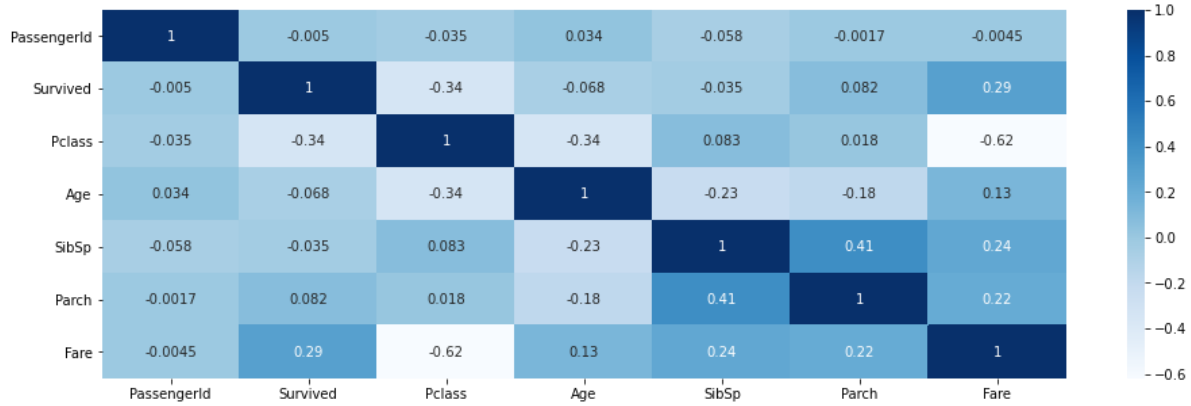
Se puede comprobar que `Fare` es la que tiene una varianza más dispersa por los outliers.

4.3. Aplicación de pruebas estadísticas para comparar los grupos de datos. En función de los datos y el objetivo de estudio, aplicar pruebas de contraste de hipótesis, correlaciones, regresiones, etc. Aplicar al menos tres métodos de análisis diferentes.

Correlaciones

Para predecir la variable `Survived`, se ha dividido el dataset en dos otra vez (como al principio), entre `train` y `test`. Primero se ha creado una matriz de correlaciones (parecida a la del punto 4.1), pero esta vez solo de los datos de entrenamiento `train`, aunque es muy parecida:

```
# Cálculo de la matriz de correlación
corr =  train.corr()

# Se dibuja el mapa de calor
plt.figure(figsize = (16,5))
sns.heatmap(corr, cmap="Blues", annot=True)
```



Regresión Logística

A continuación, se llevará a cabo un modelo de **regresión logística** para predecir la variable `Survived` en las 418 observaciones del conjunto de datos de prueba test para poder añadirlas luego al dataset original. Esto se llevará a cabo mediante las funciones `train_test_split()` (para dividir los datos), y `LogisticRegression()` de la librería `sklearn`:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

traindata = train[["Age","Pclass","Fare","PassengerId",
                   "Survived","SibSp", "Parch"]]
X = traindata.drop('Survived',axis=1)
y = traindata['Survived']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,
                                                    random_state=25)
```

A continuación, se construye el modelo y se definen las predicciones:

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
```

Antes de continuar, se puede comprobar la precisión del modelo mediante la función `accuracy_score()`:

```
accuracy_score(y_pred, y_test)
```

```
0.7150837988826816
```

Comprobamos que el modelo tiene una precisión (eficiencia) del **71,50%,** es decir, 71 valores de cada 100 estarán predichos correctamente.

En base a esto último, se puede calcular la **matriz de confusión** (valores bien clasificados VS valores mal clasificados):

```
confusion_matrix(y_pred, y_test)
```

```
array([[94, 33],
       [18, 34]], dtype=int64)
```

Finalmente, realizamos las predicciones con los datos de prueba (`test`):

```
final_test = test[["Age","Pclass","Fare","PassengerId","SibSp","Parch"]]
final_test['Survived'] = log_reg.predict(final_test)
final_test['PassengerId'] = test['PassengerId']

submission = final_test[['PassengerId','Survived']]

submission.to_csv("C:/Users/llcas/Área de Trabalho/titanic pec/submission
.csv", index=False)

submission.tail()
```

	PassengerId	Survived
1304	1305	0.0
1305	1306	1.0
1306	1307	0.0
1307	1308	0.0
1308	1309	0.0

Las predicciones se han añadido correctamente al conjunto de prueba `test`.

Así pues, para poder finalmente lidiar con los 418 valores faltantes que había en el conjunto de prueba que se ha comentado en el apartado 2, se puede fusionar la variable "`test`" con la variable "`submission`", mediante la función `merge()` y el argumento `how="left"`, que funciona como un **outer join** en SQL:

```
test = test.merge(submission, on="PassengerId", how="left")
```

Ahora la variable `test` contiene la variable "`Survived`" en todas las 418 observaciones que no estaban en el archivo "test.csv".

Una vez que la variable `test` está actualizada, solo hace falta juntarla con la variable `train` mediante la función `append()`:

```
final_df = train.append(test, ignore_index=True)
```

Y ya tendremos el dataset entero con todas las observaciones del atributo "`Survived`" sin valores nulos o faltantes.

Contraste de hipótesis

Queremos centrar el análisis en indagar un poco más sobre las características de los pasajeros que viajaban a bordo del Titanic. Para ello, queremos conocer la respuesta a preguntas como:

- **¿Los pasajeros de la Clase 1 son más jóvenes que los de la Clase 3?**
- **¿Los menores de edad tienen más probabilidad de sobrevivir que los mayores de edad?**

Nos centraremos en la primera pregunta. Para contestarla, se llevará a cabo un contraste de hipótesis:

- $H_0 = \mu_1 - \mu_2 = 0$
- $H_1 = \mu_1 - \mu_2 > 0$

Se trata de una **comparativa de medias en poblaciones normales independientes**. En este caso, μ_1 representa la media de edad de los pasajeros de la Clase1 y μ_2 la media de edad de los pasajeros de la Clase3.
En la **hipótesis nula** (H_0), por convenio, siempre se asume igualdad en ambas poblaciones (es decir, que no hay diferencia significativa de edad entre los pasajeros de la Clase 1 y la Clase 3). En contrapartida, en la **hipótesis alrternativa** (H_1) se asume que sí que existe diferencia significativa entre la media de edad de los pasajeros de ambas clases.

Se utilizará un test de hipótesis **de dos muestras sobre la media**. Primero extraemos los valores de la variable categórica `Pclass` que utilizaremos en el test:

```
# A modo informativo, veamos cuántos pasajeros hay en cada clase
final_df.groupby("Pclass").size()
```

```
Pclass
1    323
2    277
3    709
dtype: int64
```

```
# Extraemos todas las observaciones de "Age" de los pasajeros en Clase 1
class_1 = firstClass_df["Age"]

# Extraemos todas las observaciones de los pasajeros en Clase 3
class_3 = thirdClass_df["Age"]
```

Ahora se obtendrán los valores de la media (μ) y la desviación típica (σ) de cada muestra:

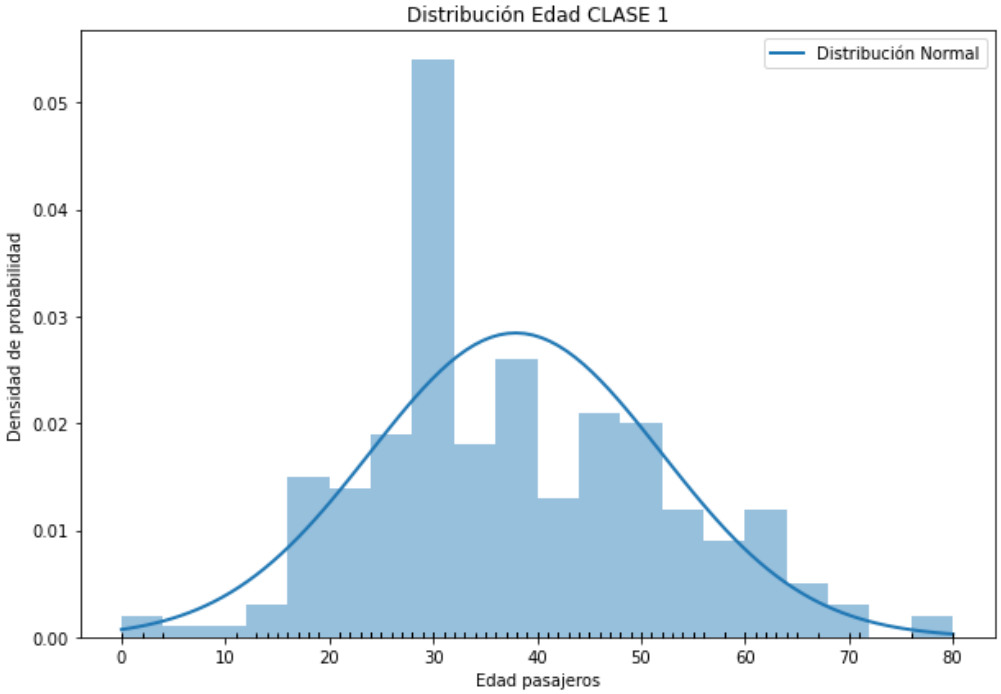
```
# Importamos el módulo "stats" de la librería scipy
from scipy import stats

# Calculamos la media y la desv. típica de la edad de la Clase 1
mu_1, sigma_1 = stats.norm.fit(class_1)

print(mu_1)
print(sigma_1)
```

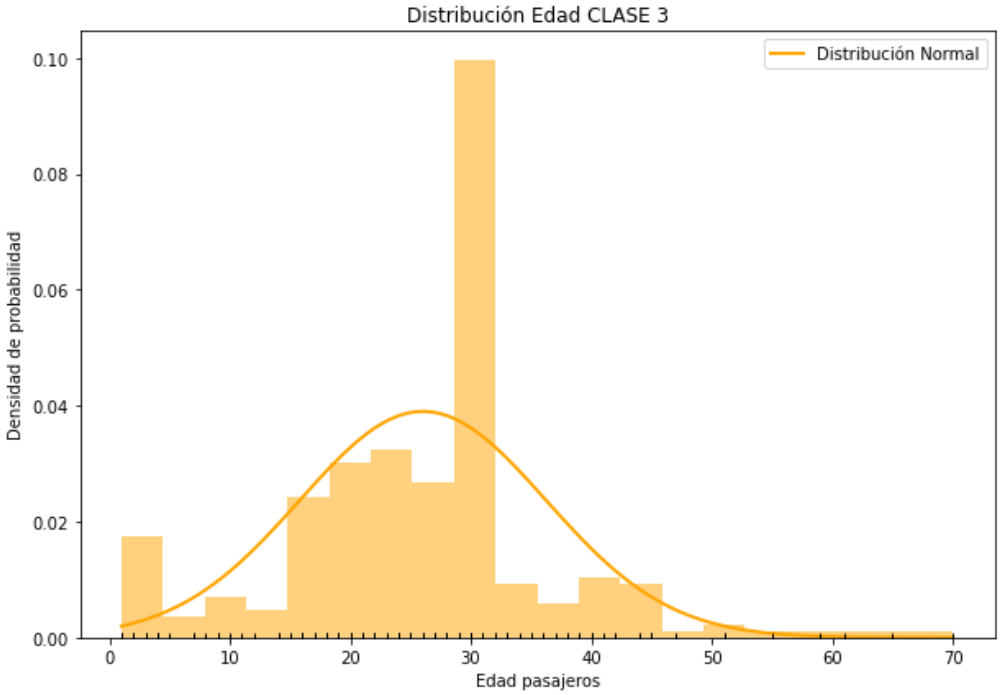
```
37.925696594427244
14.022562173217601
```

```
# Calculamos la media y la desv. típica de la edad de la Clase 3
mu_3, sigma_3 = stats.norm.fit(class_3)

print(mu_3)
print(sigma_3)
```

```
26.0197461212976
10.224402041912825
```

En el archivo de jupyter notebook se ha llevado a cabo el test de **Shapiro-Wilk** para comprobar si los datos de las dos muestras siguen una distribución normal o no, y se ha corroborado que **no** la siguen, como se puede comprobar en la siguiente imagen (el código está en el *notebook* de jupyter):





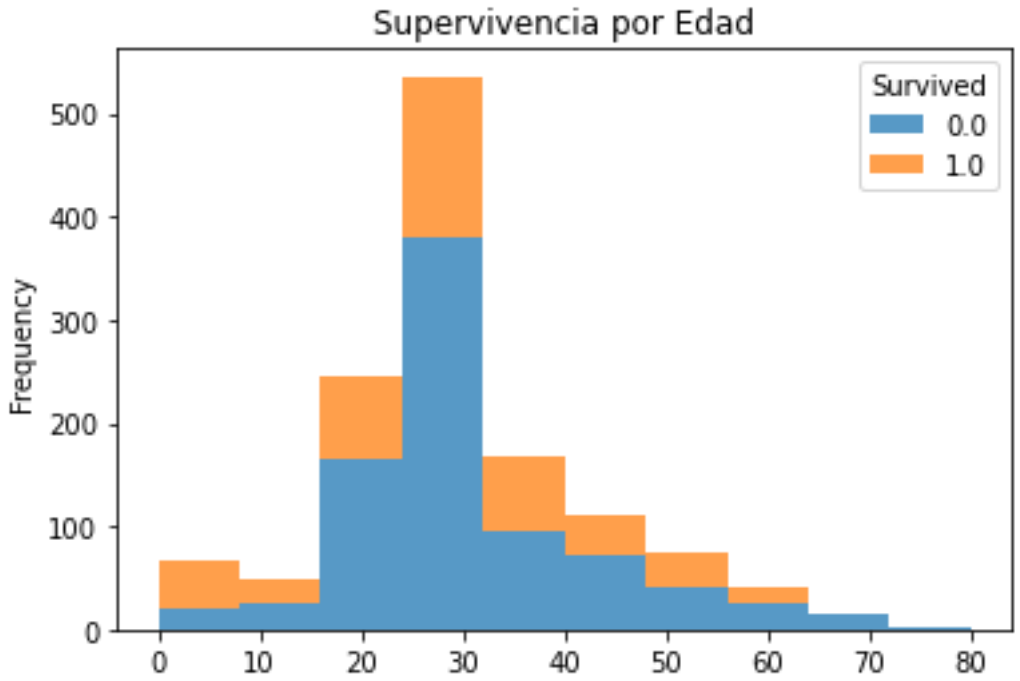
Como las muestras deben tener el mismo tamaño (número de observaciones), se sustraen los 250 primeros valores de `Class_1` y `Class_3`:

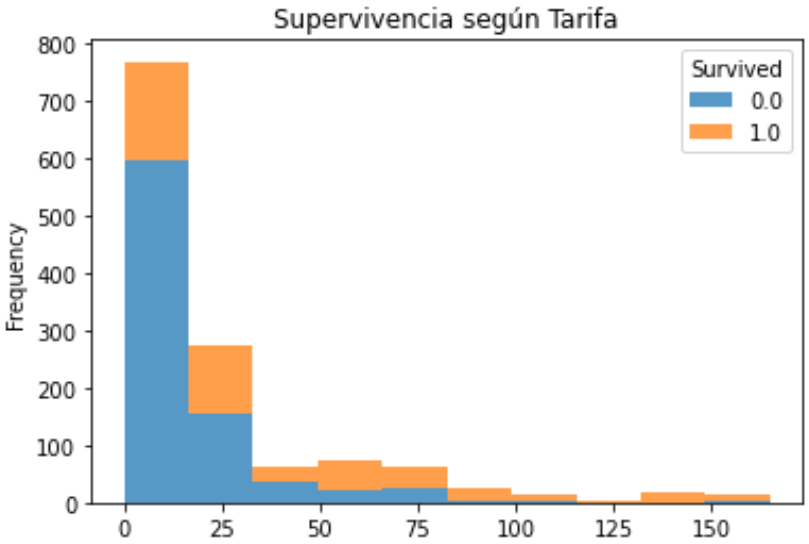
```
# 250 primeras observaciones de la Clase 1
c1 = class_1[:250]

# 250 primeras observaciones de la Clase 3
c3 = class_3[:250]
```

5. Representación de los resultados a partir de tablas y gráficas. Este apartado se puede responder a lo largo de la práctica, sin necesidad de concentrar todas las representaciones en este punto de la práctica.

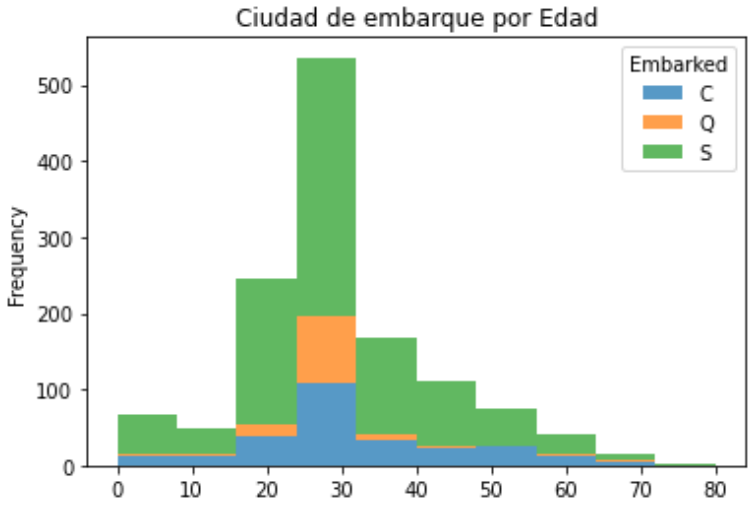
Además de las gráficas que se han ido presentando a lo largo de toda la práctica, se van a incluir algunas más que ayudan a analizar en profundidad las variables del dataset:

```
import matplotlib.pyplot as plt

survived_age = final_df.pivot(columns='Survived', values='Age')
survived_fare = final_df.pivot(columns='Survived', values='Fare')
class_age = final_df.pivot(columns='Pclass', values='Age')
embarked_fare = final_df.pivot(columns='Embarked', values='Age')


survived_age.plot.hist(stacked=True, alpha=0.75,
                       title="Supervivencia por Edad")
survived_fare.plot.hist(stacked=True, alpha=0.75,
                        title="Supervivencia según Tarifa")
class_age.plot.hist(stacked=True, alpha=0.75,
                    title="Clase por Edad")
embarked_fare.plot.hist(stacked=True, alpha=0.75,
                        title="Ciudad de embarque por Edad")

plt.show()
```

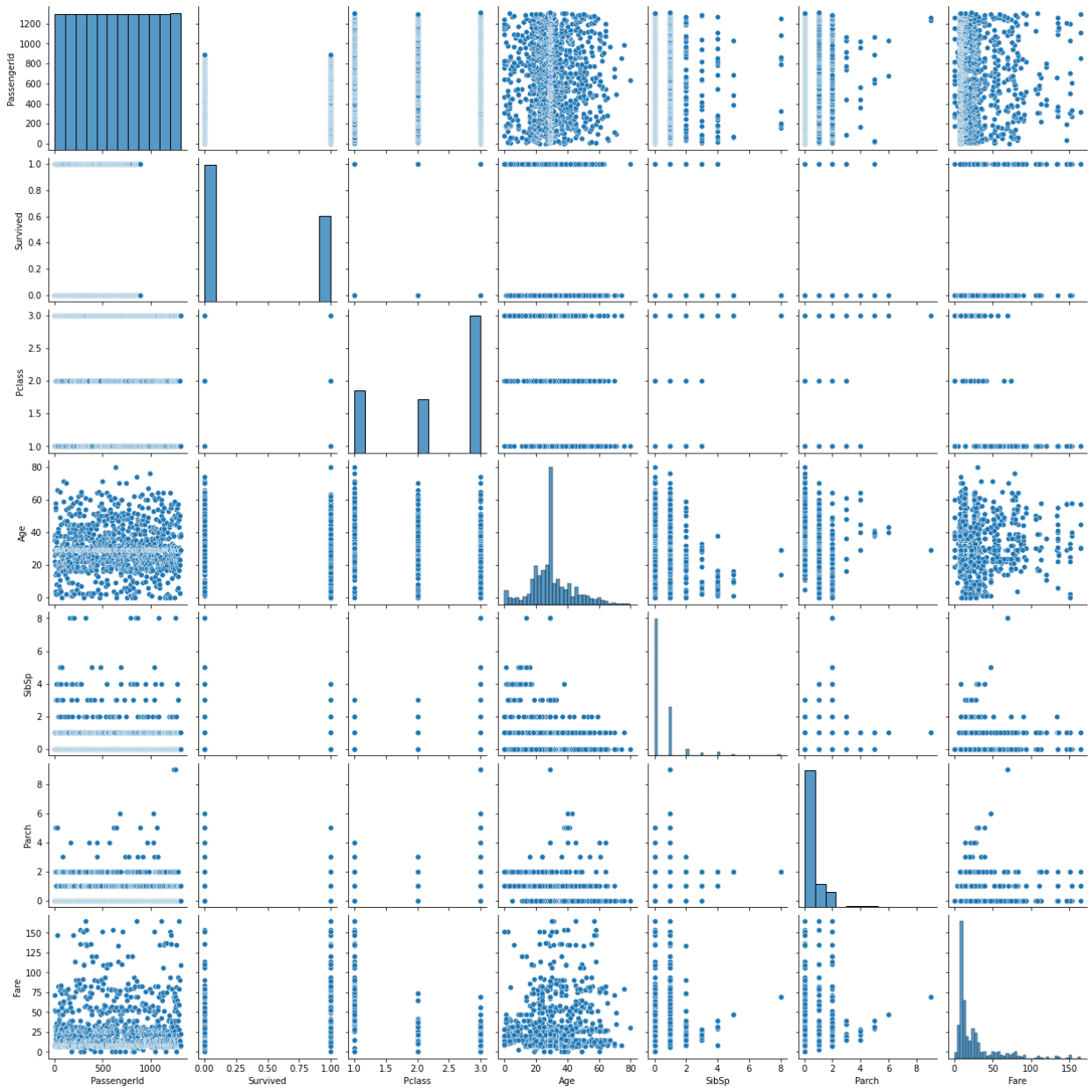



Supervivencia según Tarifa
Survived
0.0
1.0
Frequency
800
700
600
500
400
300
200
100
0
0
25
50
75
100
125
150

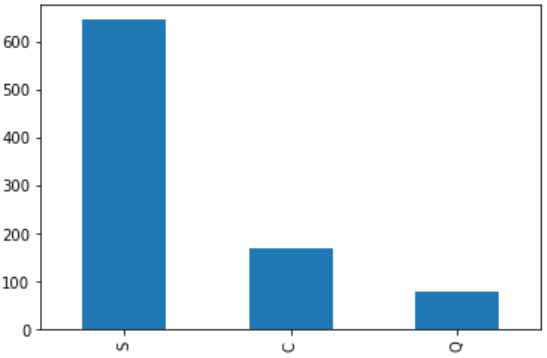

Clase por Edad
Pclass
1
2
3
Frequency
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80


Ciudad de embarque por Edad
Embarked
C
Q
S
Frequency
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80

Para poder echar un vistazo a las distribuciones bivariadas, es decir, a la distribución de una variable respecto a otra variable del dataset, se puede utilizar la función `pairplot()` de la librería `seaborn`:




```
titanic['Embarked'].value_counts().plot(kind='bar')
```



6. Resolución del problema. A partir de los resultados obtenidos, ¿cuáles son las conclusiones? ¿Los resultados permiten responder al problema?

El objetivo principal era poder predecir el atributo `Survived` en todas aquellas observaciones que tenían valor nulo en dicho atributo, y se ha conseguido con éxito: las 1309 observaciones del dataset tienen valores en todos los atributos.

Respecto al test de hipótesis, finalmente, se ha aplicado el **test "t"** para dos muestras independientes:

```
# Importamos la función "ttest_rel" de scipy
from scipy.stats import ttest_rel

# Aplicamos ttest_rel en las dos muestras
ttest , p_valor = stats.ttest_rel(c1, c3)

# Mostramos el valor de p_valor
print(p_valor)

# Test de hipótesis
if p_valor<0.05:
    print("Rechazar hipótesis nula (aceptar hipótesis alternativa)")
else:
    print("Aceptar hipótesis nula")
```

```
2.6226183808947603e-19
Rechazar hipótesis nula (aceptar hipótesis alternativa)
```

También se ha aplicado el **test Kruskal-Wallis** por no ser necesaria la normalidad de los datos, ya que estos pueden seguir un orden natural:

```
# Aplicamos Kruskal-Wallis en las dos muestras
krusk, p_valor = stats.kruskal(c1, c3)

# Mostramos el valor de p_valor
print(p_valor)

# Test de hipótesis
if p_valor<0.05:
    print("Rechazar hipótesis nula (aceptar hipótesis alternativa)")
else:
    print("Aceptar hipótesis nula")
```

```
2.1246102724689367e-20
Rechazar hipótesis nula (aceptar hipótesis alternativa)
```

Vemos que en ambos tests debemos rechazar la hipótesis nula H_0 y **aceptar la hipótesis alternativa H_1**: se puede afirmar que existe una diferencia significativa entre la media de edad de los pasajeros de la Clase 1 y los pasajeros de la Clase 3.

7. Código: hay que adjuntar el código, preferiblemente en R, con el que se ha realizado la limpieza, análisis y representación de los datos. Si lo preferís, también podéis trabajar en Python.

Todo el código se puede encontrar en el *notebook* de Jupyter adjunto a esta práctica.