January 4, 2021

TIPOLOGÍA Y CICLO DE VIDA DE LOS DATOS

PRÁCTICA 2: Limpieza y análisis de datos

Integrantes: Jaime Gimeno Ferrer- Reynel López Lantigua

---

[1]:
```
# Importación de librerias a usar
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SelectKBest, f_classif, RFE, RFECV
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import KNNImputer
import matplotlib.pyplot as plt
%matplotlib inline
```

Tabla de contenido

---

1 Detalles de la actividad

1.1 Descripción

En esta práctica se elabora un caso práctico orientado a aprender a identificar los datos relevantes para un proyecto analítico y usar las herramientas de integración, limpieza, validación y análisis de las mismas.

1.2 Objetivos

Los objetivos de esta práctica son:

- Aprender a aplicar los conocimientos adquiridos y su capacidad de resolución de problemas en entornos nuevos o poco conocidos dentro de contextos más amplios o multidisciplinares.
- Saber identificar los datos relevantes y los tratamientos necesarios (integración, limpieza y validación) para llevar a cabo un proyecto analítico.
- Aprender a analizar los datos adecuadamente para abordar la información contenida en los datos.
- Identificar la mejor representación de los resultados para aportar conclusiones sobre el problema planteado en el proceso analítico.
- Actuar con los principios éticos y legales relacionados con la manipulación de datos en función del ámbito de aplicación.
- Desarrollar las habilidades de aprendizaje que les permitan continuar estudiando de un modo que tendrá que ser en gran medida autodirigido o autónomo.
- Desarrollar la capacidad de búsqueda, gestión y uso de información y recursos en el ámbito de la ciencia de datos.

1.3 Competencias

En esta práctica se desarrollan las siguientes competencias del Máster de Data Science:

- Capacidad de analizar un problema en el nivel de abstracción adecuado a cada situación y aplicar las habilidades y conocimientos adquiridos para abordarlo y resolverlo.
- Capacidad para aplicar las técnicas específicas de tratamiento de datos (integración, transformación, limpieza y validación) para su posterior análisis.

---

2 Descripción del dataset

Para esta práctica se ha escogido el dataset propuesto en el enunciado `Titanic: Machine Learning from Disaster`. Este dataset ya viene dividido en dos sets para aplicar modelos de Machine Learning, es posible que nos interese juntar ambos en uno para aplicar nuestro proceso de limpieza y análisis.

Este dataset proporciona una serie de atributos acerca de los pasajeros del Titanic. Estos atributos se suelen utilizar para predecir un atributo especial que también se aporta e indica si un pasajero sobrevivió o no.

A continuación vemos una lista de los atributos que se incluyen en el dataset.

- survival: Identifica si un pasajero sobrevivió o no (valor 1 o 0 respectivamente).
- pclass: Identifica la clase del ticket que compró el pasajero, hay primera clase (valor 1), segunda (2) y tercera (3).
- sex: Sexo del pasajero
- Age: Edad del pasajero
- sibsp: Número de hermanos/cónyuges a bordo del Titanic
- parch: Número de padres/hijos a bordo del Titanic
- ticket: Número del ticket del pasajero
- cabin: Número de cabina donde el pasajero se alojaba
- fare: Tarifa pagada por el pasajero
- embarked: Puerto donde embarcó el pasajero

Además, en el repositorio del dataset se especifican los siguientes detalles de los datos.

La variable **pclass** tiene relación con el estatus socio-económico. - Primera clase = Clase alta - Segunda clase = Clase media - Tercera clase = Clase baja

El atributo **age** es una fracción si la edad es menor que 1. Si la edad es estimada, lo es con el formato xx.5.

Para el atributo **sibsp** se consideran las relaciones: hermano, hermana, hermanastro, hermanastra, esposa y marido (novios y amantes no fueron considerados).

En **parch** se consideran las relaciones: padre, madre, hijo, hija, hijastro, hijastra. Algún niño viajó sin niñera así que su parch = 0.

3 Importancia y objetivos del dataset

Con este dataset se pretende encontrar los atributos que influyen en mayor y menor medida a la posibilidad de supervivencia de los pasajeros del titanic. Con los resultados se pueden sacar conclusiones acerca de la diferencia entre clases, género y edad a la hora de dejar subir a un pasajero a las barcas salvavidas, por ejemplo. Así, se puede extraer un conexto ideológico de la socidad de la época (1912).

Por otra parte, este dataset es históricamente usado para probar diferentes modelos de machine learning y ha servido como primeros pasos para todas las personas que se han introducido en este mundo.

3.1 Integración y selección de los datos de interés a analizar.

A continuación, cargaremos el dataset. Como hemos dicho antes, podríamos cargar *train* y *test* y unirlos, pero de momento trabajaremos solo con train.

[2]:
```
df = pd.read_csv("../data/titanic_in.csv")
# Mostramos las 5 primeras filas
df.head()
```

[2]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Vemos como aparecen todos los atributos mencionados. Para nuestro análisis realmente no nos interesan los atributos *Cabin*, *Ticket* y *Name*, así que los descartamos del set (no aportan información nueva).

[3]:
```
df.drop(["Cabin", "Ticket", "Name"], axis=1, inplace=True)
# Mostramos las 5 primeras filas
df.head()
```

[3]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

Trabajaremos entonces con este dataset.

4 Limpieza de los datos

Ya importado el dataset podemos proceder a hacer un primer análisis de su contenido. A continuación, con el método *info* podemos ver el tipo de cada atributo y el número de filas no-nulas.

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(2)
memory usage: 62.8+ KB
```

Los atirbutos de tipo *objetc* son de tipo *object string*.

4.1 Valores nulos

En la tabla anterior, podemos ver que para algunos atributos existen valores nulos ya que no hay e mismo número de valores no-nulos para todos los atributos. Vamos a comprobar de nuevo esto viendo cuantos valroes faltan para cada atributo.

[5]:
```
# Comprobar numero de nulos
df.isnull().sum()
```

[5]:
```
PassengerId      0
Survived         0
Pclass           0
Sex              0
```

```
Age            177
SibSp            0
Parch            0
Fare             0
Embarked         2
dtype: int64
```

Como vemos, aparecen dos valores nulos en *Embarked* y 177 en *Age*. Hemos decidido eliminar las filas donde *Embarked* es null (nos lo podemos permitir al ser tan solo dos). En el caso de la variable *Age* se ha decidido utilizar un método de imputación que intenta predecir el valor ausente, es el caso del método *nearest neighbor imputation* que como su nombre indica utiliza el algoritmo KNN para predecir dicho valor. Para realizar esta operación se utiliza la función KNNImputer() del paquete *sklearn.impute*.

[6]:
```
# Sustuimos edades con los k vecinos más cercanos
imputer = KNNImputer(n_neighbors=2)
df.Age = imputer.fit_transform(df[['Age']])

# Eliminamos filas con 'Embarked' nulo
df.dropna(axis=0, inplace=True)

# Comprobar numero de nulos
df.isnull().sum()
```

[6]:
```
PassengerId    0
Survived       0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       0
dtype: int64
```

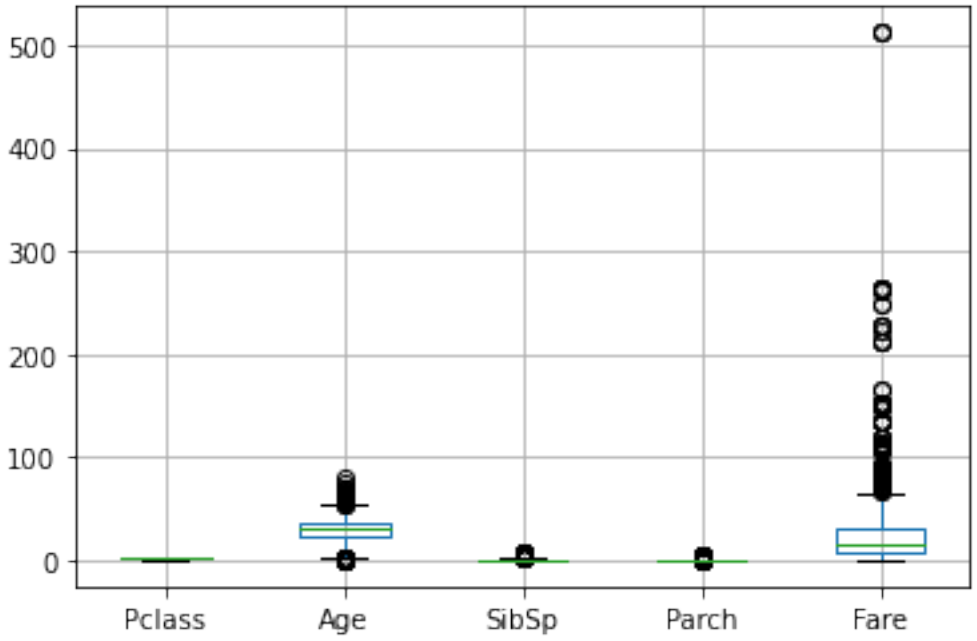
Como vemos, ya no tenemos numeros en el dataset.

4.2 Valores extremos

Otro estudio típico del "data cleansing" es el estudio de los *outliers* o valores extremos. Para buscar *outliers* en nuestro dataset podemos usar el método *boxplot* excluyendo de la representación el atributo *PassengerId* y la variable objetivo *Survived*.

[7]:
```
# Gráfico de cajas de los aributos
df.iloc[:, 2:].boxplot()
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a05510c850>
```



Vemos que aparecen *outliers* en las columnas *Age*, *SibSp*, *Parch* y *Fare*. Podemos ignorar los *outliers* de *SibSp*, *Parch* y *Age* (se decide no eliminarlos porque son valores reales y no se quiere prescindir de ellos en el modelo) pero deberíamos de tratar los de *Fare*. Para ello, basándonos en una propuesta de kaggle, caparemos los *outliers* con un valor máximo para cada atributo.

Este valor máximo lo definimos de la siguiente manera, usando el rango intercuántico (IQR).

$IQR = Q_3 - Q_1$

$Lmite\_Superior = Q_3 + 1.5 \cdot IQR$

Siendo Q_3 el tercer cuartil (cuantil 0.75) y Q_1 el primer cuartil (cuantil 0.25).

De este modo los valores que sobrepasen este límite le asignaremos un valor constante. Solo creamos un límmite superior porque no consideramos que haya *outliers* inferiormente.

[8]:
```
# Calculamos el IQR para Fare
IQR_Fare = df.Fare.quantile(0.75) - df.Fare.quantile(0.25)
print("IQR de Fare: {}".format(IQR_Fare))
```

```
IQR de Fare: 23.1042
```

[9]:
```
# Calculamos los límites superiores
Lim_Fare = df.Fare.quantile(0.75) + 1.5*IQR_Fare
print("Límite para Fare: {}".format(Lim_Fare))
```

```
Límite para Fare: 65.6563
```

Podemos ver las filas donde se encuentran estos outliers.

[10]:
```
outliers = df[df.Fare > Lim_Fare]
# Cabecera de los outliers
outliers.head()
```

[10]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
1	2	1	1	female	38.000000	1	0	71.2833	C
27	28	0	1	male	19.000000	3	2	263.0000	S
31	32	1	1	female	29.699118	1	0	146.5208	C
34	35	0	1	male	28.000000	1	0	82.1708	C
52	53	1	1	female	49.000000	1	0	76.7292	C

[11]:
```
# Cola de los outliers
outliers.tail()
```

[11]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
846	847	0	3	male	29.699118	8	2	69.5500	S
849	850	1	1	female	29.699118	1	0	89.1042	C
856	857	1	1	female	45.000000	1	1	164.8667	S
863	864	0	3	female	29.699118	8	2	69.5500	S
879	880	1	1	female	56.000000	0	1	83.1583	C

[12]:
```
print("En total tenemos {} outliers".format(outliers.shape[0]))
```

```
En total tenemos 114 outliers
```

Una vez calculados los límites podemos proceder a "capar" los valores que sobrepasen estos limites para las dos columnas.

[13]:
```
# Capamos con el cuantil 0.85 para Fare
df.Fare = np.where(df.Fare > Lim_Fare, df.Fare.quantile(0.85), df.Fare)
```

En una de las tablas superiores vemos que había un *outlier* de Fare para el passenger con *Passengerid* = 2. Si vemos qué valor tiene ahora, veremos que el valor a cambiado. En concreto el valor que vemos es el cuantil 0.85 de Fare (56.4958).

A continuación mostramos las primera filas donde podemos ver el nuevo valor de Fare para este pasajero con $Passengerid = 2$.

[14]:
```
df.head()
```

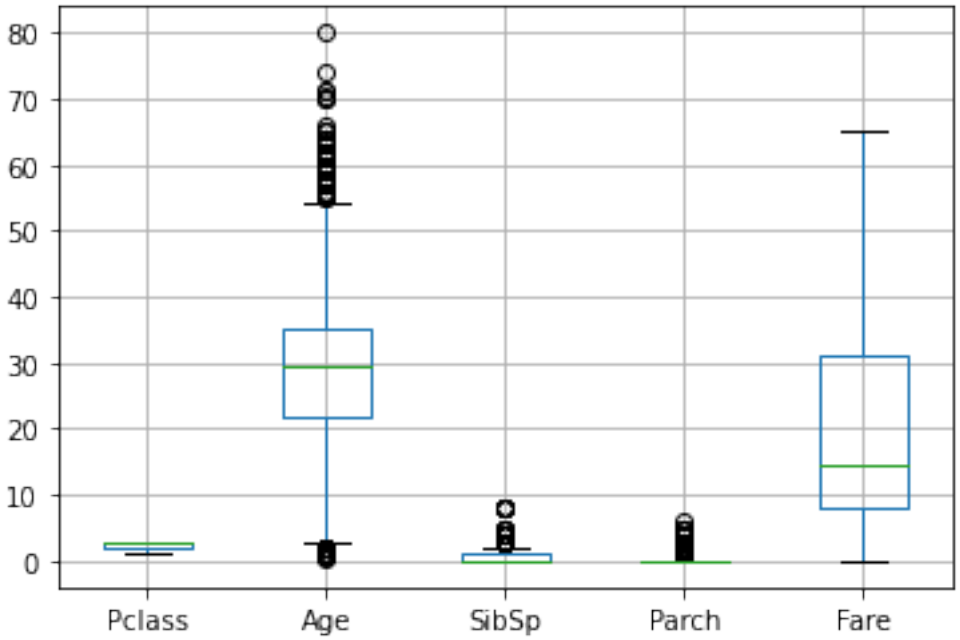
[14]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.25000	S
1	2	1	1	female	38.0	1	0	56.37664	C
2	3	1	3	female	26.0	0	0	7.92500	S
3	4	1	1	female	35.0	1	0	53.10000	S
4	5	0	3	male	35.0	0	0	8.05000	S

Si hacemos de nuevo el *boxplot* veremos como han desaparecido todos los outliers de *Fare*.

[15]:
```
df.iloc[:, 2:].boxplot()
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a05523efa0>
```



4.3 Dummies y guardado del dataset

Tenemos dos atributos que son de tipo *string*, muchos algoritmos de análisis precisan que los datos de entrada sean números.

Para las variables categóricas generaremos indicadores dummy que tendrán un 1 o 0 indicando si se trata de esa categoría o no. Esto podemos hacerlo con la función de *pandas* que se muestra a continuación (*get_dummies*).

[16]:
```
# Convertimos variables categóricas avariables indicadoras dummy
df = pd.get_dummies(df)
df
```

[16]:
```
     PassengerId  Survived  Pclass        Age  SibSp  Parch      Fare  \
0              1         0       3  22.000000      1      0   7.25000
1              2         1       1  38.000000      1      0  56.37664
2              3         1       3  26.000000      0      0   7.92500
3              4         1       1  35.000000      1      0  53.10000
4              5         0       3  35.000000      0      0   8.05000
..           ...       ...     ...        ...    ...    ...       ...
886          887         0       2  27.000000      0      0  13.00000
887          888         1       1  19.000000      0      0  30.00000
888          889         0       3  29.699118      1      2  23.45000
889          890         1       1  26.000000      0      0  30.00000
890          891         0       3  32.000000      0      0   7.75000

     Sex_female  Sex_male  Embarked_C  Embarked_Q  Embarked_S
0             0         1           0           0           1
1             1         0           1           0           0
2             1         0           0           0           1
3             1         0           0           0           1
4             0         1           0           0           1
..          ...       ...         ...         ...         ...
886           0         1           0           0           1
887           1         0           0           0           1
888           1         0           0           0           1
889           0         1           1           0           0
890           0         1           0           1           0

[889 rows x 12 columns]
```

Vemos como se han generado columnas para todas las posibles categorías de cada columna (sexo mujer, sexo hombre, embarcado en Q, S o C).

Como último paso de la limpieza de datos; reordenaremos las columnas a nuestro gusto, dejando en la última columna a la variable clase (*Survived*).

[17]:
```
df =␣
↪df[['PassengerId','Pclass','Age','SibSp','Parch','Fare','Sex_male','Sex_female','Embarked_C
```

```
df
```

[17]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Sex_male	\
0	1	3	22.000000	1	0	7.25000	1	
1	2	1	38.000000	1	0	56.37664	0	
2	3	3	26.000000	0	0	7.92500	0	
3	4	1	35.000000	1	0	53.10000	0	
4	5	3	35.000000	0	0	8.05000	1	
..	…	…	…	…	…	…	…	
886	887	2	27.000000	0	0	13.00000	1	
887	888	1	19.000000	0	0	30.00000	0	
888	889	3	29.699118	1	2	23.45000	0	
889	890	1	26.000000	0	0	30.00000	1	
890	891	3	32.000000	0	0	7.75000	1	

	Sex_female	Embarked_C	Embarked_Q	Embarked_S	Survived
0	0	0	0	1	0
1	1	1	0	0	1
2	1	0	0	1	1
3	1	0	0	1	1
4	0	0	0	1	0
..	…	…	…	…	…
886	0	0	0	1	0
887	1	0	0	1	1
888	1	0	0	1	0
889	0	1	0	0	1
890	0	0	1	0	0

[889 rows x 12 columns]

Guardamos el dataset obtenido en un archivo .csv.

[18]:
```
df.to_csv("../data/titanic_out.csv", index=False)
```

5 Análisis de los datos

5.1 Selección de datos a analizar

En este apartado se seleccionarán los atributos más relevantes para la construcción del modelo. Mediante algunas pruebas se determinan cuáles de los atributos son innecesarios y cuáles son más influyentes en la capacidad predictiva. Entre los posibles beneficios que trae consigo este proceso se encuentran los siguinetes:

- Reduce el tiempo de entrenamiento.
- Aumenta la precisión en la predicción al contar con menos datos innecesarios (ruido).
- Reduce la posibilidad del sobreentrenaiento (*overfitting*) al contar con menos datos engañosos sobre los que tomar desiciones.

Para realizar esta selección de predictores se utilizará el algoritmo llamado Eliminación Recursiva

de Variables con *cross validation* RFECV. Este algoritmo se basa en la evaluación del conjunto de datos en un modelo predictivo, asignando un valor de importancia a cada atributo según su aportación a la predicción. Inicialmente realiza la evaluación de todos los atributos del *dataset* y va disminuyendo la cantidad de atributos eliminando los menos importantes hasta que determina un valor óptimo de atributos que aportan una capacidad predictiva lo suficientemente buena al modelo. En esta variante para obtener la medida del rendimiento se utiliza la validación cruzada, y el modelo predictivo puede ser elegido arbitrariametne según convenga.

La primera tarea para implementar el algoritmo es separar la variable objetivo de las predictoras, y eliminar de las predictoras el campo "PassengerID", ya que no aporta infomración útil al modelo.

[19]:
```
df_predictors = df.drop(['Survived','PassengerId'], axis=1)
df_target = df['Survived']
```

Luego es necesario crear la instancia del modelo predictivo que se utilizará para calcular el error (*Random Forest*), para posteriormente aplicar el algoritmo. Nótese que es necesario establecer algunos parámetros en la invocación de la función RFECV, donde además de especificar el modelo es necesario indicar los siguientes parámetros:

- step: Número de atributos eliminados en cada iteración.
- cv: Hace referencia al tipo de división de los *folds*, en este caso se establece una división estratificada de 10 *folds*.
- scoring: Establece la métrica del rendimiento deseada para comprobar la calidad del modelo, en este caso se toma la exactitud (acurancy).
- min_features_to_select: Determina la cantidad mínima de atributos.
- n_jobs: Habilita el *multiprocessing*, este algoritmo puede ser computacionalmetne muy pesado y es conveniente utilizar una ayuda extra con varios procesadores. En este caso el dataset no es muy grande y tiene un número bajo de atributos, pero es igualmente interesante tener este factor en cuenta.

[20]:
```
model = RandomForestClassifier(random_state=101)
rfecv = RFECV(estimator=model,
              step=1,
              cv=StratifiedKFold(10),
              scoring='accuracy',
              n_jobs=3)
```

Una vez creada la base del algoritmo, se alimenta con las divisiones creadas anteriormente, y utilizando el atributo *n_features* se puede obtener el número óptimo de atributos devuelto automáticamente por el algoritmo.

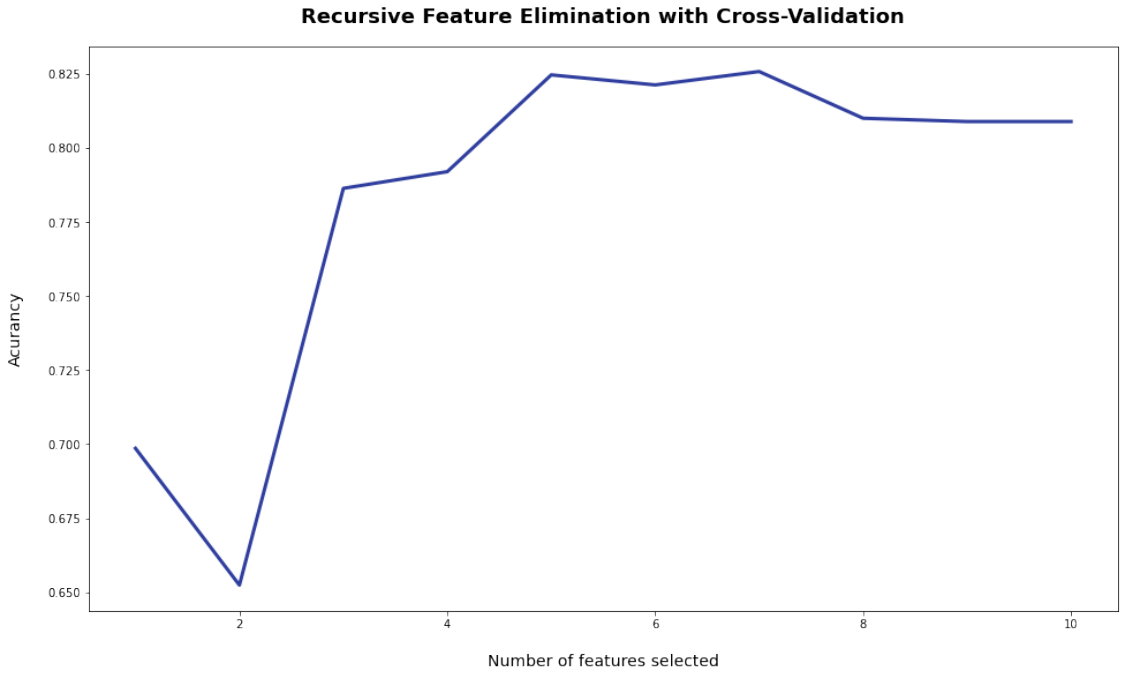
[21]:
```
rfecv.fit(df_predictors, df_target)
print('Optimal number of features: {}'.format(rfecv.n_features_))
```

```
Optimal number of features: 7
```

Como resultado se obtiene que el número óptimo de atributos es 7. Para obtener una visión más objetiva de este resultado se ofrece la siguiente g eráfican la que se muestra el valor de la exactitud para cada valor del número de atributos obtenido.

[22]:
```
plt.figure(figsize=(16, 9))
plt.title('Recursive Feature Elimination with Cross-Validation', fontsize=18,␣
 ↪fontweight='bold', pad=20)
plt.xlabel('Number of features selected', fontsize=14, labelpad=20)
plt.ylabel('Acurancy', fontsize=14, labelpad=20)
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_,␣
 ↪color='#303F9F', linewidth=3)

plt.show()
```



A continuación se imprimen los atributos menos importantes y se crea un nuevo *dataframe* con los 7 atributos obtenidos en el análisis anterior. La propiedad *support* _ del objeto rfecv muestra en una lista de *booleanos* cuáles de los indices de los atributos entra o no entre los 6 más importantes, con el apoyo de esta se realizan las tareas anteriormente descritas.

[23]:
```
print(list(df_predictors.columns[np.where(rfecv.support_ == False)[0]]))

# Desechando los atributos menos importantes
df_predictors.drop(df_predictors.columns[np.where(rfecv.support_ == False)[0]],␣
 ↪axis=1, inplace=True)
df_predictors.head()
```

```
['Embarked_C', 'Embarked_Q', 'Embarked_S']
```

[23]:

	Pclass	Age	SibSp	Parch	Fare	Sex_male	Sex_female
0	3	22.0	1	0	7.25000	1	0
1	1	38.0	1	0	56.37664	0	1
2	3	26.0	0	0	7.92500	0	1
3	1	35.0	1	0	53.10000	0	1
4	3	35.0	0	0	8.05000	1	0

Para finalizar con este apartado es interesante conocer en que medida las variables seleccionadas son importantes para el modelo, por ello se muestra este valor de importancia obtenido de la propiedad *feature_importances*.

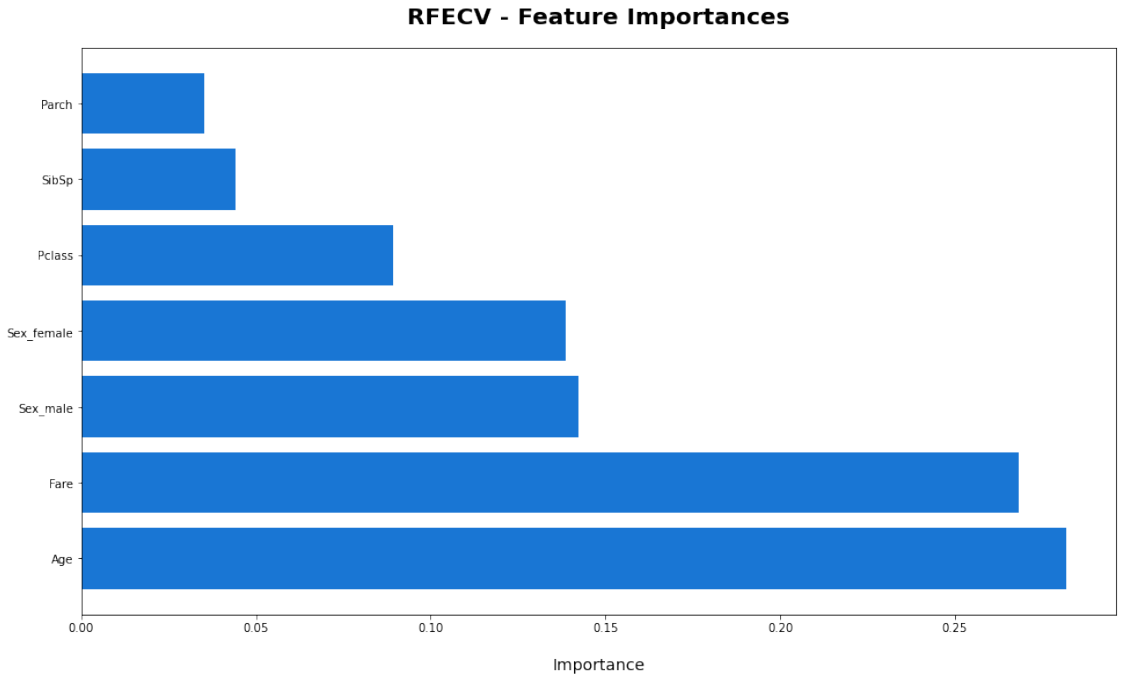
Para facilitar la creación de la gráfica se crea un nuevo *dataframe* con los nombres de las columnas y los valores de importancia de los atributos, también se ordenan ascendentemente para facilitar la comprensión de la misma.

[24]:
```
# Nuevo dataframe auxiliar para la visualización
dset = pd.DataFrame()

# Asignando los vlaores de imoprtancia y nombres de columnas
dset['attr'] = df_predictors.columns
dset['importance'] = rfecv.estimator_.feature_importances_

# Ordenando por importancia
dset = dset.sort_values(by='importance', ascending=False)


plt.figure(figsize=(16, 9))
plt.barh(y=dset['attr'], width=dset['importance'], color='#1976D4')
plt.title('RFECV - Feature Importances', fontsize=20, fontweight='bold', pad=20)
plt.xlabel('Importance', fontsize=14, labelpad=20)
plt.show()
```



5.2 Comprobación de la normalidad y homogeneidad de la varianza

5.2.1 Normalidad

Para comprobar si los datos tienen una distribución normal podemos utilizar el *estadístico de Anderson-Darling*. Este estadístico mide qué tan bien siguen los datos una distribución específica (gaussiana en nuestro caso).

Aplicando una prueba de contraste de hipótesis se puede comprobar la condición de normalidad de los datos. Las hipótesis serán las siguientes.

H_0 : Los datos de la muestra sigue una distribución normal.

H_1 : Los datos de la muestra no siguen una disribución normal.

Estableciendo el valor de significancia:

α en este caso será igual a los valores críticos (para cada nivel de significancia)que se obtienen como resultado del test AD.

Si $p <= \alpha$: Se rechaza la hipótesis nula, por lo tanto no está distribuida normalmente

Si $p > \alpha$: Se falla al hipótesis nula, por lo tanto si está distribuida normalmente

En nuestro caso tenemos tres variables no categóricas sobre las que podemos testear la normalidad: *Age*, *SibSp* y *Fare*.

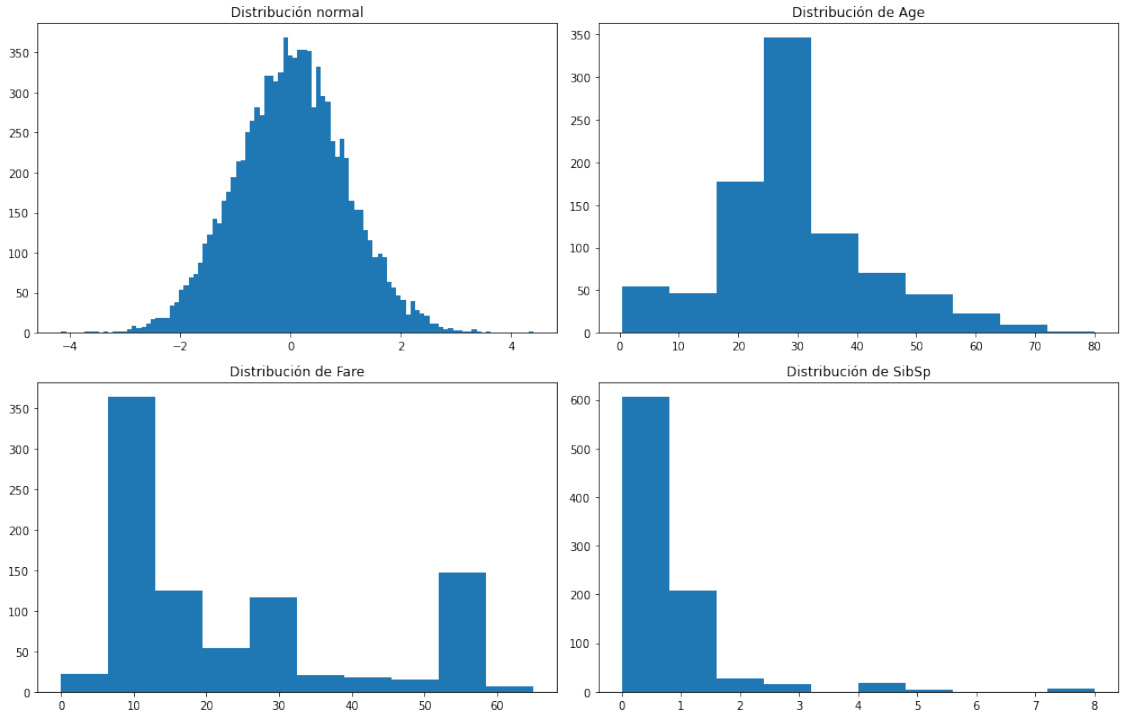
Inicialmente podemos representar sus histogramas para tener una idea de si las distribuciones parecen normales o no, esto es, si se aproximan a una campana de gauss.

[25]:
```
from numpy.random import randn

# Distribuciones de las variables no categóricas
fig, axs = plt.subplots(2,2, figsize=(14, 9))

# Distribución normal de referencia
axs[0,0].hist(randn(10000), 100)
axs[0,0].set_title('Distribución normal')
# Distribución de atributo Age
axs[0,1].hist(df_predictors.Age)
axs[0,1].set_title('Distribución de Age')
# Distribución de atributo Fare
axs[1,0].hist(df_predictors.Fare)
axs[1,0].set_title('Distribución de Fare')
# Distribución de atributo SibSp
axs[1,1].hist(df_predictors.SibSp)
axs[1,1].set_title('Distribución de SibSp')

fig.tight_layout()
```



De primera ya vemos que si comparamos con la distribución normal de referencia (primera figura), ninguna distribución parece acercarse a una distribución normal. *Age* parece acercarse algo más que el resto, pero aún así se aleja bastante de una normal.

Comprobaremos ahora con el método Anderson-Darling si se pueden considerar normales.

[26]:
```
from scipy.stats import anderson

# Comprobación para Age
result = anderson(df_predictors.Age)
print("Comprobación de la normalidad de Age")
print('Statistic: %.3f' % result.statistic)
p = 0
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < result.critical_values[i]:
        print('%.3f: %.3f, los datos siguen una distribución normal' % (sl, cv))
    else:
        print('%.3f: %.3f, los datos no siguen una distribución normal' % (sl,␣
 ↪cv))
```

```
Comprobación de la normalidad de Age
Statistic: 15.286
15.000: 0.573, los datos no siguen una distribución normal
10.000: 0.653, los datos no siguen una distribución normal
5.000: 0.783, los datos no siguen una distribución normal
2.500: 0.914, los datos no siguen una distribución normal
1.000: 1.087, los datos no siguen una distribución normal
```

Obtenemos respuestas según lo estrictos que seamos, cuanto mas alto el nivel de significancia (1, 205, 5, 10, 15), más estrictos somos y es más dificil que la distribución se ajuste a una normal. En este caso, con ningún nivel de significancia obtenemos un resultado positivo.

Comprobamos que pasa lo mismo para las otras dos variables y que ninguna sigue una distribución normal.

[27]:
```
# Comprobación para Fare
result = anderson(df.Fare)
print("Comprobación de la normalidad de Fare")
print('Statistic: %.3f' % result.statistic)
p = 0
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < result.critical_values[i]:
        print('%.3f: %.3f, los datos siguen una distribución normal' % (sl, cv))
    else:
        print('%.3f: %.3f, los datos no siguen una distribución normal' % (sl,␣
 ↪cv))

# Comprobación para SibSp
result = anderson(df.SibSp)
print("\nComprobación de la normalidad de SibSp")
print('Statistic: %.3f' % result.statistic)
p = 0
```

```
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < result.critical_values[i]:
        print('%.3f: %.3f, los datos siguen una distribución normal' % (sl, cv))
    else:
        print('%.3f: %.3f, los datos no siguen una distribución normal' % (sl,␣
 ↪cv))
```

```
Comprobación de la normalidad de Fare
Statistic: 67.372
15.000: 0.573, los datos no siguen una distribución normal
10.000: 0.653, los datos no siguen una distribución normal
5.000: 0.783, los datos no siguen una distribución normal
2.500: 0.914, los datos no siguen una distribución normal
1.000: 1.087, los datos no siguen una distribución normal

Comprobación de la normalidad de SibSp
Statistic: 146.788
15.000: 0.573, los datos no siguen una distribución normal
10.000: 0.653, los datos no siguen una distribución normal
5.000: 0.783, los datos no siguen una distribución normal
2.500: 0.914, los datos no siguen una distribución normal
1.000: 1.087, los datos no siguen una distribución normal
```

Como ya habíamos predicho con las ilustraciones de las distribuciones, ninguna de ellas cumple con la condición para aceptar la Hipótesis nula H_0, con lo que ninguno presenta una distribución normal. Esto en general puede ser por diversas causas: Valores extremos (que ya habíamos eliminado), superposición de dos procesos de muestreo, insuficiente discriminación de datos, muestreo de una población ordenada, valores cerca de cero o con un límite natural o simplemente los datos siguen otra distribución.

En el titanic realmente había mas de 2000 pasajeros y en nuestro dataset tenemos en torno a 900, por lo que no sabemos cómo se hizo realmente la toma de los datos y como afecta esto a las distribuciones.

5.2.2 Homogeneidad de la varianza

Para comprobar la homgeneidad de la varianza utilizaremos el *test de levene*, pero podríamos usar otro entre las distintas opciones.

El *test de Levene* es una prueba estadística que se usa para evaluar la igualdad entre las varianzas de dos o más grupos. En este caso lo usaremos para ver si la varianza de las variables predictoras respecto a la variable objetivo es la misma, a esto es a lo que se le llama homogeneidad de la varianza.

De manera similar a la efectuada en la comprobación de la normalidad, se aplican pruebas de contraste de hipótesis, estableciendo como H_0 la hipótesis que determina que las varianzas son iguales. Es necesario destacar que se utilizará el valor de la mediana para determinar la centralidad ya que los datos no muestran una distribución normal.

El valor de significancia α en este caso se establece su valor más típico de $0,05$.

Para realizar este test se tienen en cuenta los valores de las variables predictoras según su distribución en la variable objetivo. Entonces utilizando la función *levene()* del paquete scipy.stats se aplica dicho test.

[28]:
```
from scipy.stats import levene

age = df_predictors['Age']
SibSp = df_predictors['SibSp']
Fare = df_predictors['Fare']
target = df_target

p_age = levene(age[target==1], age[target==0], center='median').pvalue
p_sibsp = levene(SibSp[target==1], SibSp[target==0], center='median').pvalue
p_fare = levene(Fare[target==1], Fare[target==0], center='median').pvalue

p_values = {'Age': p_age , 'SibSp': p_sibsp, 'Fare': p_fare}

for i in p_values:
    if p_values[i] < 0.05:
        print('Los datos de la variable %s no presentan Homogeneidad de las␣
 ↪varianzas' % (i))
    else:
        print('Los datos de la variable %s presentan Homogeneidad de las␣
 ↪varianzas' % (i))
```

```
Los datos de la variable Age no presentan Homogeneidad de las varianzas
Los datos de la variable SibSp presentan Homogeneidad de las varianzas
Los datos de la variable Fare no presentan Homogeneidad de las varianzas
```

Como se aprecia en los resultados del análisis solo la variable "SibSp" presenta homogeneidad en las varianzas. En el caso de la variable "Age" presenta un p-value cercano al valor de referencia pero sigue siendo inferior, esto puede ser por el tratamiento que se le dió a los valores extremos o a los ausentes, recordando que esta variable tenía la peor completitud entre todas las predictoras.

Habrá que tener en cuenta las conclusiones sacadas en este apartado a la hora de aplicar las diferentes pruebas estadísticas, ya que algunas de estas asumen la normalidad y la hocedasticidad en los conjuntos de datos.

5.3 Análisis de correlaciones

En este apartado se comprueba la realción entre las diferentes variables predictoras y la variable objetivo, medido en el valor de la correlación. La manera más intuitiva de visualizar esta correlación es con un mapa de calor, en el que a partir de una matriz de correlaciones se muestran en colores los diferentes niveles de valores.

Para crear este mapa se ha utilizado la función *heatmap* del paquete *seaborn*. Se han concatenado

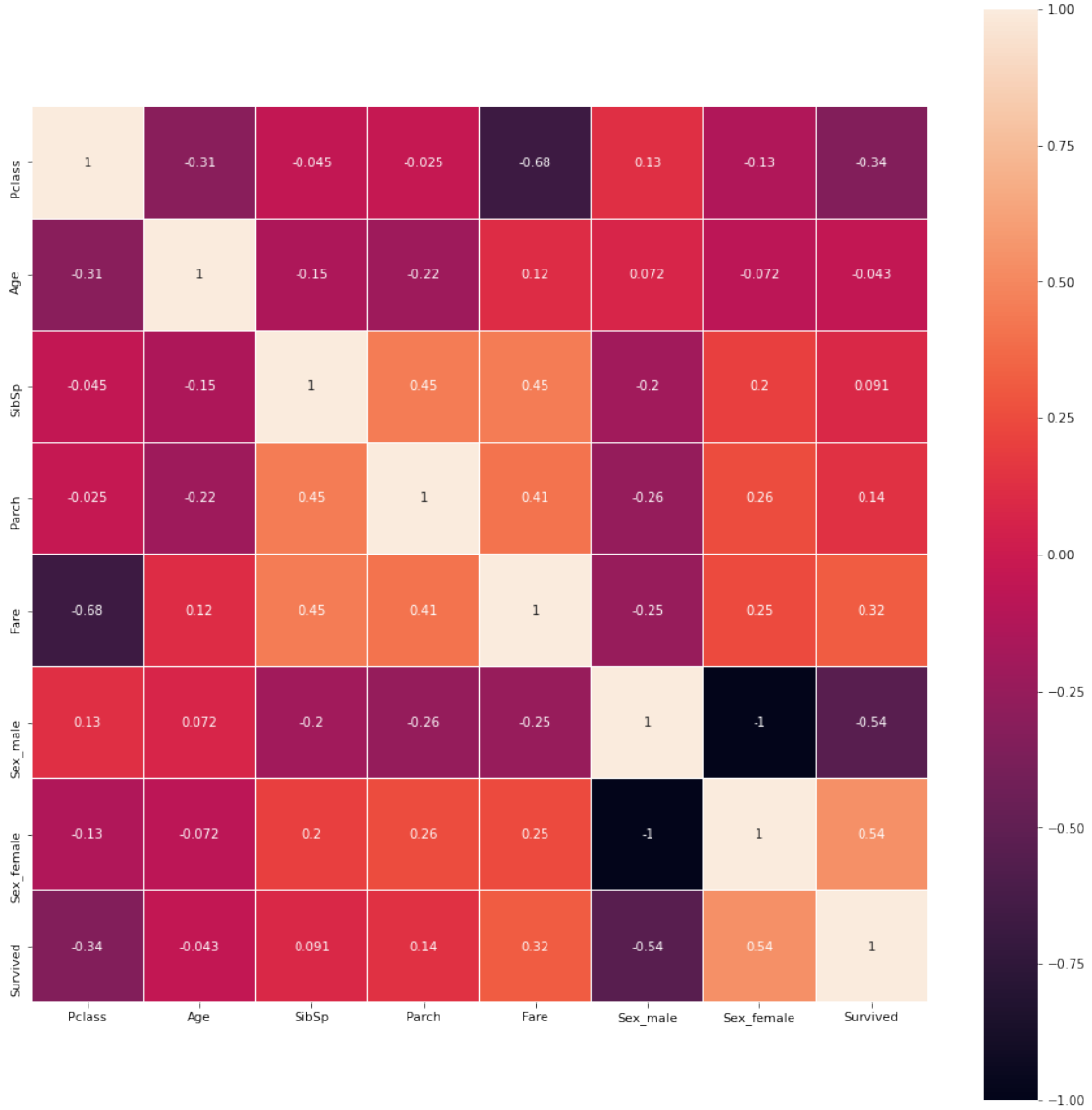
los *dataframes* de las variables predictoras y la objetivo para realizar el cálculo, destacando que el método elegido para realizar el mismo es el de "Spearman", variante no paramétrica utilizada cuando los datos no siguen una distribución normal.

[29]:
```
import seaborn as sns

df_heatmap = pd.concat([df_predictors, df_target], axis=1)

corr = df_heatmap.corr(method = 'spearman')

plt.figure(figsize=(16,16))
sns.heatmap(corr,linewidths=0.1,
            square=True,  linecolor='white', annot=True)
plt.show()
```



Como resultado se obtiene que ninguna variable predictora muestra una fuerte correlación con otra variable predictora, lo que determina que no hay redundancia en los datos y cada una aporta información útil al modelo. También se puede apreciar que algunas variables presentan valores muy interesantes sobre la variable objetivo, como es el caso de *Pclass*, *Fare* o el sexo, pero esto realmente no dice si es estadísticamente significativo a la hora de tomar en cuenta los resultados. Para commprobarlo se realiza un test de correlaciones, en el que la hipótesis nula es que los datos no están correlacionados y el valor de significancia α es el típico 0.05.

Para calcular los valores de p necesarios en el análisis se utiliza la función *spearmanr* del paquete *scipy.stats*, y con la ayuda de las *list comprehensions* se crean dos listas con los valores de los coeficientes de correlación y los *p-values* para posteriormente compararlos con el valor de significancia.

[30]:
```
from scipy.stats import spearmanr

col_names = df_predictors.columns.values
p_values = [spearmanr(df_predictors[col_names[i]], df_target).pvalue for i in␣
 ↪range(len(col_names))]
corr_values = [spearmanr(df_predictors[col_names[i]], df_target).correlation␣
 ↪for i in range(len(col_names))]


for i in range(len(col_names)):
    if p_values[i] < 0.05:
        print('La correlación: %.2f  con valor p = %.2f entre %s y la variable␣
 ↪objetivo es estadísticamente significativa' % (
            corr_values[i], p_values[i], col_names[i]))
    else:
        print('La correlación:  %.2f con valor p = %.2f entre %s y la variable␣
 ↪objetivo NO es estadísticamente significativa' % (
            corr_values[i], p_values[i], col_names[i]))
```

```
La correlación: -0.34  con valor p = 0.00 entre Pclass y la variable objetivo es
estadísticamente significativa
La correlación:  -0.04 con valor p = 0.20 entre Age y la variable objetivo NO es
estadísticamente significativa
La correlación: 0.09  con valor p = 0.01 entre SibSp y la variable objetivo es
estadísticamente significativa
La correlación: 0.14  con valor p = 0.00 entre Parch y la variable objetivo es
estadísticamente significativa
La correlación: 0.32  con valor p = 0.00 entre Fare y la variable objetivo es
estadísticamente significativa
La correlación: -0.54  con valor p = 0.00 entre Sex_male y la variable objetivo
es estadísticamente significativa
La correlación: 0.54  con valor p = 0.00 entre Sex_female y la variable objetivo
es estadísticamente significativa
```

Como se puede observar en los resultados del análisis, la única variable que no muestra una significancia estadística en el cálculo de las correlaciones es la variable *Age*.

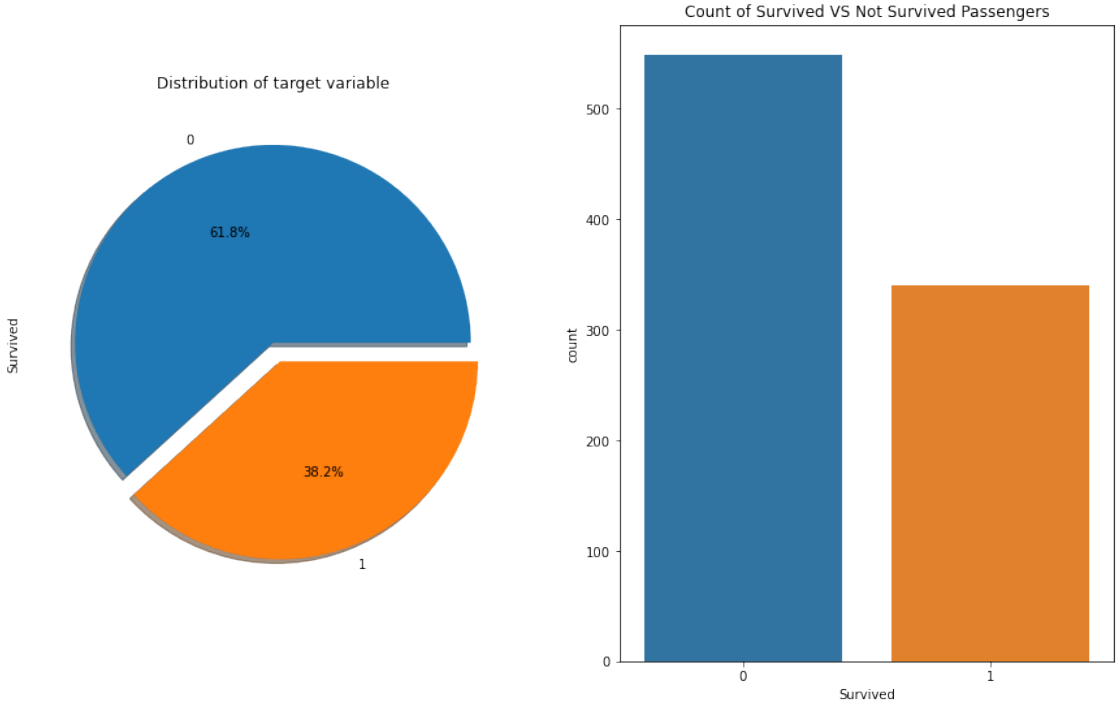
5.4 Análisis Exploratorio (EDA)

Para finalizar con este apartado de análisis de datos se realizará un anális exploratorio (EDA). En este, se utilizarán visualizaciones y cálculos sencillos para intentar descibrir patrones en los datos que no han sido descubiertos por los análisis estadísticos. Para realizar este análisis, solamente se utilizarán algunas de las variables obtenidas de la selección de atributos. Es cierto que las demás que no fueron seleccionadas también pueden aportar conocimiento, pero como estas no se toman en consideración para el modelo se toma la desición de no utilizarlas en este punto.

5.4.1 Survived

Para comenzar, es interesante conocer la distribución de la variable objetivo. En este caso se observa que hay una gran diferencia entre las personas sobrevivientes y las que no, representando los sobrevivientes solamente un 38.2%.

[31]:
```
f,ax=plt.subplots(1,2,figsize=(15,9))
df_target.value_counts().plot.pie(explode=[0,0.1],autopct='%1.
 ↪1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Distribution of target variable')
sns.countplot(df_target,ax=ax[1])
ax[1].set_title('Count of Survived VS Not Survived Passengers')
plt.show()
```



5.4.2 Sexo

Para analizar la influencia del sexo se utiliza la variable 'Sex_male', que denota a los hombres con un 1 y a las mujeres con un 0. Inicialmente se realiza una agrupación por sexo y se muestra la realación con la variable objetivo.

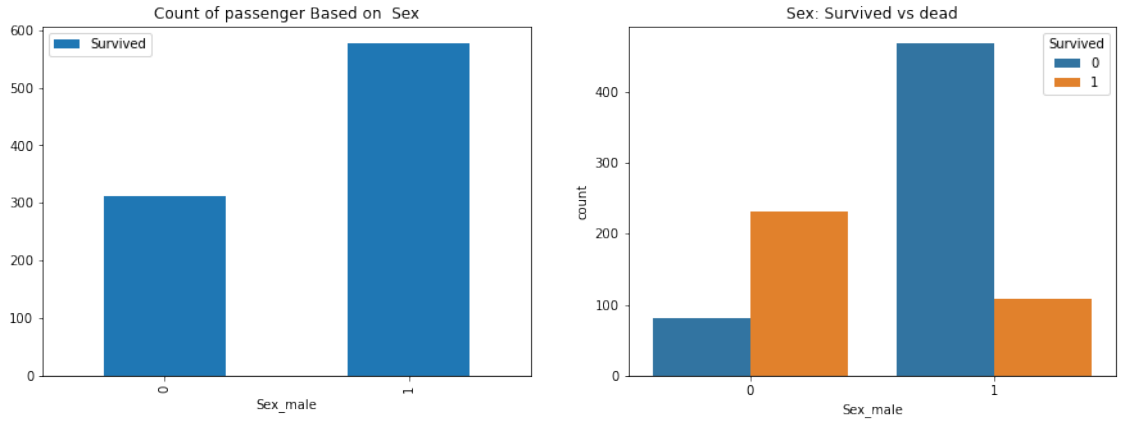
[32]:
```
print(df_heatmap.groupby(['Sex_male', 'Survived'])['Survived'].count())
```

```
Sex_male  Survived
0         0            81
          1           231
1         0           468
          1           109
Name: Survived, dtype: int64
```

Como se puede apreciar, de los supervivientes 231 eran mujeres y solo 109 hombres, teniendo en cuenta que eran muhcos más hombres que mujeres este dato indica que la mortalidad entre los hombres fue muy alta.

En las siguientes figuras se muestran estos datos, nótese como se ve una calra diferencia entre ambos sexos y los niveles de supervivencia.

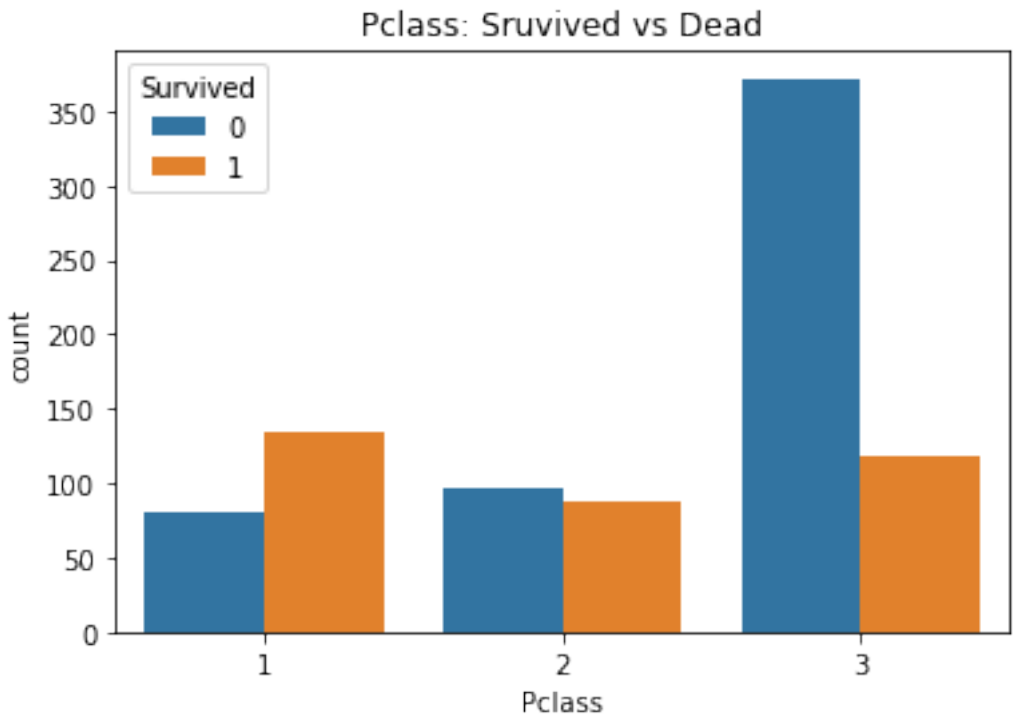
[33]:
```
# Representando gráficas de cantidad y supervivencia por sexo
f,ax=plt.subplots(1,2,figsize=(15,5))
df_heatmap[['Sex_male','Survived']].groupby(['Sex_male']).count().plot.
 ↪bar(ax=ax[0])
ax[0].set_title('Count of passenger Based on  '+ 'Sex')
sns.countplot('Sex_male',hue='Survived',data=df_heatmap,ax=ax[1])
ax[1].set_title( 'Sex'+ ': Survived vs dead')
plt.show()
```



5.4.3 Pclass (Clase)

En el caso de esta variable se procede directamente a graficar los valores de incidencia sobre la variable objetivo.

[34]:
```
sns.countplot('Pclass', hue='Survived', data=df_heatmap)
plt.title('Pclass: Sruvived vs Dead')
plt.show()
```



Lo más destacable de estos resultados es el valor de los fallecidos en la tercera clase y el vaor de los supervivientes en las dos primeras.

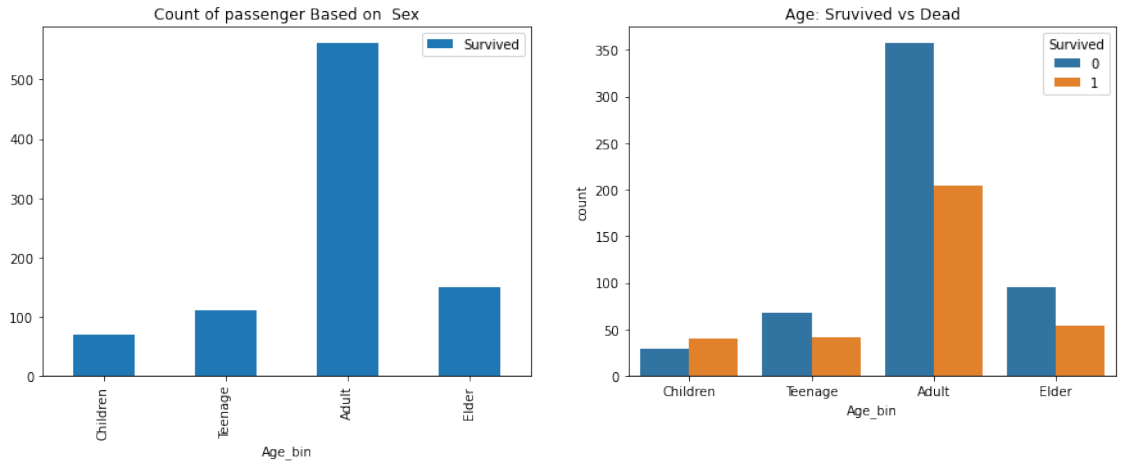
5.4.4 Age

Para analizar la edad se opta por discretizar sus valores agrupándolos en 4 segmentos, estos segmentos describen mucho mejor el comportamiento de la influencia de las edades a la hora de acceder a los botes salvavidas y por ende a la supervivencia. En la gráfica de la izquierda se puede apreciar el monto total por grupos, lo que da una idea de cuantas personas sobrevivieron por rango de edades.

[35]:
```
# Discretizando la variable Age
df_heatmap['Age_bin'] = pd.cut(df_heatmap['Age'], bins=[0,12,20,40,120],␣
 ↪labels=['Children','Teenage','Adult','Elder'])

f,ax=plt.subplots(1,2,figsize=(15,5))
```

```
df_heatmap[['Age_bin','Survived']].groupby(['Age_bin']).count().plot.
 ↪bar(ax=ax[0])
ax[0].set_title('Count of passenger Based on  '+ 'Sex')
sns.countplot('Age_bin', hue='Survived', data=df_heatmap)
ax[1].set_title( 'Age: Sruvived vs Dead')
plt.show()
```



Es evidente que el número de personas fallecidas entre los adultos fue muy grande, mientras que entre los niños y los adolescentes presentan mejores cifras.

6 Conclusiones

Después de haber realizado los distintos análisis sobre el conjunto de datos se puede dar por resuelto el problema planteado inicialmente. Se han detectado las variabels con mayor influencia en la variable objetivo y se han identificado características de los grupos de personas que determinaron en su momento la supervivencia.

Teniendo en cuenta los resultados obtenidos tanto de los análisis estadísticos como de los exploratorios se pueden resumir las siguientes conclusiones de los datos.

- Las mujeres presentan mejor probabilidad de supervivencia que los hombres.
- Proporcionalmente junto con las mujeres los niños y adolescentes presentan mejores porcientos de supervicenvia.
- Hay una clara diferencia entre las clases sociales, las personas de primera clase presentan unos datos de supervivencia mucho mejores que las de las demás clases, y si se suman los valores de primera y segunda clase resumen la mayoría de los supervivientes.

7 Contribuciones

Cotribuciones	Firma
Investigación previa	JG, RL
Redacción de las respuestas	JG, RL
Desarrollo código	JG, RL