Exploratory analysis - Titanic (extended)

June 25, 2022

##

Análisis exploratorio. Caso: Titanic

Carga de librerías

[1]:
```
# Cargado de librerías
import pandas as pd
import os
import numpy as np
```

Importado de datos

[16]:
```
# Carga de datos
path = r"C:\Users\Carlos\OneDrive\Formación\Python\Case_Analysis\Titanic"
file = "train.csv"
df = pd.read_csv(os.path.join(path,file), sep = ",")
```

0.0.1 Etapa 1: Formulación del problema general

1. ¿Cuáles son la características principales del dataset?
2. ¿Qué particularidades tenían las personas con mayor probabilidad de sobrevivir?

0.0.2 Etapa 2: Barrido general de datos

Existen variables que tienen valores faltantes

[3]:
```
df.info(verbose = True, show_counts = True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 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   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
```

```
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

0.0.3 Etapa 3: Reconocer los tipos de datos y su escala de medición

[29]:
```
df.head(3)
```

[29]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       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

   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
```

[28]:
```
df.dtypes
```

[28]:
```
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
```

0.0.4 Etapa 4: Tranformación de datos

[17]:
```
# Tratamiento de valores faltantes
df1 =  df.fillna(value={'Age':0,
                        'Cabin':'null',
                        'Embarked': 'null'})

# Cambio de valores
# Cambios sobre variables con etiquetas numéricas y con abreviaturas para hacer
 ↪más comprensible el dataset
# Tome en cuenta que esto es opcional y el proceso inverso (a números) es una
 ↪parte de la creación de variables dummy
# ¿Aumentaría el uso de la memoria si reemplzamos la variables dummy?
change = {"Survived"  : {1: "Survived", 0: "Not survived"},
          "Pclass"    : {1: "First", 2: "Second", 3: "Third"},
          "Sex"       : {"male": "Male", "female": "Female"},
          "Embarked"  : {"C":"Cherbourg", "Q": "Queenstown", "S":
 ↪"Southampton"}}

df1 = df1.replace(change)

# Cambio del tipo de dato para la variable 'Age'
df1['Age'] = df1['Age'].apply(np.floor).astype('int64')
```

[21]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    object
 2   Pclass       891 non-null    object
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    int64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        891 non-null    object
 11  Embarked     891 non-null    object
dtypes: float64(1), int64(4), object(7)
memory usage: 83.7+ KB
```

Transformación detallada **Tratamiento de valores faltantes** Todos los valores faltantes serán reemplzados por un 0 para varibles numéricas y 'null' para variables categóricas

[4]:
```
df1 =  df.fillna(value={'Age':0,
                        'Cabin':'null',
                        'Embarked': 'null'})
```

[5]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 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   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        891 non-null    object
 11  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[58]:
```
df1.head()
```

[58]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	Not survived	Third	Braund, Mr. Owen Harris	Male	22.0	1	0	A/5 21171	7.2500	NaN	Southampton
1	2	Survived	First	Cumings, Mrs. John Bradley (Florence Briggs Th…	Female	38.0	1	0	PC 17599	71.2833	C85	Cherbourg
2	3	Survived	Third	Heikkinen, Miss. Laina	Female	26.0	0					
3	4	Survived	First	Futrelle, Mrs. Jacques Heath (Lily May Peel)	Female	35.0	1					
4	5	Not survived	Third	Allen, Mr. William Henry	Male	35.0	0					

```
2      0  STON/O2. 3101282   7.9250   NaN  Southampton
3      0            113803  53.1000  C123  Southampton
4      0            373450   8.0500   NaN  Southampton
```

Cambio de valores en el dataset

[7]:
```
# Cambios sobre variables con etiquetas numéricas y con abreviaturas para hacer
 ↪más comprensible el dataset
# Tome en cuenta que esto es opcional y el proceso inverso (a números) es una
 ↪parte de la creación de variables dummy
# ¿Aumentaría el uso de la memoria si reemplzamos la variables dummy?

change = {"Survived"  : {1: "Survived", 0: "Not survived"},
          "Pclass"    : {1: "First", 2: "Second", 3: "Third"},
          "Sex"       : {"male": "Male", "female": "Female"},
          "Embarked"  : {"C":"Cherbourg", "Q": "Queenstown", "S":
 ↪"Southampton"}}

df1 = df1.replace(change)
```

Cambio de tipo de dato en la variable 'Age'

[65]:
```
df["Age"].value_counts()
```

[65]:
```
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
55.50     1
0.92      1
23.50     1
74.00     1
Name: Age, Length: 88, dtype: int64
```

[13]:
```
df1['Age'] = df1['Age'].apply(np.floor).astype('int64')
df1.dtypes
```

[13]:
```
PassengerId      int64
Survived        object
Pclass          object
Name            object
Sex             object
Age              int64
SibSp            int64
Parch            int64
```

```
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

[14]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    object
 2   Pclass       891 non-null    object
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    int64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        891 non-null    object
 11  Embarked     891 non-null    object
dtypes: float64(1), int64(4), object(7)
memory usage: 83.7+ KB
```

0.0.5 Etapa 5: Selección de estadísticas descriptivas

Consiste en identificar las medidas calculadas que se aplicarán en las variables del dataset.

[20]:
```
# Análisis para todo tipo de variables
df1.describe(include = "all")
```

[20]:

	PassengerId	Survived	Pclass	Name	Sex
count	891.000000	891	891	891	891
unique	NaN	2	3	891	2
top	NaN	Not survived	Third	Braund, Mr. Owen Harris	Male
freq	NaN	549	491	1	577
mean	446.000000	NaN	NaN	NaN	NaN
std	257.353842	NaN	NaN	NaN	NaN
min	1.000000	NaN	NaN	NaN	NaN
25%	223.500000	NaN	NaN	NaN	NaN
50%	446.000000	NaN	NaN	NaN	NaN
75%	668.500000	NaN	NaN	NaN	NaN
max	891.000000	NaN	NaN	NaN	NaN

```
                Age       SibSp       Parch  Ticket        Fare Cabin  \
count    891.000000  891.000000  891.000000     891  891.000000   891
unique          NaN         NaN         NaN     681         NaN   148
top             NaN         NaN         NaN  347082         NaN  null
freq            NaN         NaN         NaN       7         NaN   687
mean      23.783389    0.523008    0.381594     NaN   32.204208   NaN
std       17.597344    1.102743    0.806057     NaN   49.693429   NaN
min        0.000000    0.000000    0.000000     NaN    0.000000   NaN
25%        6.000000    0.000000    0.000000     NaN    7.910400   NaN
50%       24.000000    0.000000    0.000000     NaN   14.454200   NaN
75%       35.000000    1.000000    0.000000     NaN   31.000000   NaN
max       80.000000    8.000000    6.000000     NaN  512.329200   NaN

            Embarked
count            891
unique             4
top     Southampton
freq             644
mean             NaN
std              NaN
min              NaN
25%              NaN
50%              NaN
75%              NaN
max              NaN
```

Análisis estadístico preliminar

[15]:
```
# Análsis de una columna usando el método apply
# Tomar en cuenta que no deben existir valores faltantes, pues la función "len" no devolvería reusltados correctos
dataset["Fare"].apply({'sum':sum,'min':min,'max':max,'count':len,'mean':np.mean})
```

[15]:
```
sum      28693.949300
min          0.000000
max        512.329200
count      891.000000
mean        32.204208
Name: Fare, dtype: float64
```

[16]:
```
# Agregando columnas con el método aggregate. Funciona con una o más columnas.
# Tomar en cuenta que no deben existir valores faltantes pues la función "len" no devolvería reusltados correctos.
dataset[["Fare","Age","Parch"]].aggregate({sum,min,max,len,np.mean,np.var})
```

[16]:

	Fare	Age	Parch
len	891.000000	891.000000	891.000000
sum	28693.949300	21205.170000	340.000000
var	2469.436846	211.019125	0.649728
min	0.000000	0.420000	0.000000
mean	32.204208	29.699118	0.381594
max	512.329200	80.000000	6.000000

[17]:
```
# Análisis para variables cualitativas
dataset2.describe(include = "object")
```

[17]:

	Survived	Pclass	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	891	891	204	889
unique	2	3	891	2	681	147	3
top	Not survived	Third	Braund, Mr. Owen Harris	Male	347082	B96 B98	Southampton
freq	549	491	1	577	7	4	644

[18]:
```
# Por defecto solo se toman solamente las variables cuantitativas
dataset2.describe()
```

[18]:

	PassengerId	Age	SibSp	Parch	Fare
count	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	29.699118	0.523008	0.381594	32.204208
std	257.353842	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	38.000000	1.000000	0.000000	31.000000
max	891.000000	80.000000	8.000000	6.000000	512.329200

[19]:
```
# Análisis para todo tipo de variables
dataset2.describe(include = "all")
```

[19]:

	PassengerId	Survived	Pclass	Name	Sex	Ticket	Cabin	Embarked
count	891.000000	891	891	891	891	891	204	889
unique	NaN	2	3	891	2	681	147	3
top	NaN	Not survived	Third	Braund, Mr. Owen Harris	Male	347082	B96 B98	Southampton
freq	NaN	549	491	1	577	7	4	644
mean	446.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	257.353842	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
25%      223.500000           NaN    NaN                    NaN   NaN
50%      446.000000           NaN    NaN                    NaN   NaN
75%      668.500000           NaN    NaN                    NaN   NaN
max      891.000000           NaN    NaN                    NaN   NaN

                Age       SibSp       Parch  Ticket        Fare    Cabin  \
count    714.000000  891.000000  891.000000     891  891.000000      204
unique          NaN         NaN         NaN     681         NaN      147
top             NaN         NaN         NaN  347082         NaN  B96 B98
freq            NaN         NaN         NaN       7         NaN        4
mean      29.699118    0.523008    0.381594     NaN   32.204208      NaN
std       14.526497    1.102743    0.806057     NaN   49.693429      NaN
min        0.420000    0.000000    0.000000     NaN    0.000000      NaN
25%       20.125000    0.000000    0.000000     NaN    7.910400      NaN
50%       28.000000    0.000000    0.000000     NaN   14.454200      NaN
75%       38.000000    1.000000    0.000000     NaN   31.000000      NaN
max       80.000000    8.000000    6.000000     NaN  512.329200      NaN

            Embarked
count            889
unique             3
top     Southampton
freq             644
mean             NaN
std              NaN
min              NaN
25%              NaN
50%              NaN
75%              NaN
max              NaN
```

Clasificación de variables Es importante clasificar las variables del dataset para proceder con el método de análisis correcto. Tome en cuenta que no todas las variables de tipo int64 o float64 son cuantitativas (estadísticamente), es necesario evaluar la escala de medición porque pueden existir variables nominales u ordinales expresadas numéricamente.

Primero se retirarán las variables que no aportan información relevante al análisis estadístico como "PassengerId", que funciona como una llave primaria para identificar elementos y filas dentro del dataset, y de igual manera con las columnas "Name" y "Ticket" debido a que ambas solo buscan identificar a un pasajero o registro y no conviene hacer agregaciones con sus datos. Es posible que estas variables sean útiles para otros porpósitos o dentro de otras secciones, pero eso se depende de la formulación del problema, el objetivo de análisis u otra necesidad de información particular.

[20]:
```
# Eliminando variables que no aportan información relevante al análisis␣
  ↪estadístico
removed_cols = ["PassengerId","Name","Ticket"]
dataset2 = dataset2.drop(removed_cols, axis = 1)
```

```
cols = (pd.DataFrame(dataset2.dtypes).
             reset_index().
             rename(columns = {"index": "Name", 0: "Type"}))

# Selección de variables cualitativas
qual_cols = cols[cols["Type"] == "object"]["Name"]
qualitative_data = dataset2[qual_cols]

# Selección de variables cunatitativas
quant_cols = cols[cols["Type"] != "object"]["Name"]
quantitative_data = dataset2[quant_cols]
```

[21]:
```
# Verificación de resultados
print(qualitative_data.head())
print("---------------------------")
print(quantitative_data.head())
```

```
       Survived Pclass     Sex Cabin     Embarked
0  Not survived  Third    Male   NaN  Southampton
1      Survived  First  Female   C85    Cherbourg
2      Survived  Third  Female   NaN  Southampton
3      Survived  First  Female  C123  Southampton
4  Not survived  Third    Male   NaN  Southampton
---------------------------
    Age  SibSp  Parch     Fare
0  22.0      1      0   7.2500
1  38.0      1      0  71.2833
2  26.0      0      0   7.9250
3  35.0      1      0  53.1000
4  35.0      0      0   8.0500
```

Análisis para una variable cualitativa

[22]:
```
# Análisis para una variable cualitativa
study_var = "Embarked"
qual_analysis = pd.DataFrame(qualitative_data[study_var])
qual_analysis = (qual_analysis.value_counts(ascending = False, dropna = False).
                 reset_index().
                 rename(columns = {0:"Frecuencia"}))

# Frecuencia relativa
qual_analysis.loc[:,('Frecuencia_rel')] = qual_analysis["Frecuencia"]/
 ↪sum(qual_analysis["Frecuencia"])
# Frecuencia acumulada
qual_analysis.loc[:,('Frecuencia_A')] = np.cumsum(qual_analysis["Frecuencia"])
# Frecuencia relativa acumulada
```

```
qual_analysis.loc[:,('Frecuencia_rel_A')] = np.
 ↪cumsum(qual_analysis["Frecuencia_rel"])

# Adecuación del dataframe resultante
qual_analysis.insert(loc = 0,  column = "Variable", value = study_var)
qual_analysis = qual_analysis.rename(columns = {study_var:"Categoría"})

qual_analysis
```

[22]:

	Variable	Categoría	Frecuencia	Frecuencia_rel	Frecuencia_A	\
0	Embarked	Southampton	644	0.722783	644	
1	Embarked	Cherbourg	168	0.188552	812	
2	Embarked	Queenstown	77	0.086420	889	
3	Embarked	NaN	2	0.002245	891	

	Frecuencia_rel_A
0	0.722783
1	0.911336
2	0.997755
3	1.000000

Análisis para todas las variables cualitativas La función que se define a continuación consolida todo el análisis en una tabla para todas las variables cualitativas.

[23]:
```
# Tranformación del análisis anterior a una función

def qual_fun_analysis(study_var):
# Análisis para una variable cualitativa
    qual_analysis = pd.DataFrame(qualitative_data[study_var])
    qual_analysis = (qual_analysis.value_counts(ascending = False, dropna =␣
 ↪False).
                     reset_index().
                     rename(columns = {0:"Frecuencia"}))

# Frecuencia relativa
    qual_analysis.loc[:,('Frecuencia_rel')] = qual_analysis["Frecuencia"]/
 ↪sum(qual_analysis["Frecuencia"])
# Frecuencia acumulada
    qual_analysis.loc[:,('Frecuencia_A')] = np.
 ↪cumsum(qual_analysis["Frecuencia"])
# Frecuencia relativa acumulada
    qual_analysis.loc[:,('Frecuencia_rel_A')] = np.
 ↪cumsum(qual_analysis["Frecuencia_rel"])

# Adecuación del dataframe resultante
    qual_analysis.insert(loc = 0,  column = "Variable", value = study_var)
```

```
    qual_analysis = qual_analysis.rename(columns = {study_var:"Categoría"})

# Resultado de la función
    return qual_analysis
```

[24]:
```
qual_fun_analysis("Embarked")
```

[24]:
```
    Variable    Categoría  Frecuencia  Frecuencia_rel  Frecuencia_A  \
0  Embarked  Southampton         644        0.722783           644
1  Embarked    Cherbourg         168        0.188552           812
2  Embarked   Queenstown          77        0.086420           889
3  Embarked          NaN           2        0.002245           891

   Frecuencia_rel_A
0          0.722783
1          0.911336
2          0.997755
3          1.000000
```

[25]:
```
# Modelo de análisis cualitativo consolidado
final_df = pd.DataFrame({'Variable' : [],
                         'Categoría' : [],
                         'Frecuencia' : [],
                         'Frecuencia_A' : [],
                         'Frecuencia_rel' : [],
                         'Frecuencia_rel_A' : []})

for i in qual_cols:
    x = qual_fun_analysis(i)
    final_df = final_df.append(x).reset_index(drop = True)

final_df
# Exportación de resultados en csv
#final_df.to_csv('Análisis de variables cualitativas - Titanic.csv')
```

[25]:
```
      Variable     Categoría  Frecuencia  Frecuencia_A  Frecuencia_rel  \
0     Survived  Not survived       549.0         549.0        0.616162
1     Survived      Survived       342.0         891.0        0.383838
2       Pclass         Third       491.0         491.0        0.551066
3       Pclass         First       216.0         707.0        0.242424
4       Pclass        Second       184.0         891.0        0.206510
..         …             …          …             …              …
154      Cabin           A10         1.0         891.0        0.001122
155   Embarked   Southampton       644.0         644.0        0.722783
156   Embarked     Cherbourg       168.0         812.0        0.188552
157   Embarked    Queenstown        77.0         889.0        0.086420
158   Embarked           NaN         2.0         891.0        0.002245
```

```
     Frecuencia_rel_A
0            0.616162
1            1.000000
2            0.551066
3            0.793490
4            1.000000
..                ...
154          1.000000
155          0.722783
156          0.911336
157          0.997755
158          1.000000

[159 rows x 6 columns]
```

Análisis para variables cuantitativas

[26]:
```
# Se deben hacer transformaciones con el dataset origianl y definir las␣
 ↪estadísticas que entrarán al análisis

# Listado de funciones de agregación de numpy
agg_fun_math = {'value':['count','size',
                         lambda x: x.isna().sum(),
                         'max', 'min',
                         np.nanmean, np.nanmedian,
                         lambda x : np.max(x) - np.min(x),
                         lambda x: np.var(x,ddof = 1),
                         lambda x: np.std(x,ddof = 1),
                         lambda x: ((np.std(x, ddof = 1)) / (np.mean(x)))]}

# Tranformacón de las variables cuantitativas
quant_analysis = (quantitative_data.melt(id_vars = None,
                                         var_name = "Variable" ,
                                         value_vars = list(quant_cols)).
                  groupby(["Variable"]).
                  agg(agg_fun_math).
                  round(2).
                  reset_index().
                  droplevel(level = 0, axis = 1).
                  rename(columns = {"": "variable",
                                    "<lambda_0>": "count_nan",
                                    "<lambda_1>": "range",
                                    "<lambda_2>": "var",
                                    "<lambda_3>": "std",
                                    "<lambda_4>": "CV"}))
```

```
quant_analysis
```

[26]:

	variable	count	size	count_nan	max	min	nanmean	nanmedian	range	\
0	Age	714	891	177	80.00	0.42	29.70	28.00	79.58	
1	Fare	891	891	0	512.33	0.00	32.20	14.45	512.33	
2	Parch	891	891	0	6.00	0.00	0.38	0.00	6.00	
3	SibSp	891	891	0	8.00	0.00	0.52	0.00	8.00	

	var	std	CV
0	211.02	14.53	0.49
1	2469.44	49.69	1.54
2	0.65	0.81	2.11
3	1.22	1.10	2.11

Distribuciones de frecuencias para ciertas variables cuantitativas Primero se mostrará los pasos para una variable individual, luego se consolidará el análisis para todo el dataset.

1. Número de clases que no se superponen Como regla general se recomienda utilizar entre 5 y 20 clases dependiendo de la cantidad de datos, sin embargo existen métodos de cálculo empíricos como la regla de Sturges: 1 + np.log2("Número de datos")

[27]:
```
# Cantidad de datos
dataset["Age"].count()
```

[27]: 714

[28]:
```
nclasses = np.ceil(1 + np.log2(dataset["Age"].count()))
nclasses
```

[28]: 11.0

2. Ancho de clase

[29]:
```
wclasses = round(((np.max(dataset["Age"]) - np.min(dataset["Age"])) /␣
 ↪nclasses), 0)
wclasses
```

[29]: 7.0

3. Límites de clase Existe cierta libertad para la selección de los anchos de clase siempre y cuando esté en función del ancho de clase y que cada elemento de datos pertenenezca a solo una de las clases.

El ancho de clase es un intervalo cerrado por izquierda (límite inferior) y abierto por derecha (límite superior)

4. Construcción de la distribución de frecuencias

[30]:
```
# Ejemplo para rellenar un areglo de dos dimensiones

a = np.empty([3,3], dtype = int)
# Se rellenará el arreglo por columnas
# También se puede llenar por filas, solo se debería cambiar a[j][i] por a[i][j]
b = 0
for i in range(3):
    for j in range(3):
        a[j][i] = b
        b = b + 1

print("The array is: \n", a)
```

```
The array is:
 [[0 3 6]
 [1 4 7]
 [2 5 8]]
```

[31]:
```
# Ejemplo para rellenar las frecuencias absolutas respecto a un límite

nclasses = 10
wclasses = 8
list1 = []
c = 0
lower_limit = 0
upper_limit = wclasses

for element in range(nclasses):
    print("Clase número ", element + 1)
    print("  Lower limit: ",lower_limit)
    print("  Upper limit: ",upper_limit)
    for element2 in list(dataset["Age"]):
        if element + 1 != nclasses:
            if element2 >= lower_limit and element2 < upper_limit:
                c = c + 1
        else:
            if element2 >= lower_limit and element2 <= upper_limit:
                c = c + 1
    list1.append(c)
    print("The value of c is ",c)
    c = 0
    print(list1,"\n")
    lower_limit = upper_limit
    upper_limit = upper_limit + wclasses
print(np.sum(list1))
```

```
Clase número  1
  Lower limit:  0
```

```
  Upper limit:  8
The value of c is  50
[50]

Clase número  2
  Lower limit:  8
  Upper limit:  16
The value of c is  33
[50, 33]

Clase número  3
  Lower limit:  16
  Upper limit:  24
The value of c is  164
[50, 33, 164]

Clase número  4
  Lower limit:  24
  Upper limit:  32
The value of c is  181
[50, 33, 164, 181]

Clase número  5
  Lower limit:  32
  Upper limit:  40
The value of c is  123
[50, 33, 164, 181, 123]

Clase número  6
  Lower limit:  40
  Upper limit:  48
The value of c is  74
[50, 33, 164, 181, 123, 74]

Clase número  7
  Lower limit:  48
  Upper limit:  56
The value of c is  50
[50, 33, 164, 181, 123, 74, 50]

Clase número  8
  Lower limit:  56
  Upper limit:  64
The value of c is  26
[50, 33, 164, 181, 123, 74, 50, 26]

Clase número  9
  Lower limit:  64
```

```
  Upper limit:  72
The value of c is  11
[50, 33, 164, 181, 123, 74, 50, 26, 11]

Clase número  10
  Lower limit:  72
  Upper limit:  80
The value of c is  2
[50, 33, 164, 181, 123, 74, 50, 26, 11, 2]

714
```

[32]:
```
# Ejemplo para rellenar las frecuencias absolutas respecto a un límite

nclasses = 10
wclasses = 8
list1 = []
c = 0
lower_limit = 0
upper_limit = wclasses

for element in range(nclasses):
    for element2 in list(dataset["Age"]):
        if element + 1 != nclasses:
            if element2 >= lower_limit and element2 < upper_limit:
                c = c + 1
        else:
            if element2 >= lower_limit and element2 <= upper_limit:
                c = c + 1
    list1.append(c)
    c = 0
    lower_limit = upper_limit
    upper_limit = upper_limit + wclasses
print(np.sum(list1))
```

```
714
```

[33]:
```
nclasses = round(1 + np.log2(dataset["Age"].count()),0)
wclasses = np.ceil(((np.max(dataset["Age"]) - np.min(dataset["Age"])) /␣
 ↪nclasses))

# Creación de los límites superiores e inferiores
a = np.zeros([int(nclasses),3], dtype = np.int64)
b = 0

for i in range(int(nclasses)):
    for j in range(2):
```

```
        a[i][j] = b
        b = b + wclasses
    b = a[i][j]


# Creación de las frecuencias absolutas
list1 = []
c = 0
lower_limit = 0
upper_limit = wclasses

for element in range(int(nclasses)):
    for element2 in list(dataset["Age"]):
        if element + 1 != nclasses:
            if element2 >= lower_limit and element2 < upper_limit:
                c = c + 1
        else:
            if element2 >= lower_limit and element2 <= upper_limit:
                c = c + 1
    list1.append(c)
    c = 0
    lower_limit = upper_limit
    upper_limit = upper_limit + wclasses

# Fusión de los límites y frecuencias absolutas
r1 = 0

for i2 in range(int(nclasses)):
    a[r1,2] = list1[r1]
    r1 = r1 + 1

a = pd.DataFrame(a,columns=["Low","High", "Frequency"])

# Frecuencia relativa
a.loc[:,('Frecuencia_rel')] = a["Frequency"]/sum(a["Frequency"])
# Frecuencia acumulada
a.loc[:,('Frecuencia_A')] = np.cumsum(a["Frequency"])
# Frecuencia relativa acumulada
a.loc[:,('Frecuencia_rel_A')] = np.cumsum(a["Frecuencia_rel"])

a
#a.to_csv('Distribución de frecuencias - Titanic.csv')
```

[33]:

	Low	High	Frequency	Frecuencia_rel	Frecuencia_A	Frecuencia_rel_A
0	0	8	50	0.070028	50	0.070028
1	8	16	33	0.046218	83	0.116246
2	16	24	164	0.229692	247	0.345938

3	24	32	181	0.253501	428	0.599440
4	32	40	123	0.172269	551	0.771709
5	40	48	74	0.103641	625	0.875350
6	48	56	50	0.070028	675	0.945378
7	56	64	26	0.036415	701	0.981793
8	64	72	11	0.015406	712	0.997199
9	72	80	2	0.002801	714	1.000000

Distribución de frecuencias para cualquier variable

[34]:
```
study_var = "SibSp"
nclasses = round(1 + np.log2(dataset[study_var].count()),0)
wclasses = np.ceil(((np.max(dataset[study_var]) - np.min(dataset[study_var])) /␣
 ↪nclasses))
print(nclasses)
print(wclasses)

# Creación de los límites superiores e inferiores
a = np.zeros([int(nclasses),3], dtype = np.int64)
b = 0

for i in range(int(nclasses)):
    for j in range(2):
        a[i][j] = b
        b = b + wclasses
    b = a[i][j]


# Creación de las frecuencias absolutas
list1 = []
c = 0
lower_limit = 0
upper_limit = wclasses

for element in range(int(nclasses)):
    for element2 in list(dataset[study_var]):
        if element + 1 != nclasses:
            if element2 >= lower_limit and element2 < upper_limit:
                c = c + 1
        else:
            if element2 >= lower_limit and element2 <= upper_limit:
                c = c + 1
    list1.append(c)
    c = 0
    lower_limit = upper_limit
    upper_limit = upper_limit + wclasses
```

```
# Fusión de los límites y frecuencias absolutas
r1 = 0

for i2 in range(int(nclasses)):
    a[r1,2] = list1[r1]
    r1 = r1 + 1


# Trasformación y adecuación
a = pd.DataFrame(a,columns=["Low","High", "Frequency"])

# Frecuencia relativa
a.loc[:,('Frecuencia_rel')] = a["Frequency"]/sum(a["Frequency"])
# Frecuencia acumulada
a.loc[:,('Frecuencia_A')] = np.cumsum(a["Frequency"])
# Frecuencia relativa acumulada
a.loc[:,('Frecuencia_rel_A')] = np.cumsum(a["Frecuencia_rel"])

a
#a.to_csv('Distribución de frecuencias - Titanic.csv')
```

```
11.0
1.0
```

[34]:

	Low	High	Frequency	Frecuencia_rel	Frecuencia_A	Frecuencia_rel_A
0	0	1	608	0.682379	608	0.682379
1	1	2	209	0.234568	817	0.916947
2	2	3	28	0.031425	845	0.948373
3	3	4	16	0.017957	861	0.966330
4	4	5	18	0.020202	879	0.986532
5	5	6	5	0.005612	884	0.992144
6	6	7	0	0.000000	884	0.992144
7	7	8	0	0.000000	884	0.992144
8	8	9	7	0.007856	891	1.000000
9	9	10	0	0.000000	891	1.000000
10	10	11	0	0.000000	891	1.000000

Distribución de frecuencias para todas las variables cuantitativas

[37]:
```
# Creando la función base...

def frequency_dist(study_var):
    nclasses = round(1 + np.log2(dataset[study_var].count()),0)
    wclasses = np.ceil(((np.max(dataset[study_var]) - np.
 ↪min(dataset[study_var])) / nclasses))

# Creación de los límites superiores e inferiores
```

```
    a = np.zeros([int(nclasses),3], dtype = np.int64)
    b = 0

    for i in range(int(nclasses)):
        for j in range(2):
            a[i][j] = b
            b = b + wclasses
        b = a[i][j]

# Creación de las frecuencias absolutas
    list1 = []
    c = 0
    lower_limit = 0
    upper_limit = wclasses

    for element in range(int(nclasses)):
        for element2 in list(dataset[study_var]):
            if element + 1 != nclasses:
                if element2 >= lower_limit and element2 < upper_limit:
                    c = c + 1
            else:
                if element2 >= lower_limit and element2 <= upper_limit:
                    c = c + 1
        list1.append(c)
        c = 0
        lower_limit = upper_limit
        upper_limit = upper_limit + wclasses

    # Fusión de los límites y frecuencias absolutas
    r1 = 0

    for i2 in range(int(nclasses)):
        a[r1,2] = list1[r1]
        r1 = r1 + 1


# Trasformación y adecuación
    a = pd.DataFrame(a,columns=["Low","High", "Frequency"])

# Frecuencia relativa
    a.loc[:,('Frecuencia_rel')] = a["Frequency"]/sum(a["Frequency"])
# Frecuencia acumulada
    a.loc[:,('Frecuencia_A')] = np.cumsum(a["Frequency"])
# Frecuencia relativa acumulada
    a.loc[:,('Frecuencia_rel_A')] = np.cumsum(a["Frecuencia_rel"])
# Especificación de la variable de análisis
    a.insert(loc = 0,  column = "Variable", value = study_var)
```

```
    return a
```

[38]:
```
frequency_dist("Age")
```

[38]:
```
  Variable  Low  High  Frequency  Frecuencia_rel  Frecuencia_A  \
0      Age    0     8         50        0.070028            50
1      Age    8    16         33        0.046218            83
2      Age   16    24        164        0.229692           247
3      Age   24    32        181        0.253501           428
4      Age   32    40        123        0.172269           551
5      Age   40    48         74        0.103641           625
6      Age   48    56         50        0.070028           675
7      Age   56    64         26        0.036415           701
8      Age   64    72         11        0.015406           712
9      Age   72    80          2        0.002801           714

   Frecuencia_rel_A
0          0.070028
1          0.116246
2          0.345938
3          0.599440
4          0.771709
5          0.875350
6          0.945378
7          0.981793
8          0.997199
9          1.000000
```

[39]:
```
quant_cols
```

[39]:
```
3      Age
4    SibSp
5    Parch
6     Fare
Name: Name, dtype: object
```

[44]:
```
# Modelo de distribuciones de frecuencias consolidado

final_fd = pd.DataFrame({'Variable' : [],
                         'Low' : [],
                         'High' : [],
                        'Frequency' : [],
                         'Frecuencia_rel' : [],
                         'Frecuencia_A' : [],
                         'Frecuencia_rel_A' : []})
```

```
for i in quant_cols:
    x = frequency_dist(i)
    final_fd = final_fd.append(x).reset_index(drop = True)

final_fd
# Exportación de resultados en csv
#final_df.to_csv('Distribución de frecuencias - Titanic.csv')
```

[44]:

	Variable	Low	High	Frequency	Frecuencia_rel	Frecuencia_A
0	Age	0.0	8.0	50.0	0.070028	50.0
1	Age	8.0	16.0	33.0	0.046218	83.0
2	Age	16.0	24.0	164.0	0.229692	247.0
3	Age	24.0	32.0	181.0	0.253501	428.0
4	Age	32.0	40.0	123.0	0.172269	551.0
5	Age	40.0	48.0	74.0	0.103641	625.0
6	Age	48.0	56.0	50.0	0.070028	675.0
7	Age	56.0	64.0	26.0	0.036415	701.0
8	Age	64.0	72.0	11.0	0.015406	712.0
9	Age	72.0	80.0	2.0	0.002801	714.0
10	SibSp	0.0	1.0	608.0	0.682379	608.0
11	SibSp	1.0	2.0	209.0	0.234568	817.0
12	SibSp	2.0	3.0	28.0	0.031425	845.0
13	SibSp	3.0	4.0	16.0	0.017957	861.0
14	SibSp	4.0	5.0	18.0	0.020202	879.0
15	SibSp	5.0	6.0	5.0	0.005612	884.0
16	SibSp	6.0	7.0	0.0	0.000000	884.0
17	SibSp	7.0	8.0	0.0	0.000000	884.0
18	SibSp	8.0	9.0	7.0	0.007856	891.0
19	SibSp	9.0	10.0	0.0	0.000000	891.0
20	SibSp	10.0	11.0	0.0	0.000000	891.0
21	Parch	0.0	1.0	678.0	0.760943	678.0
22	Parch	1.0	2.0	118.0	0.132435	796.0
23	Parch	2.0	3.0	80.0	0.089787	876.0
24	Parch	3.0	4.0	5.0	0.005612	881.0
25	Parch	4.0	5.0	4.0	0.004489	885.0
26	Parch	5.0	6.0	5.0	0.005612	890.0
27	Parch	6.0	7.0	1.0	0.001122	891.0
28	Parch	7.0	8.0	0.0	0.000000	891.0
29	Parch	8.0	9.0	0.0	0.000000	891.0
30	Parch	9.0	10.0	0.0	0.000000	891.0
31	Parch	10.0	11.0	0.0	0.000000	891.0
32	Fare	0.0	47.0	726.0	0.814815	726.0
33	Fare	47.0	94.0	112.0	0.125701	838.0
34	Fare	94.0	141.0	22.0	0.024691	860.0
35	Fare	141.0	188.0	11.0	0.012346	871.0
36	Fare	188.0	235.0	9.0	0.010101	880.0
37	Fare	235.0	282.0	8.0	0.008979	888.0

```
38     Fare  282.0  329.0        0.0        0.000000         888.0
39     Fare  329.0  376.0        0.0        0.000000         888.0
40     Fare  376.0  423.0        0.0        0.000000         888.0
41     Fare  423.0  470.0        0.0        0.000000         888.0
42     Fare  470.0  517.0        3.0        0.003367         891.0

    Frecuencia_rel_A
0           0.070028
1           0.116246
2           0.345938
3           0.599440
4           0.771709
5           0.875350
6           0.945378
7           0.981793
8           0.997199
9           1.000000
10          0.682379
11          0.916947
12          0.948373
13          0.966330
14          0.986532
15          0.992144
16          0.992144
17          0.992144
18          1.000000
19          1.000000
20          1.000000
21          0.760943
22          0.893378
23          0.983165
24          0.988777
25          0.993266
26          0.998878
27          1.000000
28          1.000000
29          1.000000
30          1.000000
31          1.000000
32          0.814815
33          0.940516
34          0.965208
35          0.977553
36          0.987654
37          0.996633
38          0.996633
39          0.996633
```

```
40          0.996633
41          0.996633
42          1.000000
```

0.0.6 Etapa 5: Visualizar los datos

Selección de las gráficas que mejor expliquen a las variables del dataset o cuya información sea importante para responder a la formulación del problema.

0.0.7 Etapa 6: Analizar las posibles interacciones entre las variables del dataset

0.0.8 Etapa 7: Conclusiones

[ ]: