TITANIC

March 9, 2025

1 Titanic dataset analysis

1.0.1 Esta tarea implica la limpieza y el análisis del conjunto de datos del Titanic. El conjunto de datos está disponible en Kaggle y contiene información sobre los pasajeros del Titanic, como su edad, clase, tarifa, etc.

1.0.2 1. Import and clean the dataset

[1]:
```
# Import libraries
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import warnings
from scipy import stats
# Avoid warnings for a clean export
warnings.simplefilter("ignore", category=SyntaxWarning)
warnings.simplefilter("ignore", category=FutureWarning)

from bokeh.resources import CDN
from bokeh.embed import file_html
from IPython.display import display, HTML
```

[2]:
```
import bokeh


titanic_df = pd.read_csv("./assets/Datos Titanic/datoslimpios.csv",␣
 ↪encoding="latin1", on_bad_lines="warn")
titanic_df.head()
```

[2]:

	PassengerId	Survived	Pclass	Name	\
0	1	0	3	Braund	
1	2	1	1	Cumings	
2	3	1	3	Heikkinen	
3	4	1	1	Futrelle	
4	5	0	3	Allen	

```
                                        Lastname     Sex   Age  SibSp  Parch  \
0                              Mr. Owen Harris    male  22.0      1      0
1   Mrs. John Bradley (Florence Briggs Thayer)  female  38.0      1      0
2                                  Miss. Laina  female  26.0      0      0
3          Mrs. Jacques Heath (Lily May Peel)  female  35.0      1      0
4                            Mr. William Henry    male  35.0      0      0

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

[3]:
```
titanic_df.info()

column_names = [element for element in titanic_df.columns]
print(f"Columnas: {column_names}" )
```

```
<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   Lastname     891 non-null    object
 5   Sex          891 non-null    object
 6   Age          891 non-null    float64
 7   SibSp        891 non-null    int64
 8   Parch        891 non-null    int64
 9   Ticket       891 non-null    object
 10  Fare         891 non-null    float64
 11  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
Columnas: ['PassengerId', 'Survived', 'Pclass', 'Name', 'Lastname', 'Sex',
'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Embarked']
```

[4]:
```
numeric_columns = titanic_df.select_dtypes(include=['int64', 'float64']).columns
numeric_df = titanic_df[numeric_columns]
# Remove the PassengerId column
numeric_df = numeric_df.drop(columns=['PassengerId'])
description = numeric_df.describe()
```

[5]:
```
tendencia_central = numeric_df.describe().applymap(lambda x: f"{x:0.3f}")
tendencia_central
```

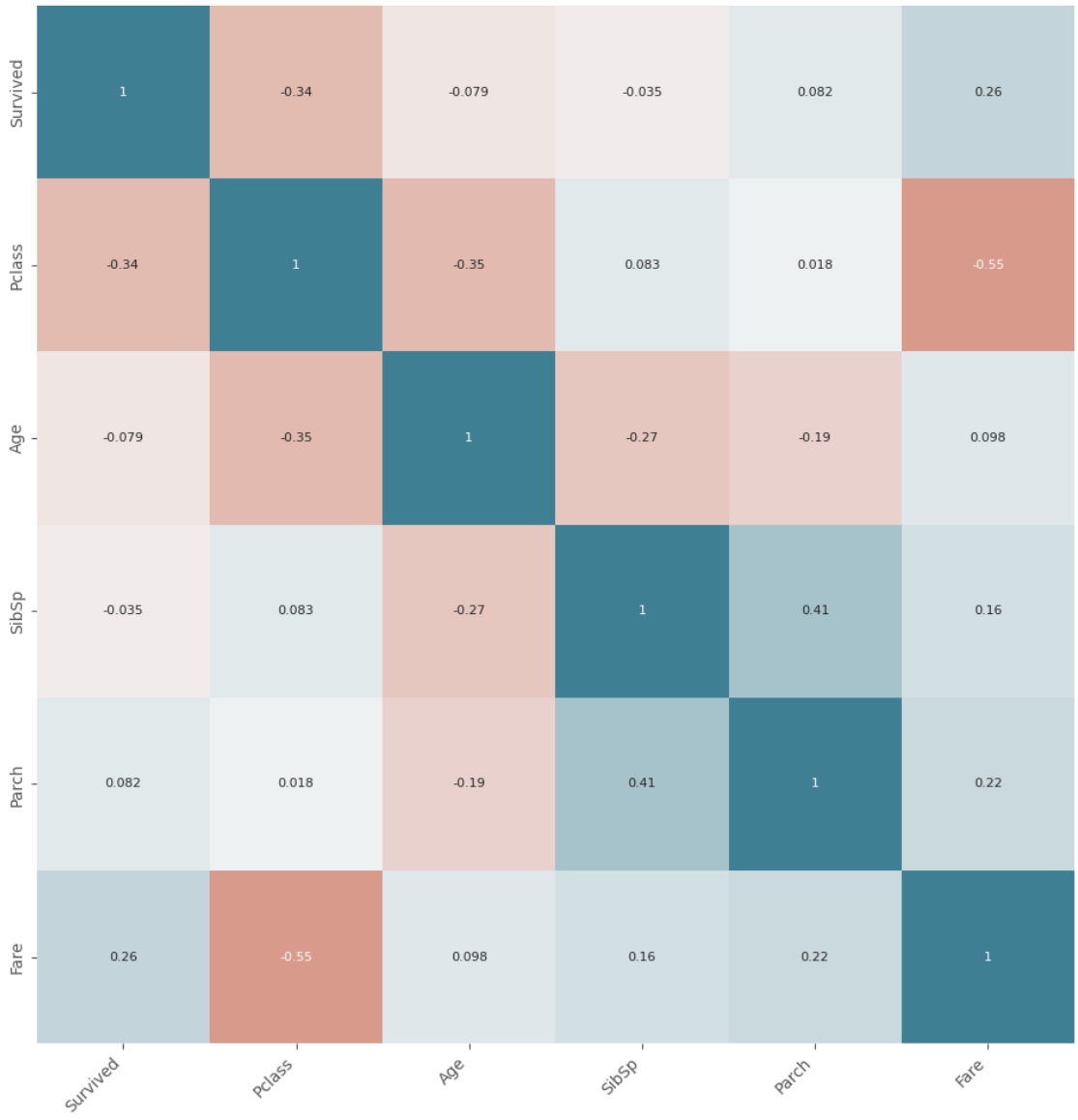
[5]:

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000	891.000	891.000	891.000	891.000	891.000
mean	0.384	2.309	29.385	0.523	0.382	32.204
std	0.487	0.836	13.260	1.103	0.806	49.693
min	0.000	1.000	0.420	0.000	0.000	0.000
25%	0.000	2.000	21.000	0.000	0.000	7.910
50%	0.000	3.000	30.000	0.000	0.000	14.454
75%	1.000	3.000	35.000	1.000	0.000	31.000
max	1.000	3.000	80.000	8.000	6.000	512.329

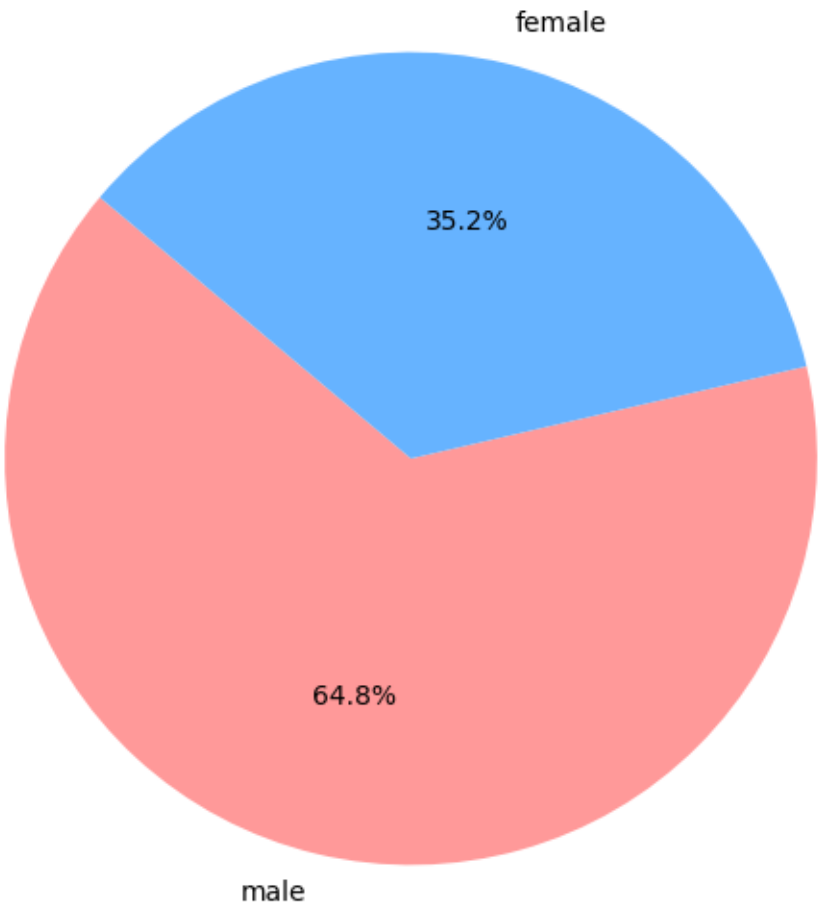
[6]:
```
numeric_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Age       891 non-null    float64
 3   SibSp     891 non-null    int64
 4   Parch     891 non-null    int64
 5   Fare      891 non-null    float64
dtypes: float64(2), int64(4)
memory usage: 41.9 KB
```

[7]:
```
corr_matrix = numeric_df.corr(method='pearson')
# Print corr matrix as a pretty chart of big size
plt.style.use('ggplot')

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10, 10))
sns.heatmap(corr_matrix, annot=True, cbar=False, annot_kws={"size": 8},␣
 ↪vmin=-1, vmax=1, center=0,
            cmap=sns.diverging_palette(20, 220, n=200), square=True, ax=ax)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
ax.tick_params(labelsize=10)

# Adjust layout to center the plot
fig.tight_layout()
plt.show()
```

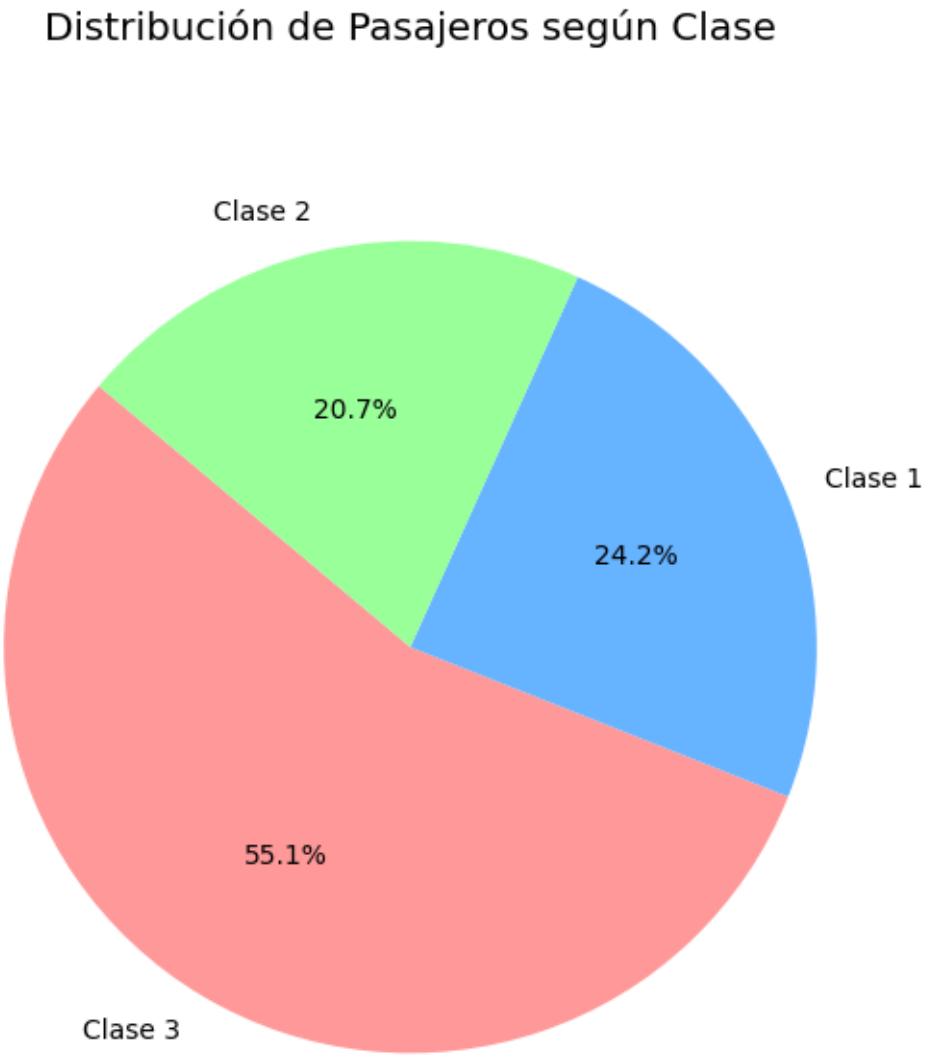


[62]:
```
plt.style.use('ggplot')
plt.figure(figsize=(6, 8))
labels = titanic_df['Sex'].unique()
plt.pie(titanic_df['Sex'].value_counts(), labels =titanic_df['Sex'].
 ↪value_counts().index,
        autopct='%1.1f%%', startangle=140,␣
 ↪colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0'])
plt.axis('equal')
plt.title("Distribución de pasajeros según género")
plt.show()
```



[66]:
```
# Imprime un pie plot de la columna Pclass
# El Pie chart debe mostrar el valor porcentual y el numero de pasajeros por␣
 ↪clase


plt.style.use('ggplot')
plt.figure(figsize=(6, 8))
labels = titanic_df['Pclass'].unique()
pasajeros = titanic_df['Pclass'].value_counts()
```

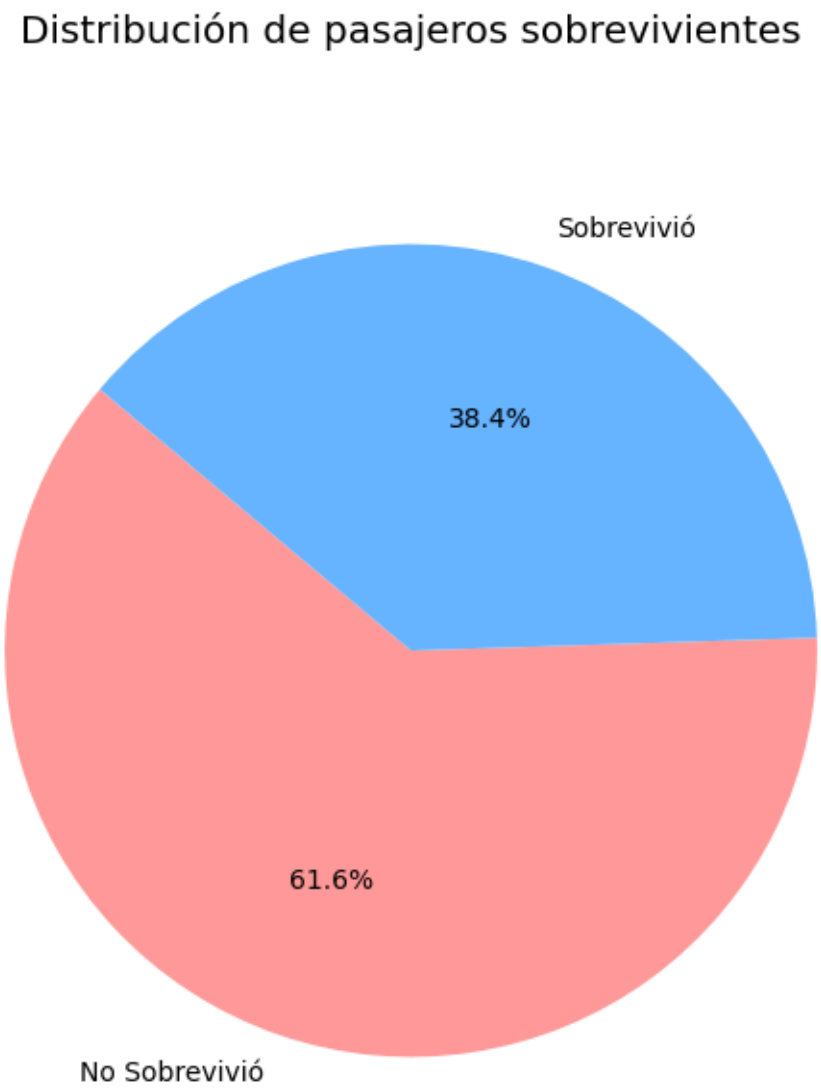
```
labels = [f"Clase {label}" for label in labels]
plt.pie(titanic_df['Pclass'].value_counts(), labels=labels,
        autopct='%1.1f%%', startangle=140,␣
 ↪colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0'])
plt.axis('equal')
plt.title("Distribución de Pasajeros según Clase")
plt.show()
#
```



Dado que la clase, a pesar de estar determinada con un valor númerico, se trata de una variable categórica, carece de sentido analizarla por su distribución estadística. Una mejor forma de representar esa información puede ser con gráficos especializados en mostrar variables categóricas.

Como podemos observar, la gran mayoría de pasajeros se encontraban en **tercera clase**.

[68]:
```
# Imprime un pie plot de la columna Survived
# El Pie chart debe mostrar el valor porcentual y el numero de pasajeros por␣
 ↪clase


plt.style.use('ggplot')
plt.figure(figsize=(6, 8))
labels = titanic_df['Survived'].unique()
labels = [f"Sobrevivió" if label == 1 else "No Sobrevivió" for label in labels]
plt.pie(titanic_df['Survived'].value_counts(), labels=labels,
        autopct='%1.1f%%', startangle=140,␣
 ↪colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0'])
plt.axis('equal')
plt.title("Distribución de pasajeros sobrevivientes")
plt.show()
```



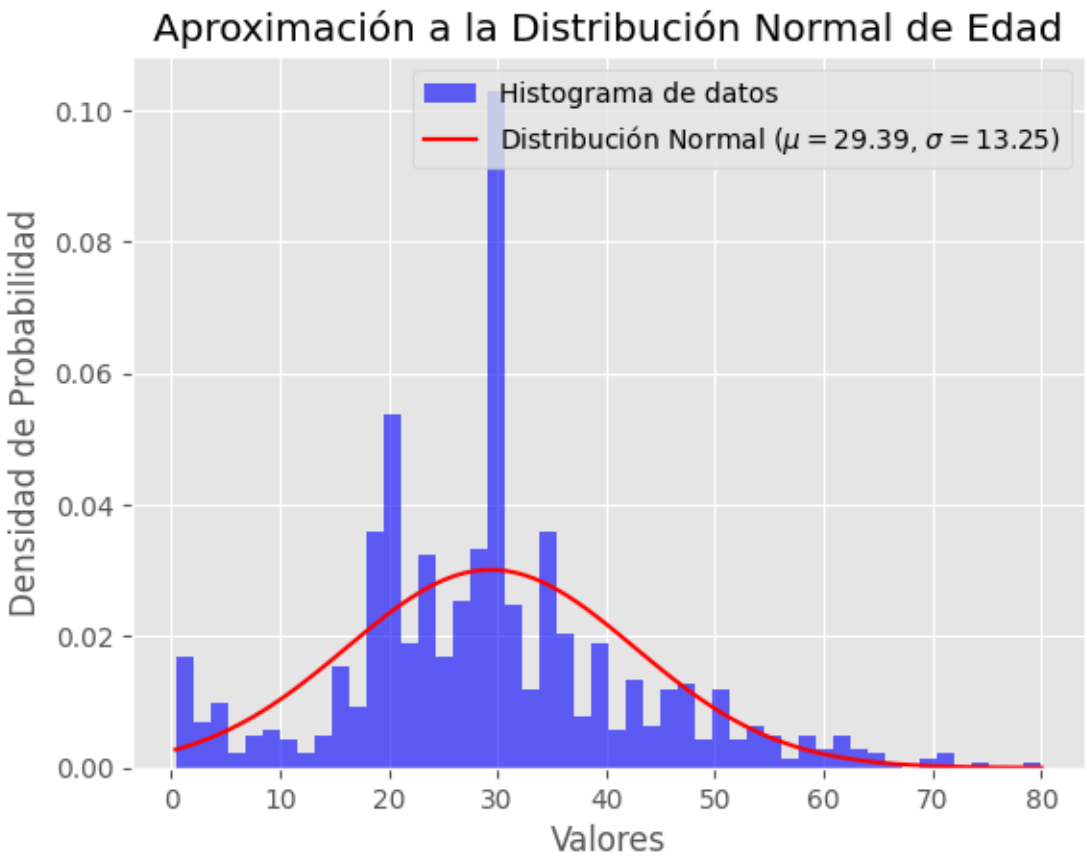
[69]:
```
numeric_df = numeric_df.drop(columns=['Survived'])
numeric_df = numeric_df.drop(columns=['Pclass'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[69], line 1
----> 1 numeric_df = numeric_df.drop(columns=[          ])
      2 numeric_df = numeric_df.drop(columns=['Pclass'])
```

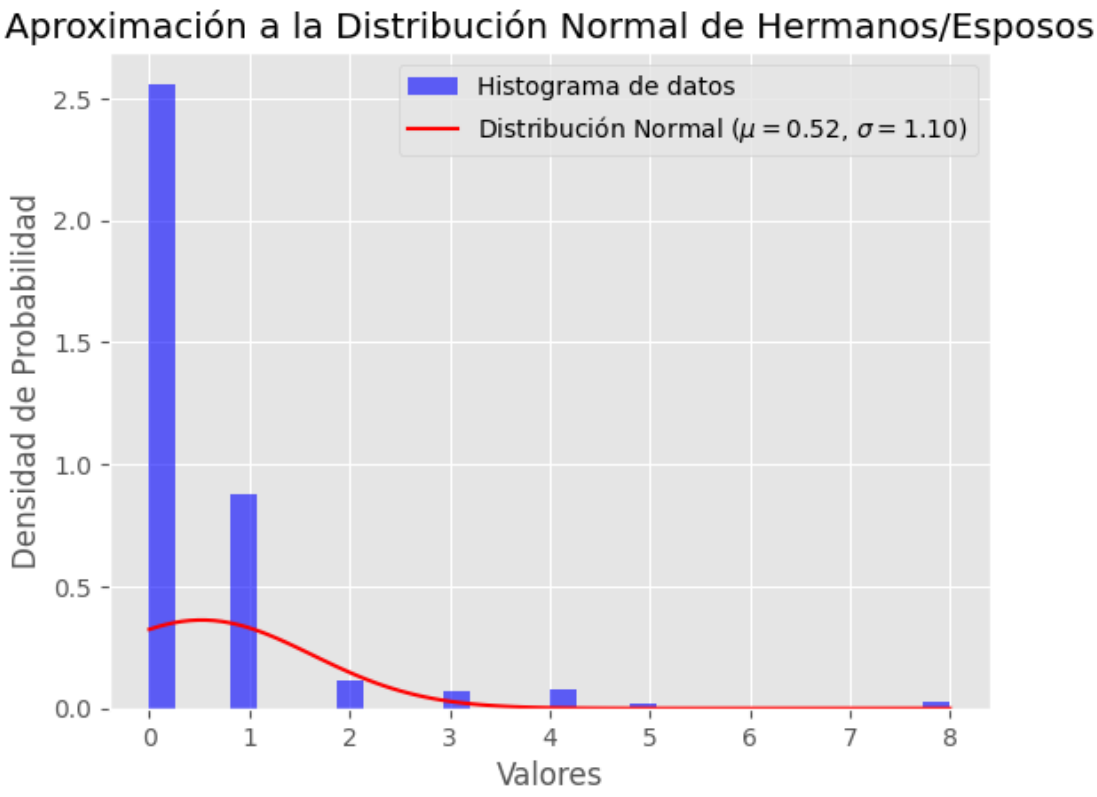
```
File ~/Documents/Master/BigData/covid-da-project/.venv/lib/python3.12/
 ↪site-packages/pandas/core/frame.py:5581, in DataFrame.drop(self, labels, axis␣
 ↪index, columns, level, inplace, errors)
   5433 def drop(
   5434     self,
   5435     labels: IndexLabel | None = None,
   (…)   5442     errors: IgnoreRaise = "raise",
   5443 ) -> DataFrame | None:
   5444     """
   5445     Drop specified labels from rows or columns.
   5446
   (…)   5579             weight  1.0     0.8
   5580     """
-> 5581     return super().drop(
   5582         labels=labels,
   5583         axis=axis,
   5584         index=index,
   5585         columns=columns,
   5586         level=level,
   5587         inplace=inplace,
   5588         errors=errors,
   5589     )

File ~/Documents/Master/BigData/covid-da-project/.venv/lib/python3.12/
 ↪site-packages/pandas/core/generic.py:4788, in NDFrame.drop(self, labels, axis␣
 ↪index, columns, level, inplace, errors)
   4786 for axis, labels in axes.items():
   4787     if labels is not None:
-> 4788         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4790 if inplace:
   4791     self._update_inplace(obj)

File ~/Documents/Master/BigData/covid-da-project/.venv/lib/python3.12/
 ↪site-packages/pandas/core/generic.py:4830, in NDFrame._drop_axis(self, labels␣
 ↪axis, level, errors, only_slice)
   4828         new_axis = axis.drop(labels, level=level, errors=errors)
   4829     else:
-> 4830         new_axis = axis.drop(labels, errors=errors)
   4831     indexer = axis.get_indexer(new_axis)
   4833 # Case for non-unique axis
   4834 else:

File ~/Documents/Master/BigData/covid-da-project/.venv/lib/python3.12/
 ↪site-packages/pandas/core/indexes/base.py:7070, in Index.drop(self, labels,␣
 ↪errors)
   7068 if mask.any():
   7069     if errors != "ignore":
-> 7070         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7071     indexer = indexer[~mask]
```

```
    7072 return self.delete(indexer)

KeyError: "['Survived'] not found in axis"
```

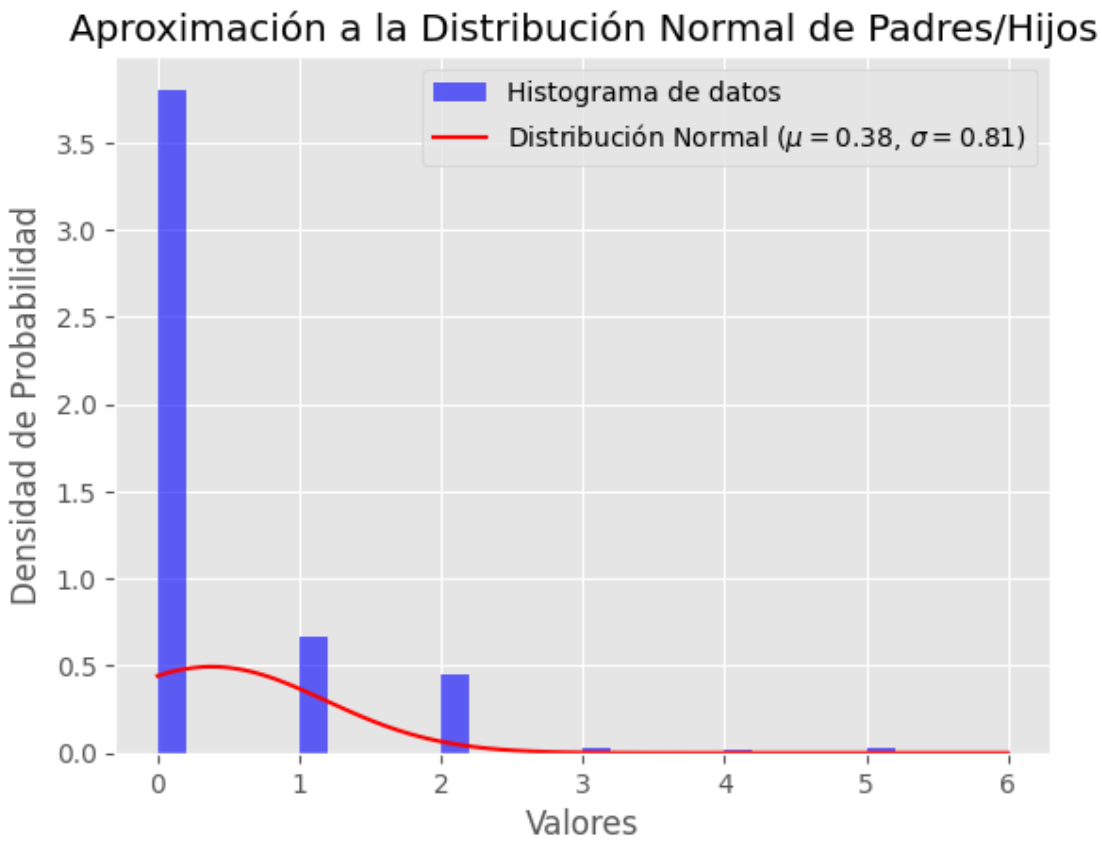
[70]:
```
# Estimar parámetros de la distribución normal
mu, sigma = np.mean(numeric_df['Age']), np.std(numeric_df['Age'])

# Crear el rango de valores para la curva
x = np.linspace(min(numeric_df['Age']), max(numeric_df['Age']), 100)

y = stats.norm.pdf(x, mu, sigma)
plt.style.use('ggplot')
# Graficar el histograma y la curva de densidad
plt.hist(numeric_df['Age'], bins=50, density=True, alpha=0.6, color='b',␣
 ↪label='Histograma de datos')
plt.plot(x, y, 'r', label=f'Distribución Normal ($\mu={mu:.2f}$, $\sigma={sigma:
 ↪.2f}$)')
plt.xlabel('Valores')
plt.ylabel('Densidad de Probabilidad')
plt.title('Aproximación a la Distribución Normal de Edad')
plt.legend()
plt.show()
```



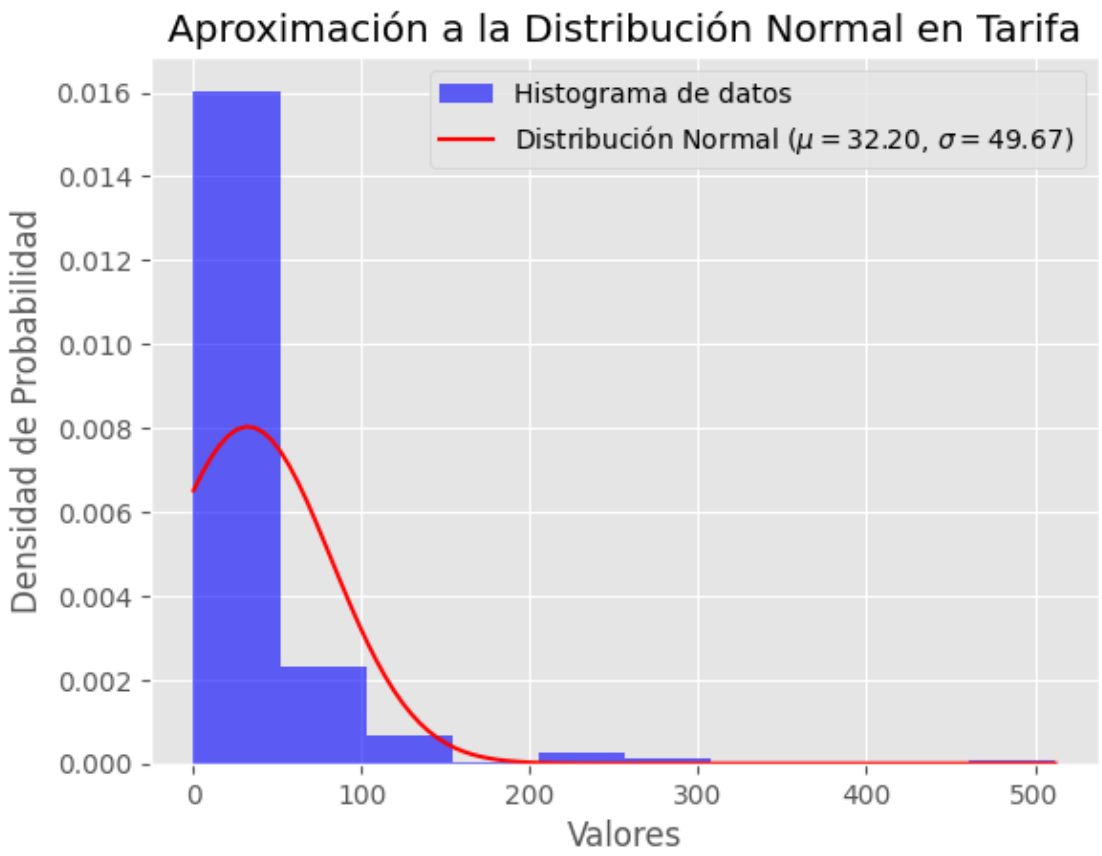
[71]:
```
# Estimar parámetros de la distribución normal
mu, sigma = np.mean(numeric_df['SibSp']), np.std(numeric_df['SibSp'])

# Crear el rango de valores para la curva
x = np.linspace(min(numeric_df['SibSp']), max(numeric_df['SibSp']), 100)

y = stats.norm.pdf(x, mu, sigma)

# Graficar el histograma y la curva de densidad
plt.hist(numeric_df['SibSp'], bins=30, density=True, alpha=0.6, color='b',␣
 ↪label='Histograma de datos')
plt.plot(x, y, 'r', label=f'Distribución Normal ($\mu={mu:.2f}$, $\sigma={sigma:
 ↪.2f}$)')
plt.xlabel('Valores')
plt.ylabel('Densidad de Probabilidad')
plt.title('Aproximación a la Distribución Normal de Hermanos/Esposos')
plt.legend()
plt.show()
```



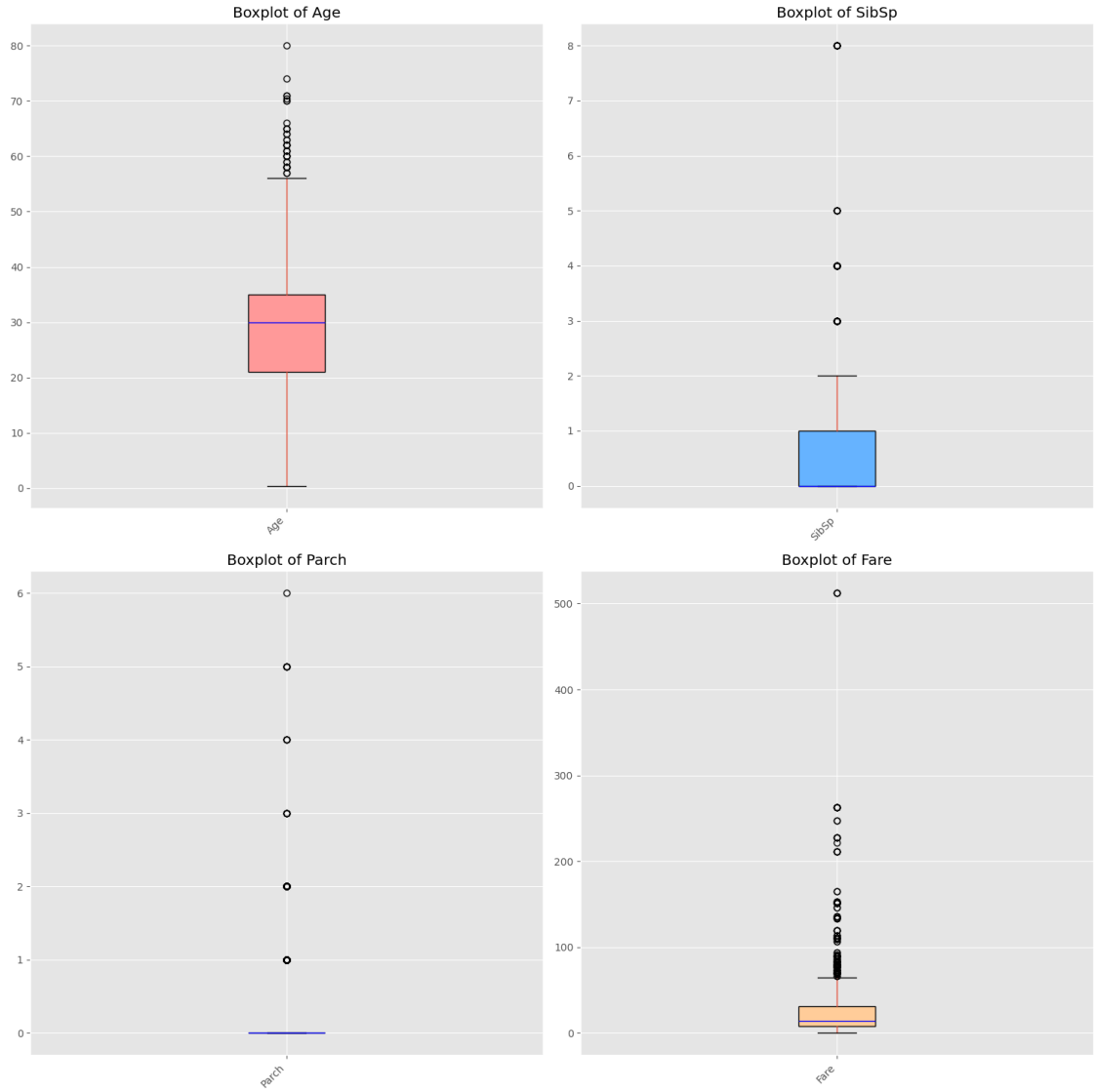
[72]:
```
# Estimar parámetros de la distribución normal
mu, sigma = np.mean(numeric_df['Parch']), np.std(numeric_df['Parch'])

# Crear el rango de valores para la curva
x = np.linspace(min(numeric_df['Parch']), max(numeric_df['Parch']), 100)

y = stats.norm.pdf(x, mu, sigma)

# Graficar el histograma y la curva de densidad
plt.hist(numeric_df['Parch'], bins=30, density=True, alpha=0.6, color='b',␣
 ↪label='Histograma de datos')
plt.plot(x, y, 'r', label=f'Distribución Normal ($\mu={mu:.2f}$, $\sigma={sigma:
 ↪.2f}$)')
plt.xlabel('Valores')
plt.ylabel('Densidad de Probabilidad')
plt.title('Aproximación a la Distribución Normal de Padres/Hijos')
plt.legend()
plt.show()
```



[74]:
```
# Estimar parámetros de la distribución normal
mu, sigma = np.mean(numeric_df['Fare']), np.std(numeric_df['Fare'])

# Crear el rango de valores para la curva
x = np.linspace(min(numeric_df['Fare']), max(numeric_df['Fare']), 100)

y = stats.norm.pdf(x, mu, sigma)

# Graficar el histograma y la curva de densidad
plt.hist(numeric_df['Fare'], bins=10, density=True, alpha=0.6, color='b',␣
 ↪label='Histograma de datos')
plt.plot(x, y, 'r', label=f'Distribución Normal ($\mu={mu:.2f}$, $\sigma={sigma:
 ↪.2f}$)')
plt.xlabel('Valores')
plt.ylabel('Densidad de Probabilidad')
plt.title('Aproximación a la Distribución Normal en Tarifa')
plt.legend()
plt.show()
```



[17]:
```
plt.style.use('ggplot')
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 15))
colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0']
# Flatten the axes array for easy iteration
axes = axes.flatten()

# Generate a boxplot for each column in the dataframe
for i, column in enumerate(numeric_df.columns):
    numeric_df.boxplot(column=column, ax=axes[i], patch_artist=True,
                       boxprops=dict(facecolor=colors[i], color='black'),
                       medianprops=dict(color='blue'))
    axes[i].set_title(f'Boxplot of {column}')
    axes[i].tick_params(labelsize=10)
    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')

# Adjust layout to prevent overlap
fig.tight_layout()
plt.show()
```



1.1 Equipo:

- Coconi Dafne
- Cortés López
- Sánchez Erik
- Villegas Getsemaní

Ejemplo de grafico interactivo con plotly

[ ]:
```
import plotly.graph_objects as go
from IPython.display import display, HTML

import plotly
plotly.offline.init_notebook_mode()
```

```
display(HTML(
    '<script type="text/javascript" async src="https://cdnjs.cloudflare.com/
 ↪ajax/libs/mathjax/2.7.1/MathJax.js?config=TeX-MML-AM_SVG"></script>'
))
# Estimar parámetros de la distribución normal
mu, sigma = np.mean(numeric_df['Fare']), np.std(numeric_df['Fare'])

# Crear el rango de valores para la curva
x = np.linspace(min(numeric_df['Fare']), max(numeric_df['Fare']), 100)
y = stats.norm.pdf(x, mu, sigma)

# Crear el histograma y la curva de densidad usando plotly
fig = go.Figure()

# Agregar el histograma
fig.add_trace(go.Histogram(
    x=numeric_df['Fare'],
    nbinsx=60,
    histnorm='probability density',
    name='Histograma de datos',
    marker_color='blue',
    opacity=0.6
))

# Agregar la curva de densidad
fig.add_trace(go.Scatter(
    x=x,
    y=y,
    mode='lines',
    name=r'Distribución Normal ($\mu= {0:.2f},\sigma={1:.2f}$)'.format(mu,␣
 ↪sigma),
    line=dict(color='red')
))

# Actualizar el layout para mejorar la visualización
fig.update_layout(
    title='Aproximación a la Distribución Normal en Tarifa',
    xaxis_title='Valores',
    yaxis_title='Densidad de Probabilidad',
    legend=dict(x=0.7, y=0.95),
    template='plotly_white'
)

fig.show()
```

```
<IPython.core.display.HTML object>
```

[77]:
```
import plotly.graph_objects as go
from IPython.display import display, HTML

# Estimar parámetros de la distribución normal
mu, sigma = np.mean(numeric_df['Age']), np.std(numeric_df['Age'])

# Crear el rango de valores para la curva
x = np.linspace(min(numeric_df['Age']), max(numeric_df['Age']), 100)
y = stats.norm.pdf(x, mu, sigma)

# Crear el histograma y la curva de densidad usando plotly
fig = go.Figure()

# Agregar el histograma
fig.add_trace(go.Histogram(
    x=numeric_df['Age'],
    nbinsx=60,
    histnorm='probability density',
    name='Histograma de datos',
    marker_color='blue',
    opacity=0.6
))

# Agregar la curva de densidad
fig.add_trace(go.Scatter(
    x=x,
    y=y,
    mode='lines',
    name=r'Distribución Normal ($\mu= {0:.2f},\sigma={1:.2f}$)'.format(mu,␣
 ↪sigma),
    line=dict(color='red')
))

# Actualizar el layout para mejorar la visualización
fig.update_layout(
    title='Aproximación a la Distribución Normal en Edad',
    xaxis_title='Valores',
    yaxis_title='Densidad de Probabilidad',
    legend=dict(x=0.7, y=0.95),
    template='plotly_white'
)

fig.show()
```