Exploratory Data Analysis - Titanic Dataset

Brief Introduction:

```
The 'Titanic' is a popular dataset for data analysis and machine
learning. This dataset consists of inofrmation about the
passengers onboard the RMS Titanic, which was shiwrecked
unfortuantely, and can be used to predict whether a passenger
survived the disaster or not. The features like age, gender,
fare, cabin, survival_status, etc. are used to perform
Exploratory Data Analysis (EDA) in this project.
```

About the Data:

```
The dataset has the following features,
```

0. PassengerID: Unique ID of Passengers
1. Survived: Survival Status (Yes = 1, No = 0)
2. Pclass: Passenger Classes (1 = First Class, 2 = Second Class, 3 = Third Class)
3. Sex: Passengers Gender
4. Age: Passengers Age
5. SibSp: Number of Siblings/ or Spouses onboard,
6. Parch: Number of Parents/ Child onboard,
7. Fare: Fare paid for the ticket
8. Embarked: Part of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)
9. Name: Name of the Passengers
10. Cabin: Cabin Number
11. Ticket: Ticket Number

Importing Libraries:

In [ ]:
```
import numpy as np
import pandas as pd
%matplotlib inline
from matplotlib import pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
sns.set_style('whitegrid')
```

Loading the Dataset:

In [ ]:
```
df = pd.read_csv('Titanic-Dataset.csv')
```

Exploring the Dataset:

1. Exploring the dataset:

In [ ]:
```
df.head(5)
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0

In [ ]:
```
df.tail(5)
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

So, this is how are data looks like from the start and end.

2. Checking the dimension of the dataset:

In [ ]:
```
df.shape
```

Out[ ]:
```
(891, 12)
```

Observation:

1. We can see this dataset has 12 features available. We have previously described about these features.
2. We have, both, Numerical and Categorical data present in the Dataset.

3. Fetching the info about the dataset:

In [ ]:
```
df.info
```

Out[ ]:
```
<bound method DataFrame.info of      PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         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
..                                                ...     ...   ...    ...
886                            Montvila, Rev. Juozas    male  27.0      0
887                     Graham, Miss. Margaret Edith  female  19.0      0
888         Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                            Behr, Mr. Karl Howell    male  26.0      0
890                              Dooley, Mr. Patrick    male  32.0      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
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]>
```

In [ ]:
```
df.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          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
```

Observations:

1. The Shape of the dataset is (891, 12) which indicates our dataset has 891 number of rows (/records) and 12 columns (/features). Each of the rows shows information about a single passenger. So, totally we have a dataset of 891 passengers in this case.

2. From the output gained, we have columns consist of the name of each column, along with two extra information namely, Non-null Count, which indicates how many non-null values are there in the dataset, and Dtype, which represents what type of value that particular column has.

   ```
   [  i. int64 means int value,
     ii. float64 means float value,
    iii. object means string value.]
   ```

3. In the 'age' column we can see that out of 891 values, we have 714 non-null values which implies the fact that we have (891 - 714) = 177 Null values present in that particular column.

4. Similarly, in the 'Cabin' feature, out of 891 values, we have only 204 non-null vlaues which implies we have (891 - 204) = 687 Null values. But, this poses a challenge espcially because the Null values present in this particualar column is huge. Only 23% of the data is present [(204/891)*100 = 23% (approx, 22.89% to be exact)], while 77% of the data is missing [(687/891)*100 = 77% (approx, 77.10% to be exact)]. So, we need to drop this particular feature.

5. In the 'Embarked' Column, we see 889 Non-null values, which leaves us only 2 Null (/missing) values. This is a minor issue that can be dealt with easily.

6. Except for the 'Age', 'Embarked', and 'Cabin' feature, we do not have any null values in other columns.

In [ ]:
```
df.nunique()
```

Out[ ]:
```
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

Observation:

Self-explanatory.

4. Getting the Summary Statistics:

In [ ]:
```
a = round(df.describe().T, 4)
a
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
PassengerId	891.0	446.0000	257.3538	1.00	223.5000	446.0000	668.5	891.0000
Survived	891.0	0.3838	0.4866	0.00	0.0000	0.0000	1.0	1.0000
Pclass	891.0	2.3086	0.8361	1.00	2.0000	3.0000	3.0	3.0000
Age	714.0	29.6991	14.5265	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.5230	1.1027	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.3816	0.8061	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.2042	49.6934	0.00	7.9104	14.4542	31.0	512.3292

Observations:

1. Some of the features which are unimportant, like 'PassengerID', 'Survived', etc. are ignored. Features, such as, 'Age' and 'Fare' are only considered.
2. **The Age Feature:**

   A. It seems that the count of 714 means we have age value of 714 people, while the rest are missing, as can be see from the above.

   B. So, we have a mean value of 29.69 that indicates the average age of all the passengers, irrespective of the age-group they belonged to, is 29.69 years.

   C. Also, the std (Standard Deviation) value as 14.52 says that the most of the people have their age-range between (29.96 - 14.52) to (29.63 + 14.52), as we already know that for a Continuous Randome Variable, most ot the values can be indentified in the range on (mean_value - std_dev) to (mean_value + std_dev).

D. The min age is 0.42 which shows that out of all the passengers we have a 0.4 years old person as a minimum age of any passengers.

E. The max age of 80 shows out of all the passengers we have the highest aged person of 80 years.

F. Now, the 25% value (25th percentile) as 20.12 years says that 25% percent of the passengers have an age of less than 20.12 years.

G. Similarly, we have 50% (50th percentile) as 28 which means 50% of the passengers have less than 28 years.

H. Lastly, 75% (75th percentile) as 38 tells us that 75% of people have age less than 38 years.

3. **The Fare Feature:** The 'Fare' feature can also be summarised in a similar fashion like we have done with the 'Age' feature, from the output gained.

EDA:

A. Data PreProcessing:

```
Following are the steps that will be covered,
            1. Missing Value Handle,
            2. Feature Coversion,
            3. Visualizatinon.
```

1. Missing Value Handle:

In [ ]:
```
df.isnull().sum()
```

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

In [ ]:
```
a1 = round(df.isnull().sum()*100 / len(df), 2)
a1
```

Out[ ]:
```
PassengerId     0.00
Survived        0.00
Pclass          0.00
Name            0.00
Sex             0.00
Age            19.87
SibSp           0.00
Parch           0.00
Ticket          0.00
Fare            0.00
Cabin          77.10
Embarked        0.22
dtype: float64
```

Observations:

The features such as, 'Age', 'Cabin', and 'Embarked' have Null Values.

i. Cabin Feature:

It has 687 rows of missing records. Though it sounds plaussible to delete the whole feature from the dataset to make a more compelling dataset, stil it would much interesting to convert it into something else. A cabin number looks like 'C123' where 'C' is the Deck (floor) number and '123' refers to a particular section of that deck. So, we will first extract the variables and make a new feature out of it, named as 'Deck' that represents the deck of the cabin, and then we will convert the feature into a numberic variables. The missing values will be converted to Zero.

In [ ]:
```
df = df_new
```

In [ ]:
```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [df_new]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").s
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)
# we can now drop the cabin feature
# df = df.drop(['Cabin'], axis=1)
df_new = df_new.drop(['Cabin'], axis=1)

# It's showing error because the 'Cabin' feature has already been dropped.
# If you want to see it, then make the dataset load again.
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:\Users\Pritam - PC\AppData\Local\Programs\Python\Python312\Lib\site-packag
es\pandas\core\indexes\base.py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas\\_libs\\hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.Py
ObjectHashTable.get_item()

File pandas\\_libs\\hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.Py
ObjectHashTable.get_item()

KeyError: 'Cabin'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[67], line 6
      3 data = [df_new]
      5 for dataset in data:
----> 6     dataset['Cabin'] = dataset['Cabin'].fillna("U0")
      7     dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-
Z]+)").search(x).group())
      8     dataset['Deck'] = dataset['Deck'].map(deck)

File c:\Users\Pritam - PC\AppData\Local\Programs\Python\Python312\Lib\site-packag
es\pandas\core\frame.py:4090, in DataFrame.__getitem__(self, key)
   4088 if self.columns.nlevels > 1:
   4089     return self._getitem_multilevel(key)
-> 4090 indexer = self.columns.get_loc(key)
   4091 if is_integer(indexer):
   4092     indexer = [indexer]

File c:\Users\Pritam - PC\AppData\Local\Programs\Python\Python312\Lib\site-packag
es\pandas\core\indexes\base.py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'Cabin'
```

ii. age feature:

As we have seen, the 'Age' feature contains 177 missing values.
An easy way of imputing new values into the missing records is

to use <.replace()> function to fill in with the mean value. but, this will impose a problem beacuse, if say the mean value is 29.0, this will show a child's age as 29 years, which is impossible.

So, what we will do here is we are going to normalize the feature by creating Array having random values, which are computed on the basis of Mean Value regarding the std_dev and isnull.

In [ ]:
```
data = [df]

for dataset in data:
    mean = df["Age"].mean()
    std = df["Age"].std()
    is_null = dataset["Age"].isnull().sum()
    # compute random numbers between the mean, std and is_null
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)
    # fill NaN values in Age column with random values generated
    age_slice = dataset["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    dataset["Age"] = age_slice
    dataset["Age"] = df["Age"].astype(int)
    dataset["Age"] = df_new["Age"].astype(int)

df["Age"].isnull().sum()
df_new['Age'].isnull().sum()
```

Out[ ]:
```
0
```

iii. Embarked Feature:

Since the 'Embarked' feature contains only 2 missing values, we will be filling these two missing values with the most common value.

In [ ]:
```
common_value = 'S'                  # S = Southampton
data = [df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].fillna(common_value)
    df_new = dataset['Embarked'].fillna(common_value)
```

2. Dropping unnecessary feature:

In [ ]:
```
df_new = df.drop(['PassengerId'], axis=1)
```

Out[ ]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emba
0	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	
2	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	
4	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	
...	...	...	...	...	...	...	...	...	...	
886	0	2	Montvila, Rev. Juozas	male	27	0	0	211536	13.0000	
887	1	1	Graham, Miss. Margaret Edith	female	19	0	0	112053	30.0000	
888	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29	1	2	W./C. 6607	23.4500	
889	1	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30.0000	
890	0	3	Dooley, Mr. Patrick	male	32	0	0	370376	7.7500	

891 rows × 11 columns

3. Feature Conversion:

Here, we will deal with five features. 'Fare', 'Name', 'Sex', 'Ticket', 'Embarked'

In [ ]:
```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    object
 3   Sex       891 non-null    object
 4   Age       891 non-null    int32
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    object
 8   Fare      891 non-null    float64
 9   Embarked  891 non-null    object
 10  Deck      891 non-null    int32
dtypes: float64(1), int32(2), int64(4), object(4)
memory usage: 69.7+ KB
```

In [ ]:
```
df.shape
```

Out[ ]: (891, 12)

In [ ]:
```
df_new.shape

# The PassengerID feature has been dropped.
```

Out[ ]: (891, 11)

i. Fare Feature:

We will covert the data type fro float64 to int64 by using the <.astype()> function.

In [ ]:
```
data = [df_new]

for dataset in data:
    dataset['Fare'] = dataset['Fare'].fillna(0)
    dataset['Fare'] = dataset['Fare'].astype(int)
```

ii. Name Feature:

We will use the name feture to extract the Titles in order to build a new feature.

In [ ]:
```
data = [df_new]
titles = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

for dataset in data:
    # extract titles
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)
    # replace titles with a more common title or as Rare
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col
                                                 'Major', 'Rev', 'Sir', 'Jonkheer', '
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
```

```
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
    # convert titles into numbers
    dataset['Title'] = dataset['Title'].map(titles)
    # filling NaN with 0, to get safe
    dataset['Title'] = dataset['Title'].fillna(0)
df_new = df_new.drop(['Name'], axis=1)
```

iii. Sex Feature:

We will the convert this feature into Numeric.

In [ ]:
```
# For some reason, df_new['Sex'] was NaN, So I had to replace them with df['Sex'
a = [df_new]

for data in a:
    data['Sex'] = df['Sex']

df_new['Sex']
```

Out[ ]:
```
0        male
1      female
2      female
3      female
4        male
        ...
886      male
887    female
888    female
889      male
890      male
Name: Sex, Length: 891, dtype: object
```

In [ ]:
```
genders = {"male": 0, "female": 1}
data = [df_new]

for dataset in data:
    dataset['Sex'] = dataset['Sex'].map(genders)
```

In [ ]:
```
df_new['Sex']
```

Out[ ]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    1
889    0
890    0
Name: Sex, Length: 891, dtype: int64
```

iv. Ticket:

In [ ]:
```
df_new['Ticket'].describe()
```

Out[ ]:
```
count        891
unique       681
top       347082
freq           7
Name: Ticket, dtype: object
```

Since this feature has 681 numbers of unique values, it will be better if we simply drop this.

In [ ]:
```
df_new = df_new.drop(['Ticket'], axis=1)
```

v. Embarked Feature:

Just like the 'Sex' feature, we will convert it into Numeric.

In [ ]:
```
ports = {"S": 0, "C": 1, "Q": 2}
data = [df_new]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

In [ ]:
```
df_new.info()

# Data type of 'Fare' has been changed. Also, 'Sex' and 'Embarked' has changed d
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    int32
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    int32
 7   Embarked  891 non-null    int64
 8   Deck      891 non-null    int32
 9   Title     891 non-null    int64
dtypes: int32(3), int64(7)
memory usage: 59.3 KB
```

In [ ]:
```
df_new.shape

# Since the 'Ticket' Feature has been dropped.
```

Out[ ]:
```
(891, 10)
```

In [ ]:
```
df_new.head(20).T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Survived	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	1
Pclass	3	1	3	1	3	3	1	3	3	2	3	1	3	3	3	2	3	2
Sex	0	1	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0
Age	22	38	26	35	35	40	54	2	27	14	4	58	20	39	14	55	2	35
SibSp	1	1	0	1	0	0	0	3	0	1	1	0	0	1	0	0	4	0
Parch	0	0	0	0	0	0	0	1	2	0	1	0	0	5	0	0	1	0
Fare	7	71	7	53	8	8	51	21	11	30	16	26	8	31	7	16	29	13
Embarked	0	1	0	0	0	2	0	0	0	1	0	0	0	0	0	0	2	0
Deck	8	3	8	3	8	8	5	8	8	8	7	3	8	8	8	8	8	8
Title	1	3	2	3	1	1	1	4	3	3	2	2	1	1	2	3	4	1

Feature Creation:

5. Visualization:

For Visualization, we can take one of the two routes for most of the graphs.
1. Building separate graphs as ususal,
2. Make a function designed specifically for graph/ chart selection based on different criterias.
(Basically, here we shall make a function which will automatically select the graph appropriate for
a given scenario. This will reduce our time and effort due to recursion by availing the facilities
provided by Function.)

We'll go for the Usual Method.

In [ ]:
```
df_new.head(20).T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Survived	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	1
Pclass	3	1	3	1	3	3	1	3	3	2	3	1	3	3	3	2	3	2
Sex	0	1	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0
Age	22	38	26	35	35	40	54	2	27	14	4	58	20	39	14	55	2	35
SibSp	1	1	0	1	0	0	0	3	0	1	1	0	0	1	0	0	4	0
Parch	0	0	0	0	0	0	0	1	2	0	1	0	0	5	0	0	1	0
Fare	7	71	7	53	8	8	51	21	11	30	16	26	8	31	7	16	29	13
Embarked	0	1	0	0	0	2	0	0	0	1	0	0	0	0	0	0	2	0
Deck	8	3	8	3	8	8	5	8	8	8	7	3	8	8	8	8	8	8
Title	1	3	2	3	1	1	1	4	3	3	2	2	1	1	2	3	4	1

In [ ]:
```
df_new.tail(20).T
```

Out[ ]:

	871	872	873	874	875	876	877	878	879	880	881	882	883	884
Survived	1	0	0	1	1	0	0	0	1	1	0	0	0	0
Pclass	1	1	3	2	3	3	3	3	1	2	3	3	2	3
Sex	1	0	0	1	1	0	0	0	1	1	0	1	0	0
Age	47	33	47	28	15	20	19	36	56	25	33	22	28	25
SibSp	1	0	0	1	0	0	0	0	0	0	0	0	0	0
Parch	1	0	0	0	0	0	0	0	1	1	0	0	0	0
Fare	52	5	9	24	7	9	7	7	83	26	7	10	10	7
Embarked	0	0	0	1	1	0	0	0	1	0	0	0	0	0
Deck	4	2	8	8	8	8	8	8	3	8	8	8	8	8
Title	3	1	1	3	2	1	1	1	3	3	1	2	1	1

In [ ]:
```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    int32
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    int32
 7   Embarked  891 non-null    int64
 8   Deck      891 non-null    int32
 9   Title     891 non-null    int64
dtypes: int32(3), int64(7)
memory usage: 59.3 KB
```

In [ ]:
```
a = round(df_new.describe(), 2)
a.T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Survived	891.0	0.38	0.49	0.0	0.0	0.0	1.0	1.0
Pclass	891.0	2.31	0.84	1.0	2.0	3.0	3.0	3.0
Sex	891.0	0.35	0.48	0.0	0.0	0.0	1.0	1.0
Age	891.0	29.72	13.54	0.0	21.0	29.0	38.0	80.0
SibSp	891.0	0.52	1.10	0.0	0.0	0.0	1.0	8.0
Parch	891.0	0.38	0.81	0.0	0.0	0.0	0.0	6.0
Fare	891.0	31.79	49.70	0.0	7.0	14.0	31.0	512.0
Embarked	891.0	0.36	0.64	0.0	0.0	0.0	1.0	2.0
Deck	891.0	6.94	2.07	0.0	8.0	8.0	8.0	8.0
Title	891.0	1.73	1.03	1.0	1.0	1.0	2.0	5.0

In [ ]:
```
df_new.shape
```

Out[ ]:
```
(891, 10)
```

In [ ]:
```
df_new.nunique()
```

Out[ ]:
```
Survived     2
Pclass       3
Sex          2
Age         71
SibSp        7
Parch        7
Fare        91
Embarked     3
Deck         9
Title        5
dtype: int64
```

In [ ]:
```
df_new.columns
```

Out[ ]:
```
Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
       'Embarked', 'Deck', 'Title'],
      dtype='object')
```

In [ ]:
```
print(df_new['Survived'].value_counts())
print(df_new['Pclass'].value_counts())
print(df_new['Sex'].value_counts())
print(df_new['SibSp'].value_counts())
print(df_new['Embarked'].value_counts())
```

```
Survived
0    549
1    342
Name: count, dtype: int64
Pclass
3    491
1    216
2    184
Name: count, dtype: int64
Sex
0    577
1    314
Name: count, dtype: int64
SibSp
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: count, dtype: int64
Embarked
0    646
1    168
2     77
Name: count, dtype: int64
```

In [ ]:
```
total = df_new.isnull().sum().sort_values(ascending=False)
pcnt_1 = df_new.isnull().sum()/df_new.isnull().count()*100
pcnt_2 = (round(pcnt_1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, pcnt_2], axis=1, keys=['Total', '%'])
missing_data.head(5)
```

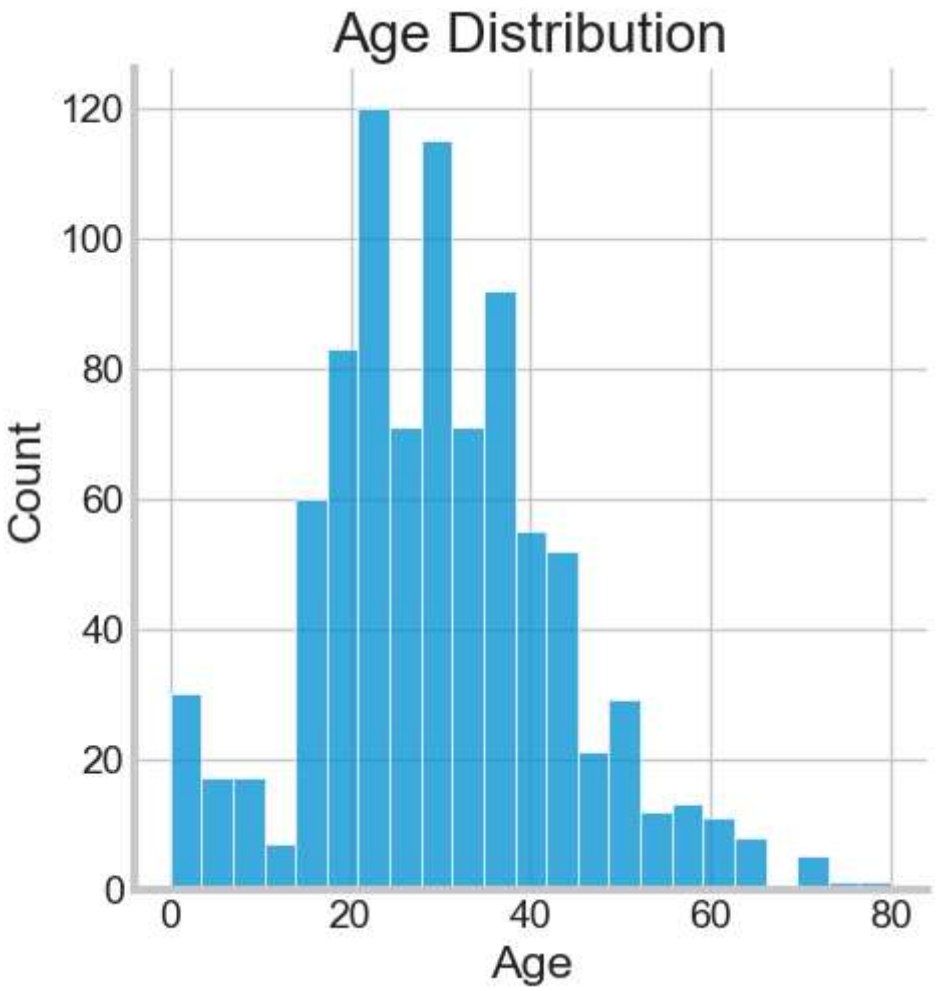
Out[ ]:

	Total	%
Survived	0	0.0
Pclass	0	0.0
Sex	0	0.0
Age	0	0.0
SibSp	0	0.0

In [ ]:
```
plt.figure(figsize=(4, 3))
sns.displot(df_new['Age'])
```

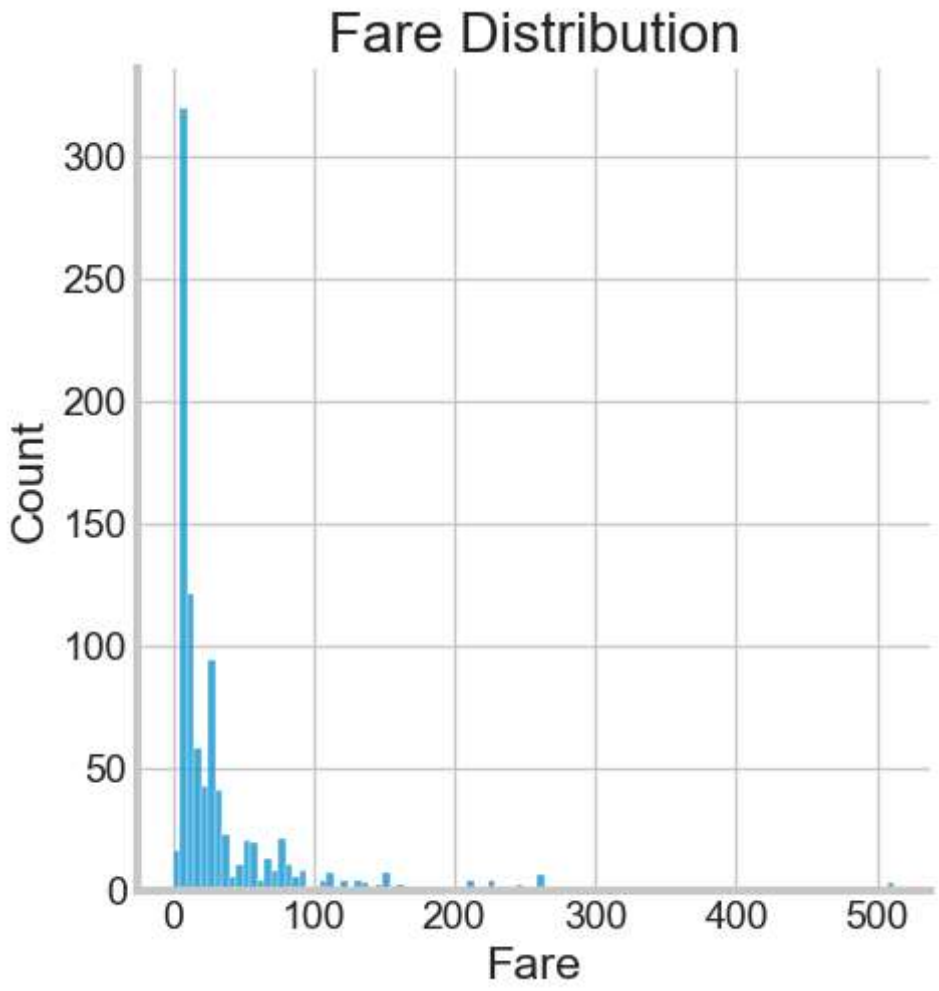
```
plt.title('Age Distribution')
plt.show()
```

```
<Figure size 400x300 with 0 Axes>
```

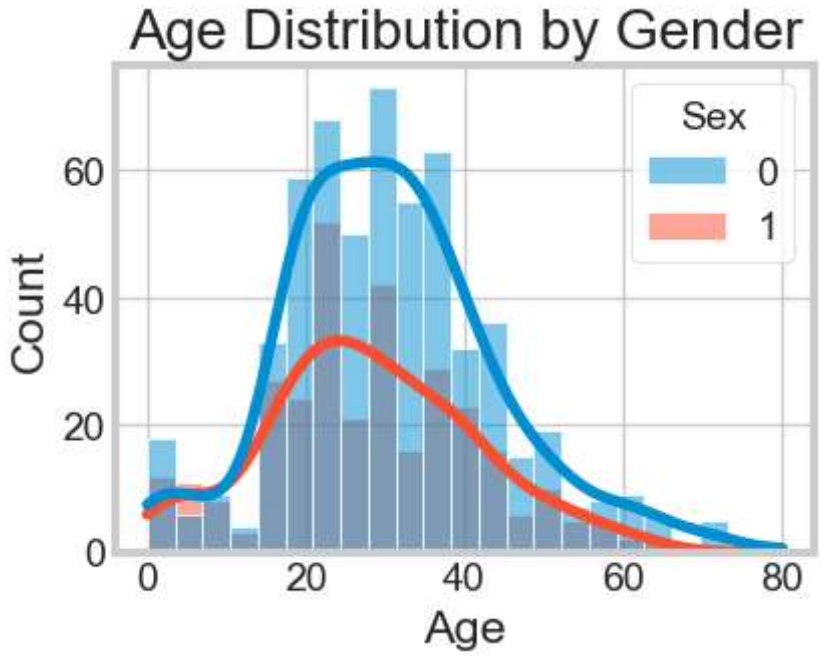


In [ ]:
```
plt.figure(figsize=(4, 3))
sns.displot(df_new['Fare'])
plt.title('Fare Distribution')
plt.show()
```

```
<Figure size 400x300 with 0 Axes>
```

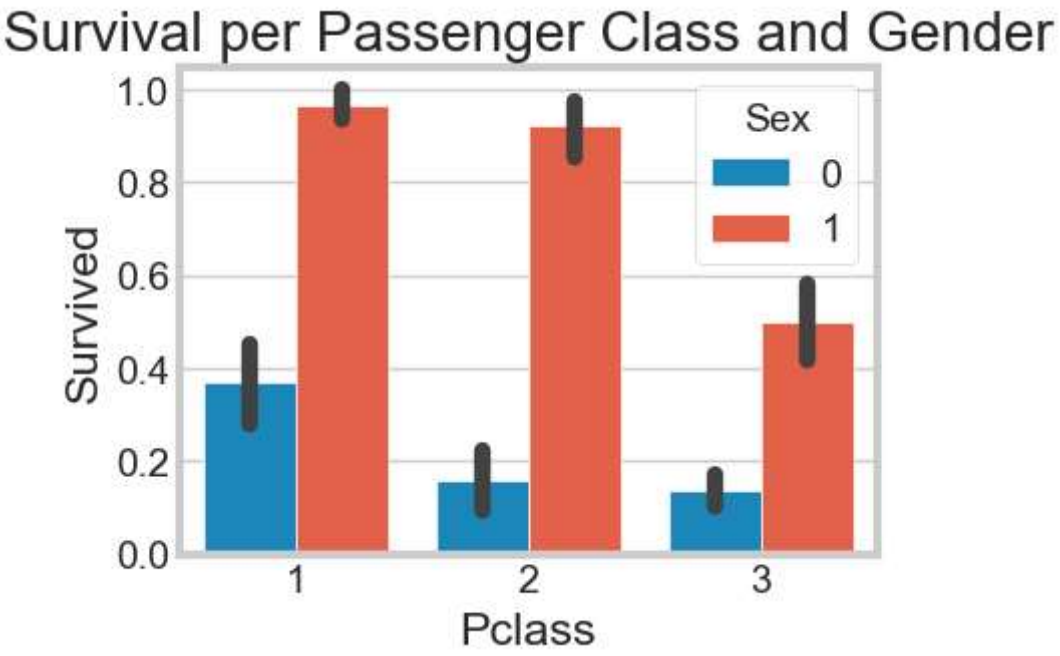


In [ ]:
```
plt.figure(figsize=(4, 3))
sns.histplot(data=df_new, x='Age', kde=True, hue='Sex')
plt.title('Age Distribution by Gender')
plt.show()
```

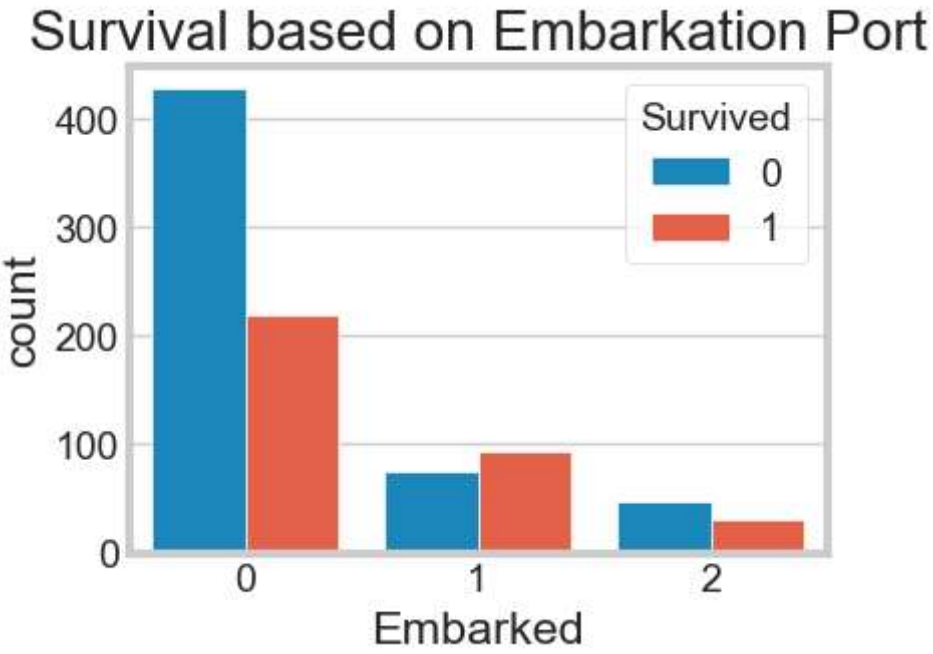


In [ ]:
```
plt.figure(figsize=(4, 3))
sns.barplot(data=df_new, x='Pclass', y='Survived', hue='Sex')
```

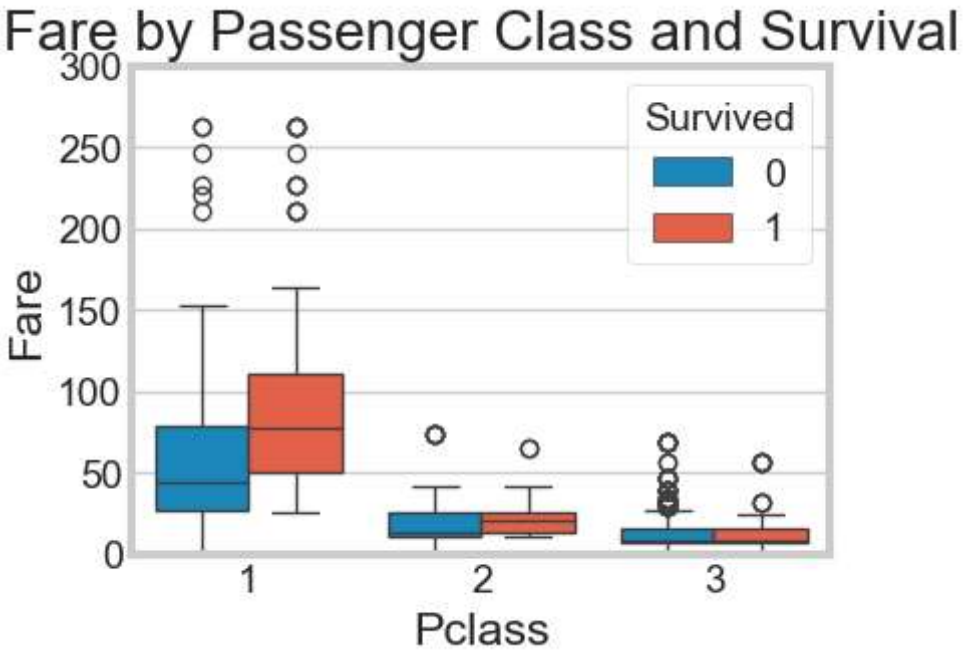
```
plt.title('Survival per Passenger Class and Gender')
plt.show()
```



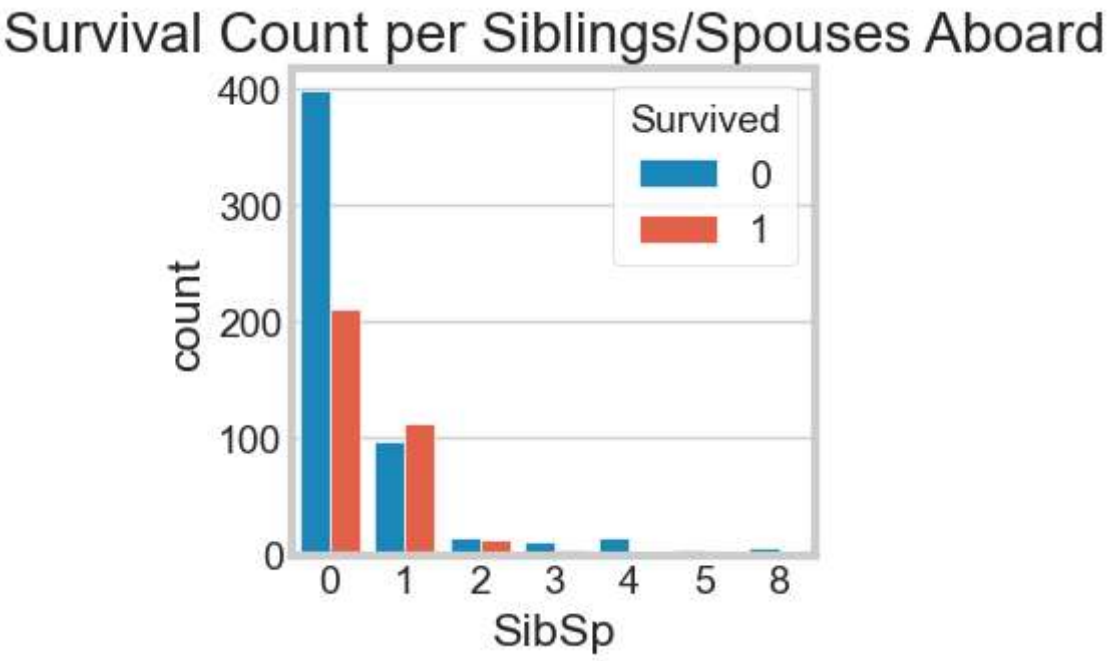
In [ ]:
```
plt.figure(figsize=(4, 3))
sns.countplot(data=df_new, x='Embarked', hue='Survived')
plt.title('Survival based on Embarkation Port')
plt.show()
```



In [ ]:
```
plt.figure(figsize=(4, 3))
sns.boxplot(data=df_new, x='Pclass', y='Fare', hue='Survived')
plt.ylim(0, 300)  # Limiting y-axis to 300 for better visualization
plt.title('Fare by Passenger Class and Survival')
plt.show()
```



In [ ]:
```
plt.figure(figsize=(3, 3))
sns.countplot(data=df_new, x='SibSp', hue='Survived')
plt.title('Survival Count per Siblings/Spouses Aboard')
plt.show()
```



Data Analysis Findings Based on the analysis we've discussed above, here's a summary of findings for the Titanic incident:

Gender and Survival: Women had a significantly higher survival rate than men.

Passenger Class: First-class passengers had a higher survival rate, indicating socio-economic status played a role in survival chances.

Embarkation Port: The survival count varied based on the embarkation port, potentially reflecting the socio-economic distribution of passengers from these ports.

Fare Distribution: The majority of passengers paid lower fares, aligning with a larger number of third-class tickets.

Fare and Survival: Within each passenger class, there wasn't a consistent pattern to suggest that higher fares directly led to better survival chances.

Siblings/Spouses: Those with one sibling or spouse onboard seemed to have a slightly better survival rate than those alone or with many siblings/spouses.

Parents/Children: Passengers traveling alone or with one parent/child had higher survival rates compared to larger families.

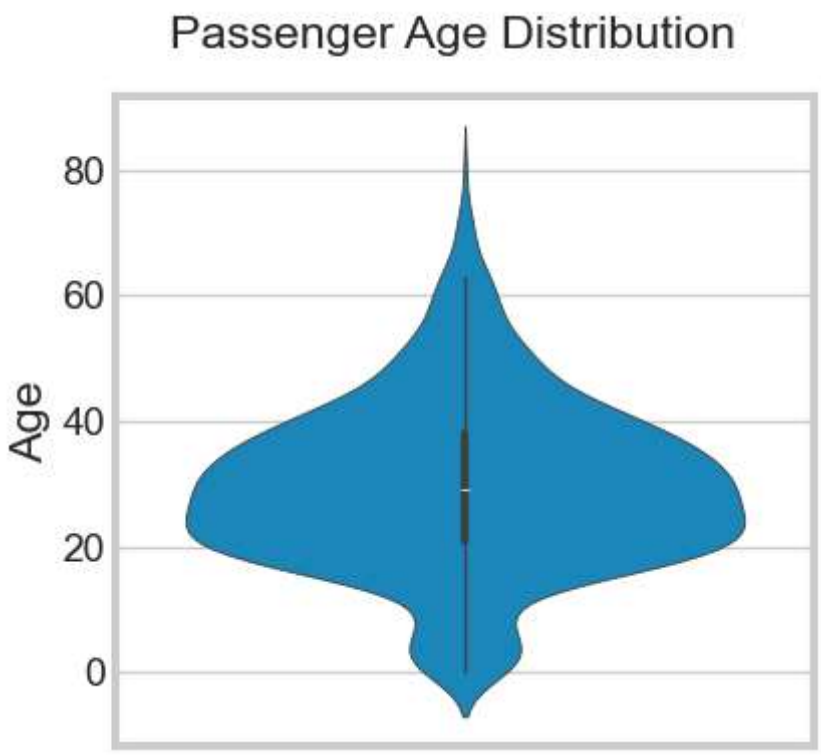
Family Size: Solo travelers and those with a small family size (1-3 members) had better survival outcomes than larger families.

Titles and Survival: Certain titles extracted from names, potentially indicating social status or profession, had varied survival rates.

Age Distribution: Younger passengers (children) had a better survival rate, while the elderly had lower survival chances. Middle-aged individuals, especially males, formed the bulk of casualties.

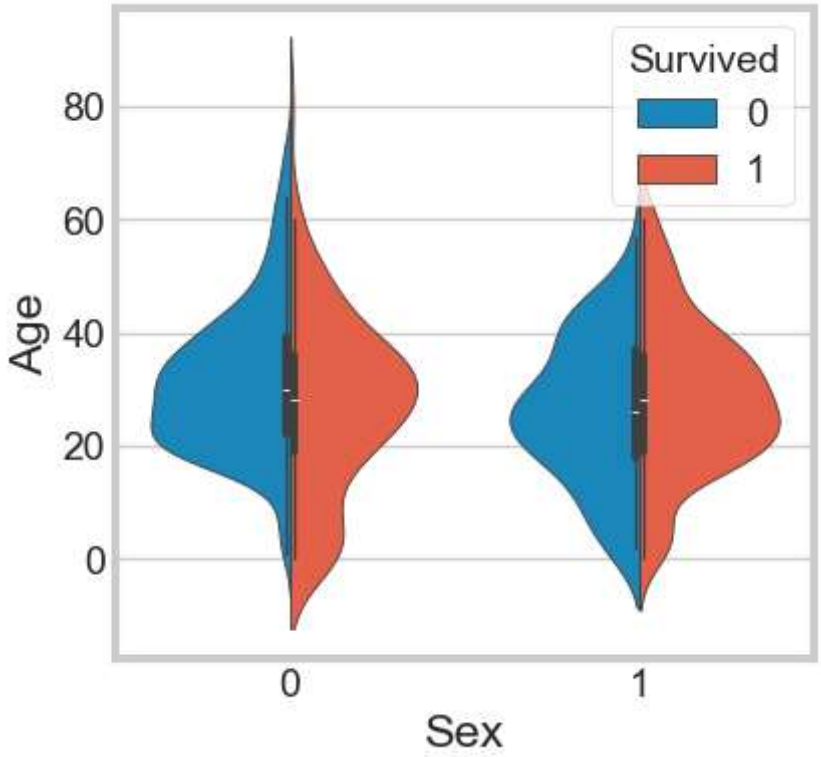
In [ ]:
```
axis = plt.figure(figsize=(4,4))
plt.suptitle('Passenger Age Distribution')
sns.violinplot(df_new['Age'])
```

Out[ ]:
```
<Axes: ylabel='Age'>
```



In [ ]:
```
axis = plt.figure(figsize=(4,4))
sns.violinplot(x = 'Sex', y = 'Age', hue = 'Survived', data = df_new, split = Tr
```

Out[ ]:
```
<Axes: xlabel='Sex', ylabel='Age'>
```

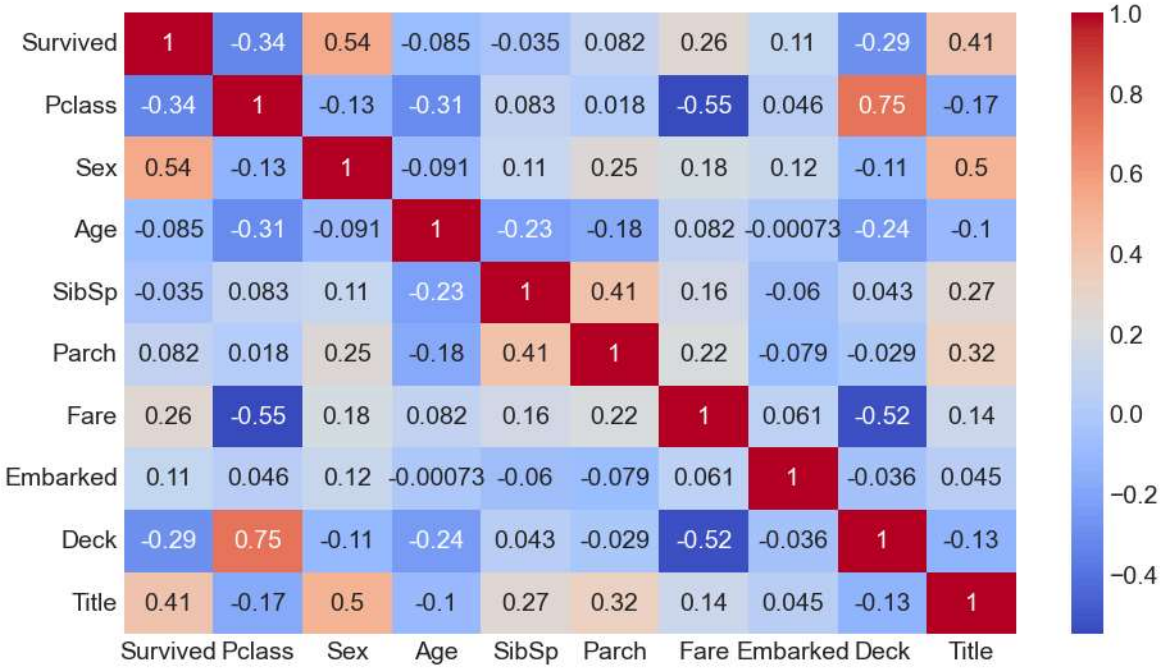


Observations:

This graph showsthe age rangeof men, women and children who were saved. Survival Rate, 1. Good for Children, 2. High for women between the age range 20-50, 3. Less for men with increasing age.

In [ ]:
```
corr = df_new.corr()
plt.figure(figsize = (10, 6))
sns.heatmap(corr, annot = True, cmap = 'coolwarm')
```

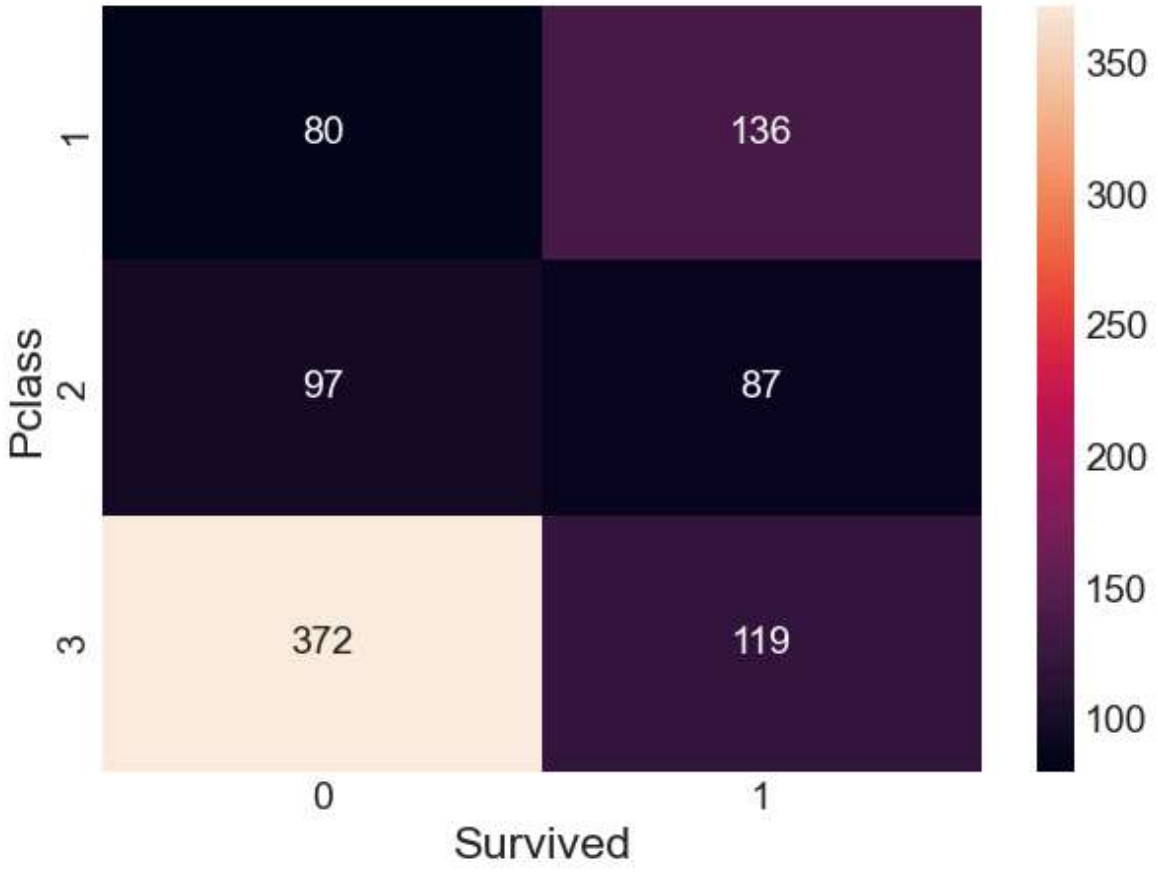
Out[ ]: <Axes: >



The fare shows (-)ve correlation with Pclass, and fare has some level of correlation with all classes.

In [ ]:
```
a = df_new.groupby(['Pclass', 'Survived'])
pclass_survived = a.size().unstack()
sns.heatmap(pclass_survived, annot = True, fmt = "d")
```

Out[ ]: <Axes: xlabel='Survived', ylabel='Pclass'>



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: