In [1]:
```
import pandas as pd
import numpy as np
#
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
#
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
#
import csv
#
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
#
import scipy.stats as stats
#
from sklearn.impute import KNNImputer
#
from sklearn import metrics
#
from sklearn.metrics import classification_report
#
import warnings
warnings.filterwarnings('ignore')
```

Read Data

In [2]:
```
data=pd.read_csv("tested.csv",sep=",",encoding="utf-8")
data.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	Na
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	Na
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	Na
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	Na
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	Na

Pclass: Passenger class (1 = 1st; 2 = 2nd; 3 = 3rd)

Survival: A Boolean indicating whether the passenger survived or not (0 = No; 1 = Yes); this is our target

Name: A Passenger name

Sex: passenger's gender(male/female)

Age: Age, asignificant portion of values aremissing

Sibsp: Number of siblings/spouses aboard

Parch: Number of parents/children aboard

Ticket: Ticket number.

Fare: Passenger fare (British Pound).

Cabin: Doesthe location of the cabin influence chances of survival?

Embarked: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

In [4]:
```
data.describe()
```

Out[4]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

In [5]:
```
data.describe(include="object").T
```

Out[5]:

	count	unique	top	freq
Name	418	418	Kelly, Mr. James	1
Sex	418	2	male	266
Ticket	418	363	PC 17608	5
Cabin	91	76	B57 B59 B63 B66	3
Embarked	418	3	S	270

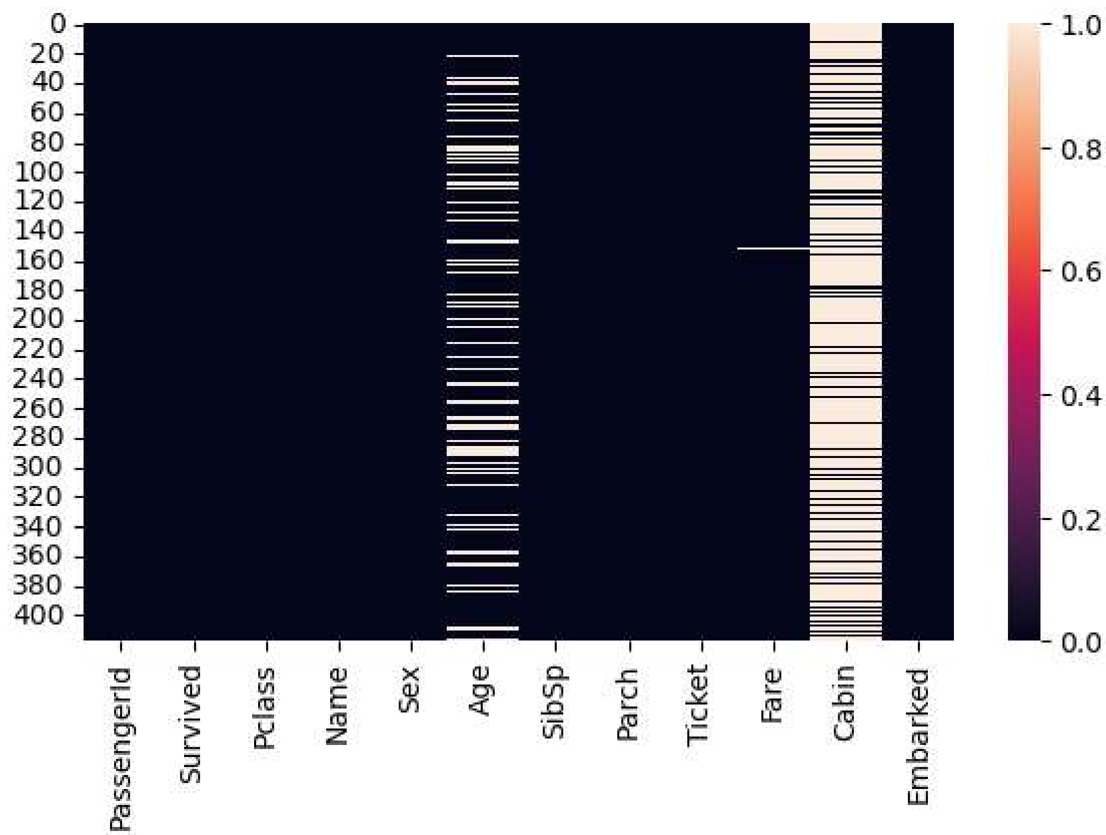
Preprocessing

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [7]:
```
plt.figure(figsize=(7,4))
sns.heatmap(data.isnull())
```

Out[7]:
```
<AxesSubplot:>
```



In [8]:
```
data[data["Age"]<=0].shape
```

Out[8]:
```
(0, 12)
```

In [9]:
```
data["PassengerId"].duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
numerical_data = []
object_data = []

for column in data.columns:
    if data.dtypes[column] != 'object':
        numerical_data.append(column)
    else:
        object_data.append(column)
```

In [11]:
```
numerical_data
```

Out[11]:
```
['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

In [12]:
```
imputer = KNNImputer(n_neighbors=5)
```

In [13]:
```
data[numerical_data] = imputer.fit_transform(data[numerical_data])
```

In [14]:
```
data.isnull().sum()
```

Out[14]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

Random Choice

In [15]:
```
for column in data.columns:
    missing_indices = data[data[column].isnull()].index
    available_values = data[column].dropna()

    for index in missing_indices:
        random_choice = np.random.choice(available_values)
        data.at[index, column] = random_choice
```

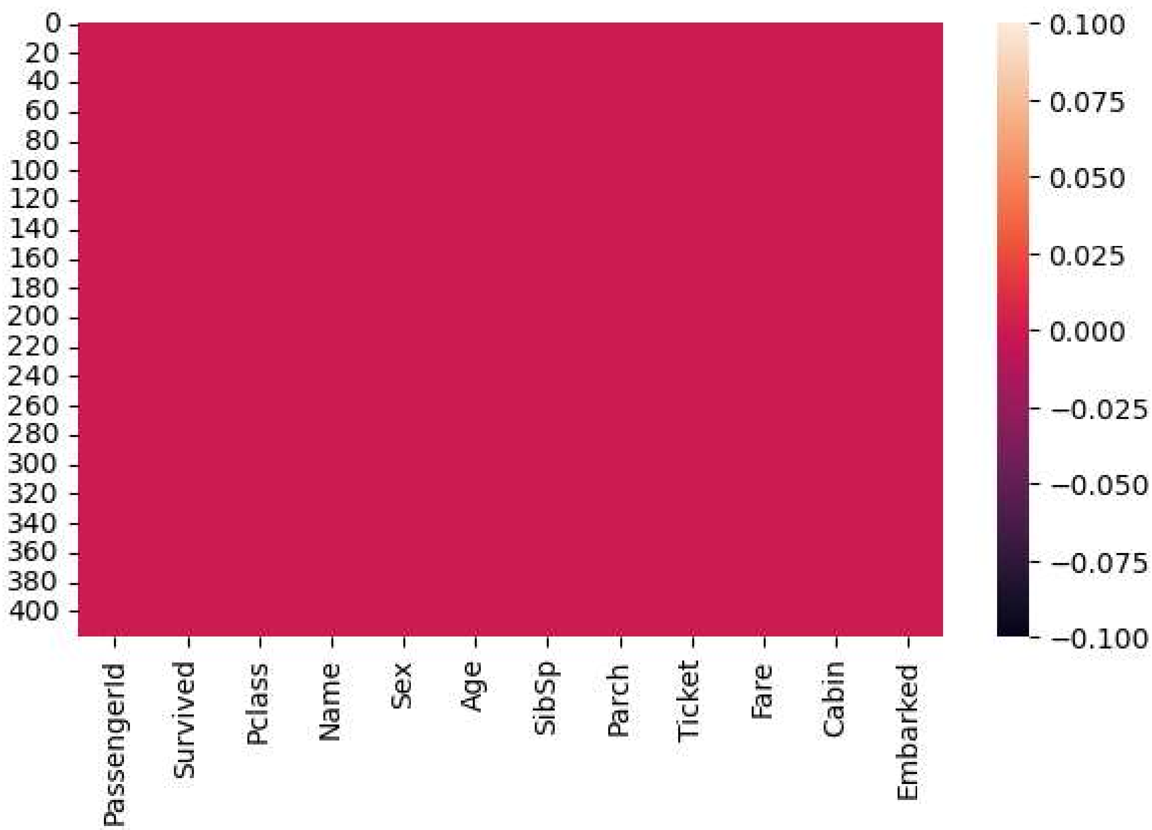
In [16]:
```
data.isnull().sum()
```

Out[16]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

In [17]:
```
data['Fare']=data['Fare'].round(2)
```

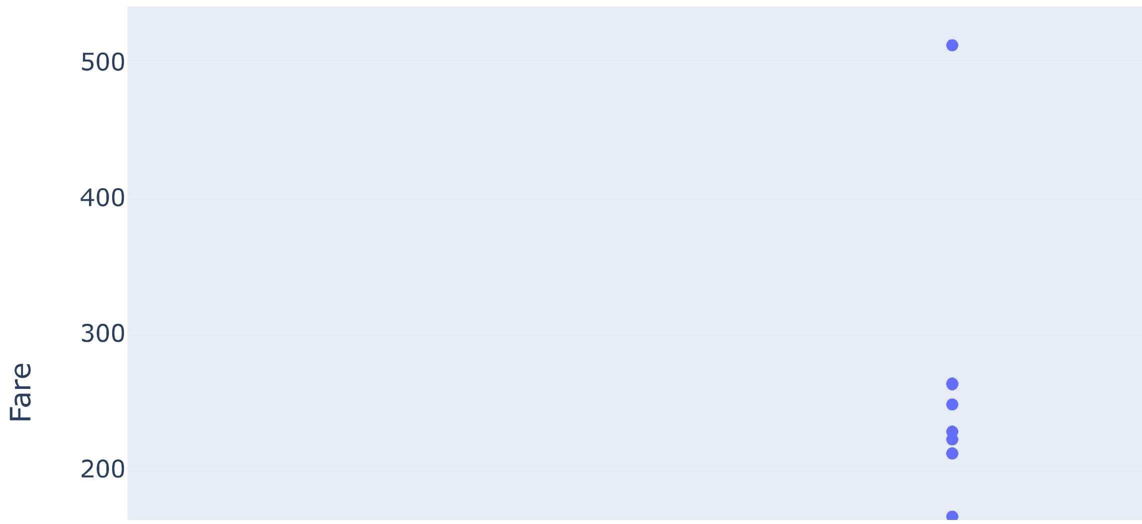
In [18]:
```
plt.figure(figsize=(7,4))
sns.heatmap(data.isnull())
```

Out[18]:
```
<AxesSubplot:>
```



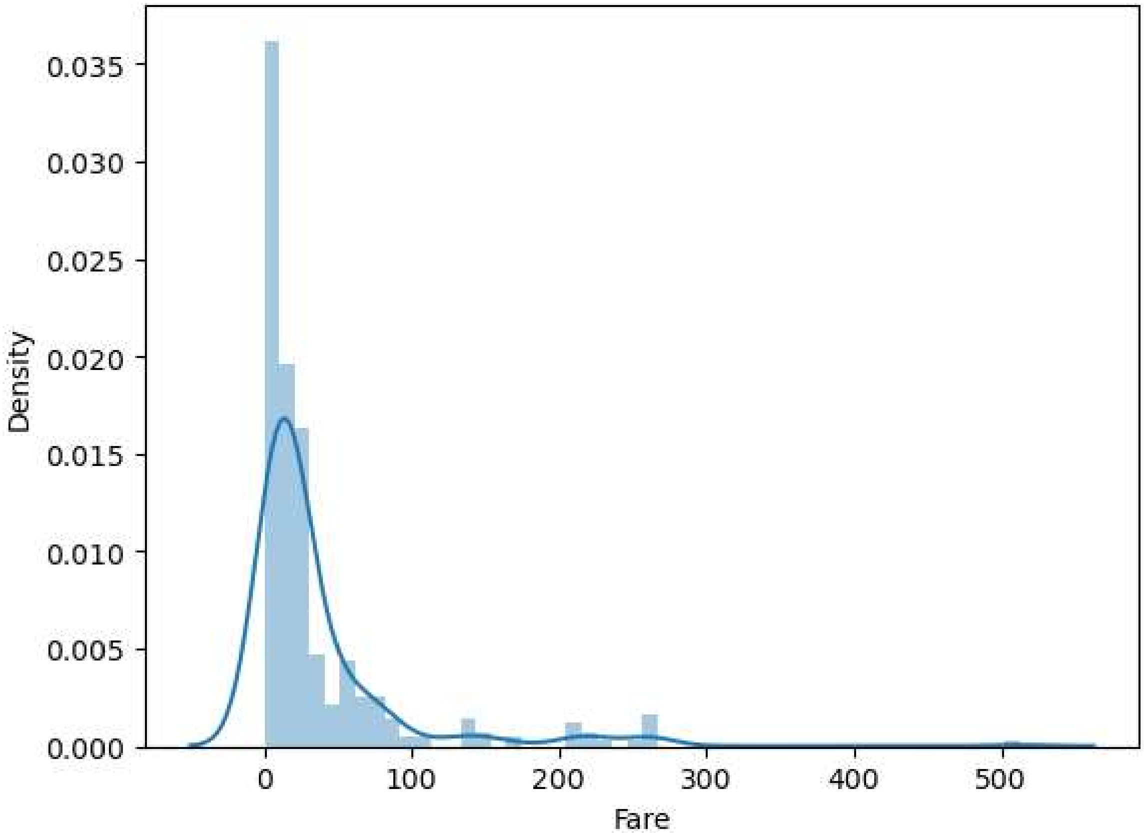
Check & Remove Outliers

In [19]:
```
plt.figure(figsize=(7,5))
px.box(data_frame=data, y="Fare")
```




```
<Figure size 700x500 with 0 Axes>
```

In [20]:
```
sns.distplot(data["Fare"])
plt.show()
```



In [21]:
```
(data["Fare"]>=76).sum()
```

Out[21]: 45

In [22]:
```
#find the limits
upper_limit=data["Fare"].mean() + 3*data["Fare"].std()
lower_limit=data["Fare"].mean() - 3*data["Fare"].std()
print("upper limit: ",upper_limit)
print("lower limit: ",lower_limit)
```

```
upper limit:  203.31658471212668
lower limit:  -131.891321554232
```

In [23]:
```
#find the outliers
outliers_df=data.loc[(data["Fare"]> upper_limit) |(data["Fare"] < lower_limit)
outliers_df.shape
```

Out[23]: (18, 12)

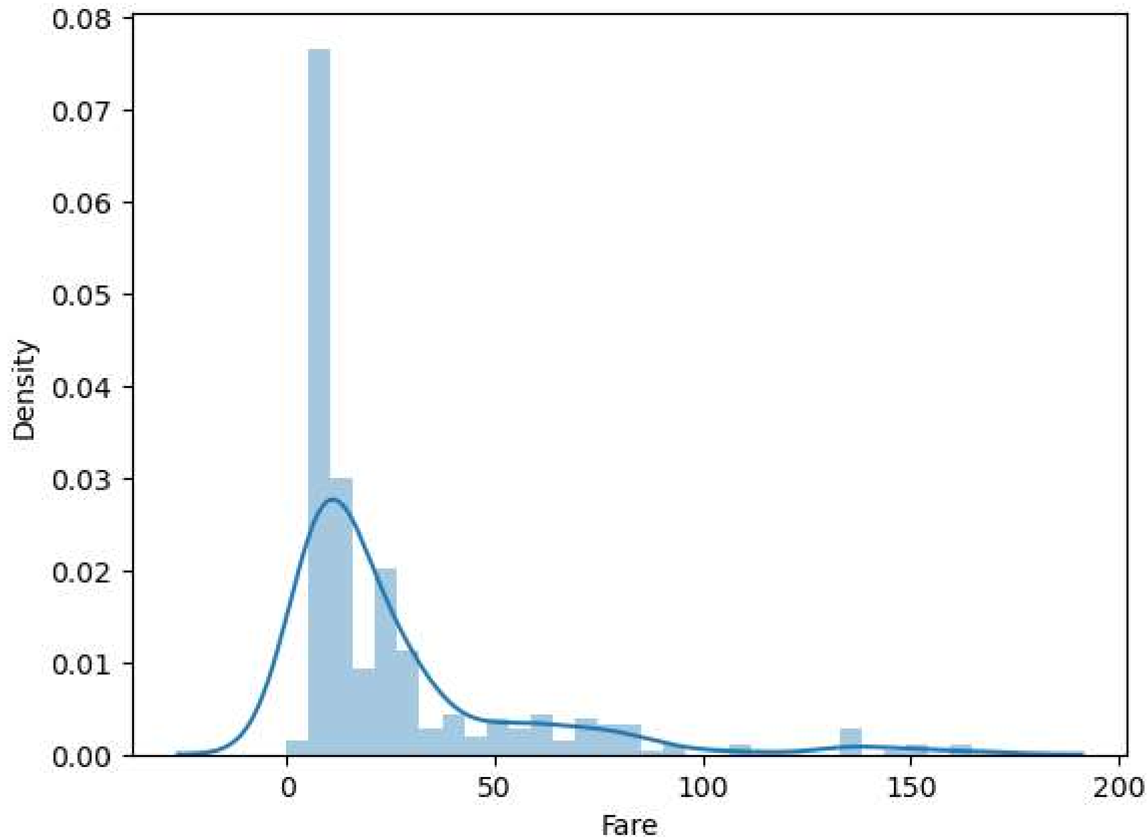
In [24]:
```
#remove outliers from the data
new_df =data.loc[(data["Fare"]< upper_limit) & (data["Fare"] > lower_limit)]
print("before removing the outliers: ",len(data))
print("after removing the outliers: ",len(new_df))
print("the outliers: ",len(data)-len(new_df))
```

```
before removing the outliers:  418
after removing the outliers:  400
the outliers:  18
```

In [25]:
```
plt.figure(figsize=(7,5))
px.box(data_frame=new_df, y="Fare")
```


```
<Figure size 700x500 with 0 Axes>
```

In [26]:
```
sns.distplot(new_df["Fare"])
plt.show()
```



In [27]:
```
file_path = "new_df.csv"

with open(file_path, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(new_df)

print("Data saved to", file_path)
```

```
Data saved to new_df.csv
```

EDA

In [28]:
```
from pandas_profiling import ProfileReport

#EDA using pandas-profiling
profile = ProfileReport(pd.read_csv('new_df.csv'), explorative=True)

#Saving results to a HTML file
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	11
Number of observations	11
Missing cells	64
Missing cells (%)	52.9%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	4.6 KiB
Average record size in memory	426.4 B

Variable types

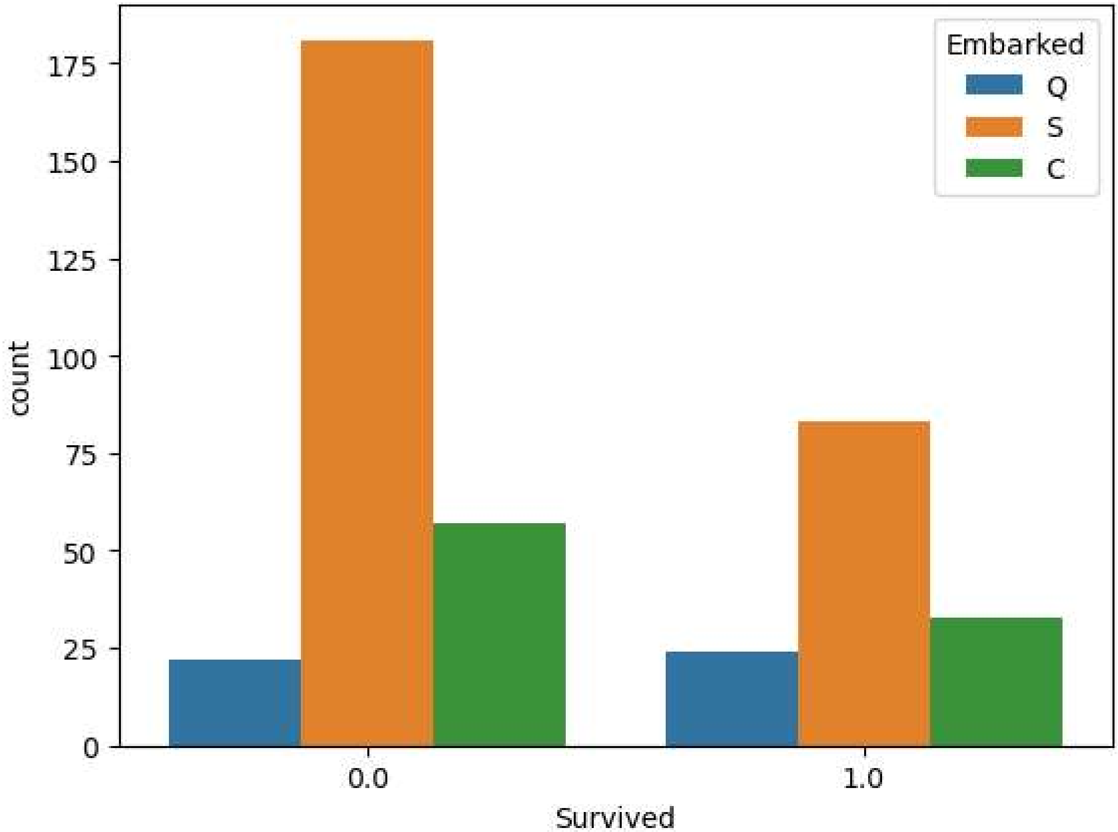
Categorical	8
Unsupported	3

Alerts

`g` has constant value "e"	Constant
`e.1` has constant value "d"	Constant
`P` is highly overall correlated with `e` and 1 other fields (e, n)	High correlation

Out[28]:

In [29]:
```
sns.countplot( x='Survived', data=new_df, hue="Embarked");
```



In [30]:
```
plt.figure(figsize=(10,4))
sns.heatmap(new_df.corr(),annot=True)
plt.show
```

Out[30]: <function matplotlib.pyplot.show(close=None, block=None)>

Preparation

In [31]:
```
new_df["Embarked"].value_counts()
```

Out[31]:
```
S    264
C     90
Q     46
Name: Embarked, dtype: int64
```

In [32]:
```
new_df["Embarked"]=new_df["Embarked"].replace("S",0)
new_df["Embarked"]=new_df["Embarked"].replace("C",1)
new_df["Embarked"]=new_df["Embarked"].replace("Q",2)
```

In [33]:
```
new_df["Sex"].value_counts()
```

Out[33]:
```
male      260
female    140
Name: Sex, dtype: int64
```

In [34]:
```
new_df["Sex"]=new_df["Sex"].replace("male",0)
new_df["Sex"]=new_df["Sex"].replace("female",1)
```

In [35]:
```
new_df
```

Out[35]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	892.0	0.0	3.0	Kelly, Mr. James	0	34.5	0.0	0.0	330911	7.83	
1	893.0	1.0	3.0	Wilkes, Mrs. James (Ellen Needs)	1	47.0	1.0	0.0	363272	7.00	
2	894.0	0.0	2.0	Myles, Mr. Thomas Francis	0	62.0	0.0	0.0	240276	9.69	
3	895.0	0.0	3.0	Wirz, Mr. Albert	0	27.0	0.0	0.0	315154	8.66	
4	896.0	1.0	3.0	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	1	22.0	1.0	1.0	3101298	12.29	F
...	...	...	...	...	...	...	...	...	...	...	
413	1305.0	0.0	3.0	Spector, Mr. Woolf	0	22.5	0.0	0.0	A.5. 3236	8.05	
414	1306.0	1.0	1.0	Oliva y Ocana, Dona. Fermina	1	39.0	0.0	0.0	PC 17758	108.90	
415	1307.0	0.0	3.0	Saether, Mr. Simon Sivertsen	0	38.5	0.0	0.0	SOTON/O.Q. 3101262	7.25	
416	1308.0	0.0	3.0	Ware, Mr. Frederick	0	22.5	0.0	0.0	359309	8.05	
417	1309.0	0.0	3.0	Peter, Master. Michael J	0	26.5	1.0	1.0	2668	22.36	

400 rows × 12 columns

In [36]:
```
new_df["Embarked"]=new_df["Embarked"].astype("int64")
new_df["Sex"]=new_df["Sex"].astype("int64")
new_df["PassengerId"]=new_df["PassengerId"].astype("int64")
new_df["Pclass"]=new_df["Pclass"].astype("int64")
new_df["Age"]=new_df["Age"].astype("int64")
new_df["SibSp"]=new_df["SibSp"].astype("int64")
new_df["Parch"]=new_df["Parch"].astype("int64")
```

Feature Selection

In [37]:
```
# Create a contingency table for each categorical column
for col in new_df.columns:
    contingency_table = pd.crosstab(new_df[col],new_df['Survived'])
    # Apply the chi-square test
    chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
    print(f"Chi-square test results for {col}:")
    print(f"Chi-square statistic: {chi2}")
    print(f"P-value: {p}")
    print(f"Degrees of freedom: {dof}")
    print(f"Expected frequencies table:\n{expected}\n")
```

```
Chi-square test results for PassengerId:
Chi-square statistic: 400.0
P-value: 0.4764888981006025
Degrees of freedom: 399
Expected frequencies table:
[[0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
 [0.65 0.35]
```

Encoding

In [38]:
```
new_df =new_df.apply(lambda x: x.astype('category').cat.codes)
new_df.head()
```

Out[38]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
0	0	0	2	196	0	33	0	0	149	21	12	
1	1	1	2	385	1	46	1	0	218	5	63	
2	2	0	1	258	0	58	0	0	71	37	22	
3	3	0	2	390	0	26	0	0	144	30	67	
4	4	1	2	169	1	21	1	1	135	42	69	

Split Data

In [39]:
```
X = new_df.drop(['Survived','Name','Cabin','Ticket'],axis=1)
y = new_df['Survived']
```

Scaling

In [40]:
```
sc = StandardScaler()
X = sc.fit_transform(X)
```

Modling

1-Random Forest

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [42]:
```
RF = RandomForestClassifier(n_estimators=100, random_state=42)
RF.fit(X_train, y_train)
```

Out[42]:
```
RandomForestClassifier(random_state=42)
```

In [43]:
```
print(RF.score(X_train,y_train))
print(RF.score(X_test,y_test))
```

```
1.0
1.0
```

In [44]:
```
y_pred=RF.predict(X_test)
y_pred
```

Out[44]:
```
array([0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0], dtype=int8)
```

In [45]:
```
df=pd.DataFrame({"y_predect":y_pred,"y_test":y_test})
df
```

Out[45]:

	y_predect	y_test
223	0	0
294	0	0
34	0	0
224	1	1
101	0	0
...	...	...
260	0	0
241	1	1
386	0	0
188	1	1
303	0	0

80 rows × 2 columns

In [46]:
```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        34

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

2-XGBoost

In [47]:
```
xgboost = xgb.XGBClassifier(objective='multi:softmax', num_class=3, random_sta

xgboost.fit(X_train, y_train)

y_pred = xgboost.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```