Import Librarys

```
import pandas as pd # pandas use to manipulate the data
import matplotlib.pyplot as plt #use for visualization
import seaborn as sns ##use for advance visualization
```

About this file

Use the Titanic dataset to build a model that predicts whether a passenger on the Titanic survived or not. This is a classic beginner project with readily available dat

a. The dataset typically used for this project contains information about individual passengers, such as their age, gender, ticket class, fare, cabin, and whether or not they survived.

Data collection and processing

```
df = pd.read_csv("/content/Titanic-Dataset.csv") #Load the dataset

df.head() #head use for starting 5 rows
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 891,\n  \"fields\": [\
n    {\n      \"column\": \"PassengerId\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 257,\n        \"min\": 1,\n
\"max\": 891,\n        \"num_unique_values\": 891,\n
\"samples\": [\n          710,\n          440,\n          841\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Survived\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Pclass\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 1,\n        \"max\": 3,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          3,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Name\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 891,\n        \"samples\": [\n
\"Moubarek, Master. Halim Gonios (\\\"William George\\\")\",\n
\"Kvillner, Mr. Johan Henrik Johannesson\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sex\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"female\",\n          \"male\"\n        ],\
```

```
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 14.526497332334044,\
n        \"min\": 0.42,\n        \"max\": 80.0,\n
\"num_unique_values\": 88,\n        \"samples\": [\n          0.75,\n
22.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SibSp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 8,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Parch\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 6,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Ticket\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 681,\n        \"samples\": [\n
\"11774\",\n          \"248740\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Fare\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 49.693428597180905,\n
\"min\": 0.0,\n        \"max\": 512.3292,\n
\"num_unique_values\": 248,\n        \"samples\": [\n
11.2417,\n          51.8625\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Cabin\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 147,\n
\"samples\": [\n          \"D45\",\n          \"B49\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Embarked\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"S\",\n          \"C\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
df.info() # info use for know the data type are present in our dataset
how many memory are use this dataset to store data and how many total
columns are present etc
```

```
<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
```

```
df.columns # columns use to know the names of the all columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
df.shape # shape use for know about the total row and columns
```

```
(891, 12)
```

```
df.shape[0]*df.shape[1] #it show the total data present in the
datasets
```

```
10692
```

```
df.isnull().sum() # isnull use to know how many null values are
present in the dataset
```

```
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
```

Hendling missing data

```
# drop the "Cabin" column from the dataframe
df = df.drop(columns='Cabin', axis=1)
```

```
# replacing the missing values in "Age" column with mean value
df['Age'].fillna(df['Age'].mean(), inplace=True)
```

```
<ipython-input-11-bb3c0ec081ae>:2: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Age'].fillna(df['Age'].mean(), inplace=True)
```

```
# finding the mode value of "Embarked" column
print(df['Embarked'].mode())
```

```
0    S
Name: Embarked, dtype: object
```

```
print(df['Embarked'].mode()[0])
```

```
S
```

```
# replacing the missing values in "Embarked" column with mode value
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
```

```
<ipython-input-16-ae2c81114828>:2: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
```

```
# check the number of missing values in each column
df.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
```

```
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

```
# getting some statistical measures about the data
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"PassengerId\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 320.8159711429856,\n
\"min\": 1.0,\n        \"max\": 891.0,\n        \"num_unique_values\":
6,\n        \"samples\": [\n          891.0,\n          446.0,\n
668.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Survived\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 314.8713661874558,\n        \"min\":
0.0,\n        \"max\": 891.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.3838383838383838,\n          1.0,\n
0.4865924542648585\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Pclass\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 314.2523437079693,\n        \"min\": 0.8360712409770513,\n
\"max\": 891.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          891.0,\n          2.308641975308642,\n
3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Age\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 305.2978992449289,\n        \"min\": 0.42,\n        \"max\":
891.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\n
891.0,\n          29.69911764705882,\n          35.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SibSp\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 314.4908277465442,\n
\"min\": 0.0,\n        \"max\": 891.0,\n        \"num_unique_values\":
6,\n        \"samples\": [\n          891.0,\n
0.5230078563411896,\n          8.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Parch\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 314.65971717879,\n
\"min\": 0.0,\n        \"max\": 891.0,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          0.38159371492704824,\n
6.0,\n          0.8060572211299559\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Fare\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 330.6256632228577,\n
\"min\": 0.0,\n        \"max\": 891.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          32.204207968574636,\n
14.4542,\n          891.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```

```
# value of count use for know the no of people Survived and not
Survived
df['Survived'].value_counts()
```

```
Survived
0    549
1    342
Name: count, dtype: int64
```

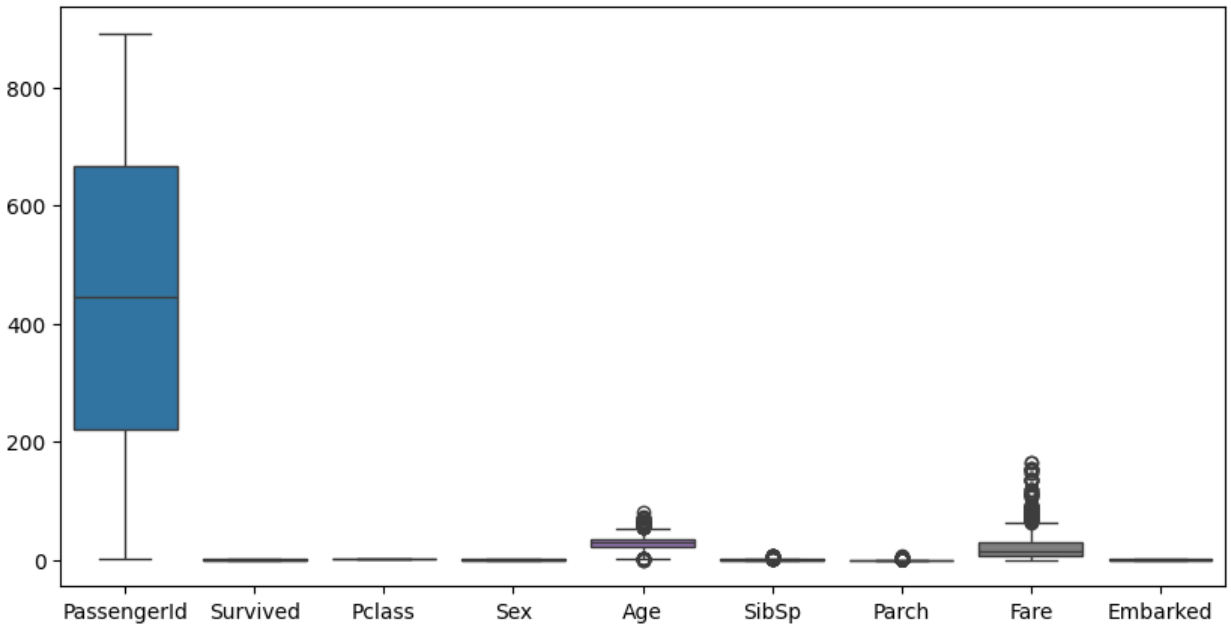
```
# value  count use for know the no of  Male and female
df['Sex'].value_counts()
```

```
Sex
male      577
female    314
Name: count, dtype: int64
```

Outlire

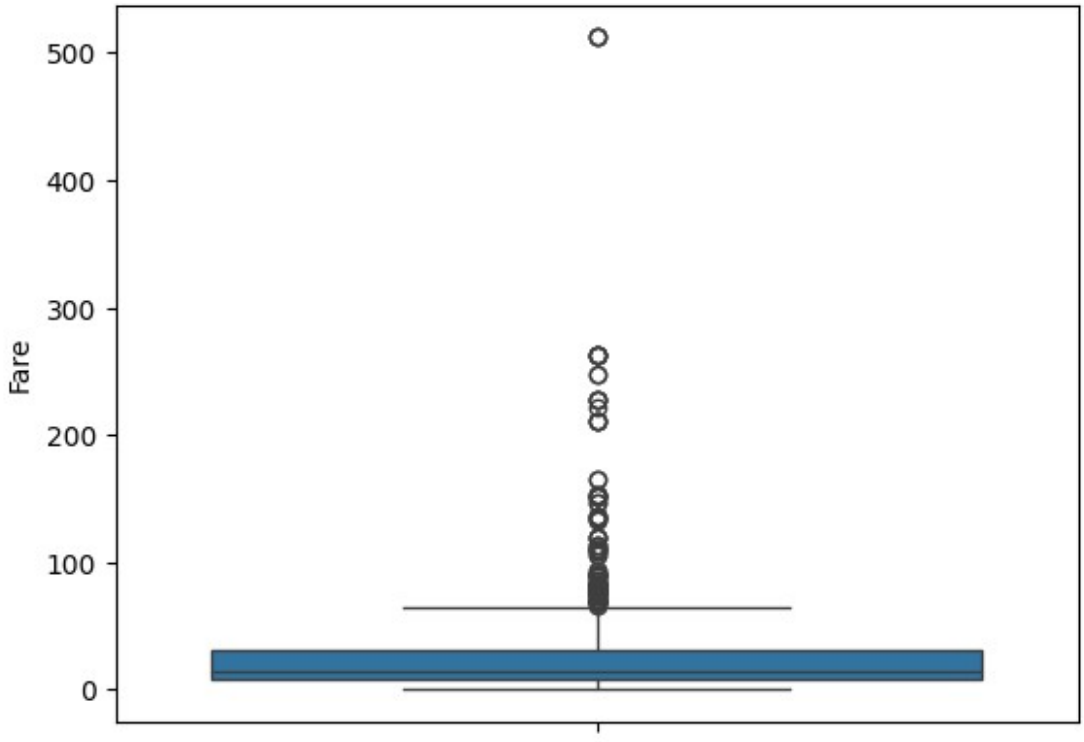
```
plt.figure(figsize=(10,5))
sns.boxplot(df)
```

```
<Axes: >
```




```
sns.boxplot(df["Fare"])
```

```
<Axes: ylabel='Fare'>
```



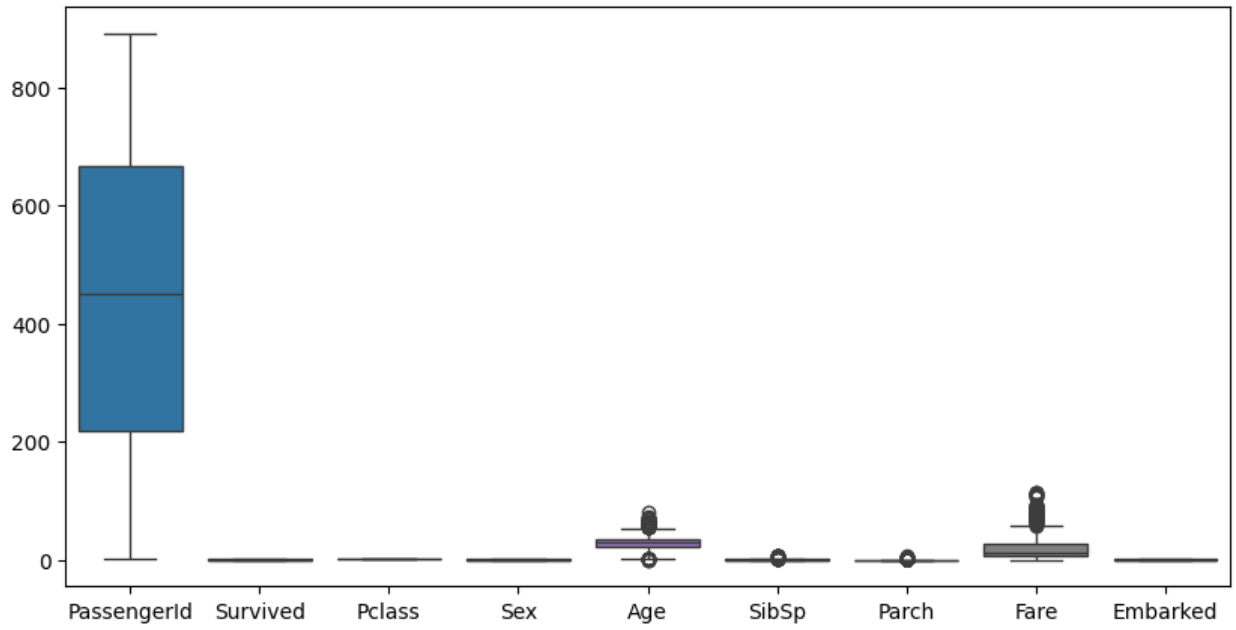
Handling Outlires

```
# handling Outlire withe the help of Z Score
for i in df[["Age","SibSp","Parch","Fare"]]:
    Upper_Boundary = df[i].mean() + 3 * df[i].std()
    Lower_Boundary = df[i].mean() - 3 * df[i].std()
df =df[(df['Fare']<Upper_Boundary) & (df['Fare']>Lower_Boundary)]

df.shape
```

```
(849, 11)
```

```
# we can see we have a 891 rows in dataset and after Handling outlire
we have 849 data

plt.figure(figsize=(10,5))
sns.boxplot(df)
```

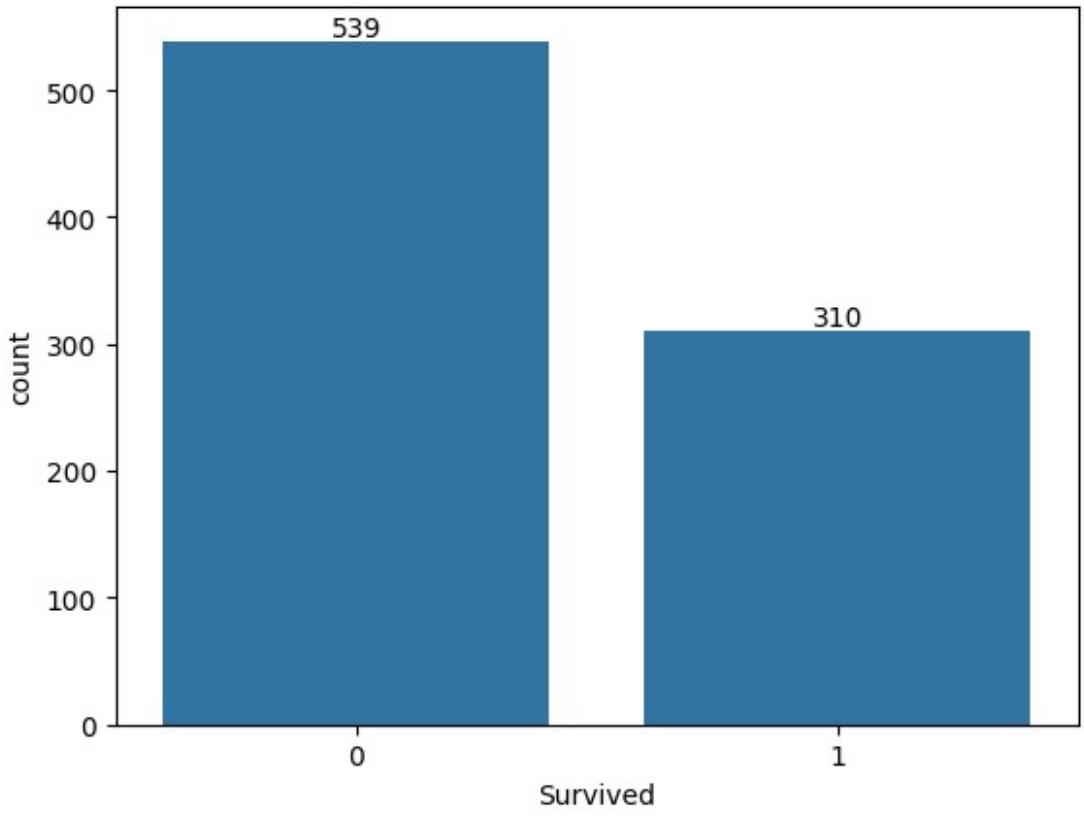
```
<Axes: >
```



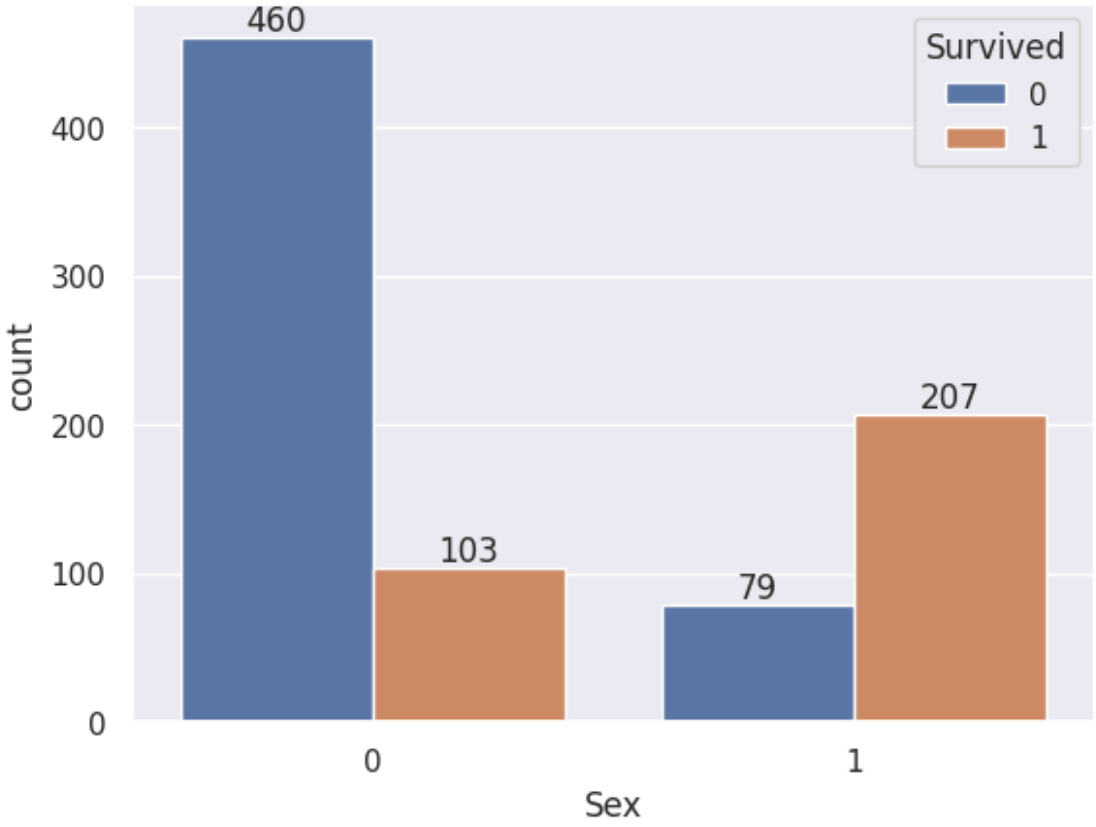
Data Visualization

```
#use a countplot to count the number for Survived peoples on non
survived by bar graph
ax = sns.countplot(x= "Survived",data = df)
ax.bar_label(ax.containers[0])
```

```
[Text(0, 0, '539'), Text(0, 0, '310')]
```

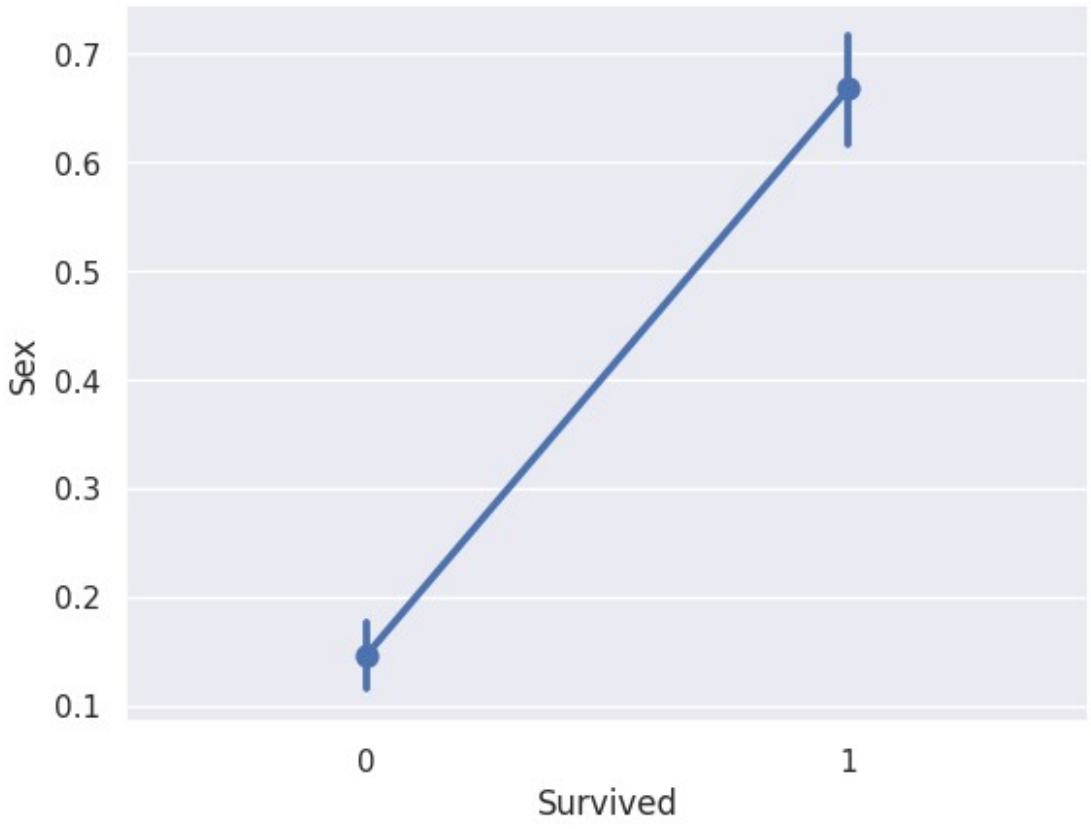



```
sns.set()

#use a countplot to count the number for Survived peoples on non
survived by by gender

ax = sns.countplot(data = df, x = 'Sex', hue = 'Survived')

for bars in ax.containers:
    ax.bar_label(bars)
```



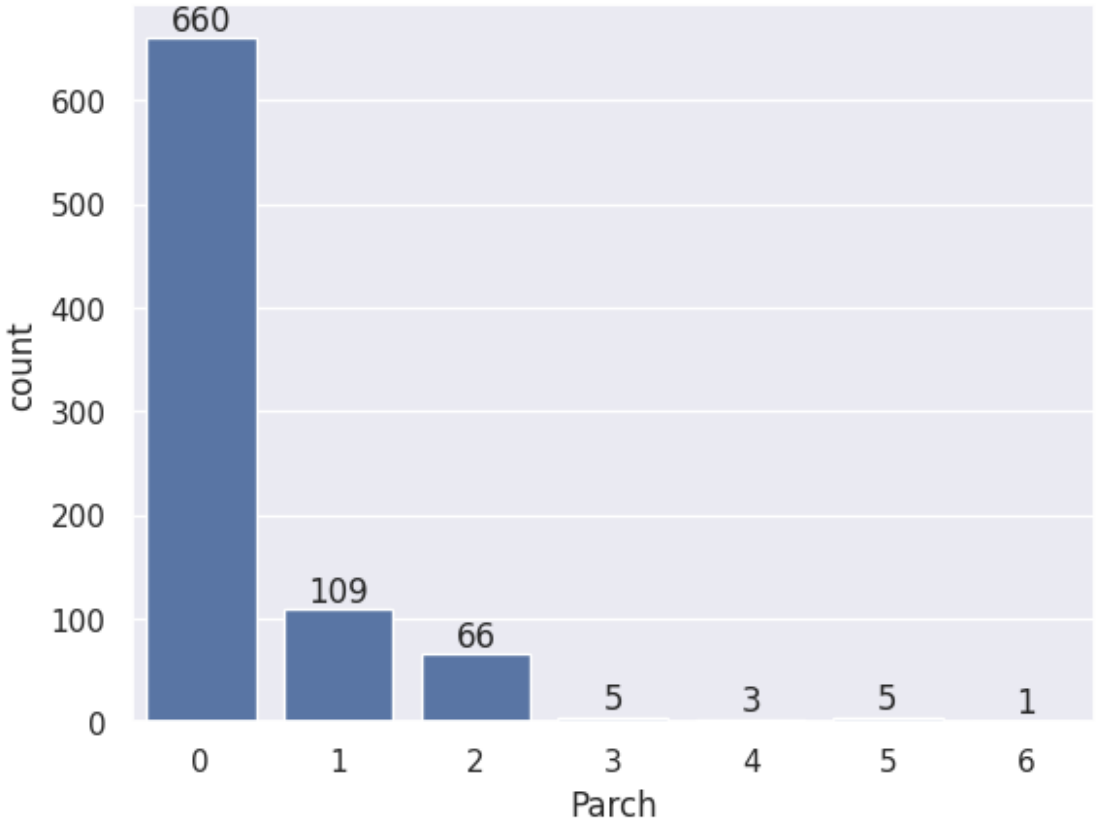

```
sns.pointplot(x = df.Survived,y = df.Sex)
```

```
<Axes: xlabel='Survived', ylabel='Sex'>
```

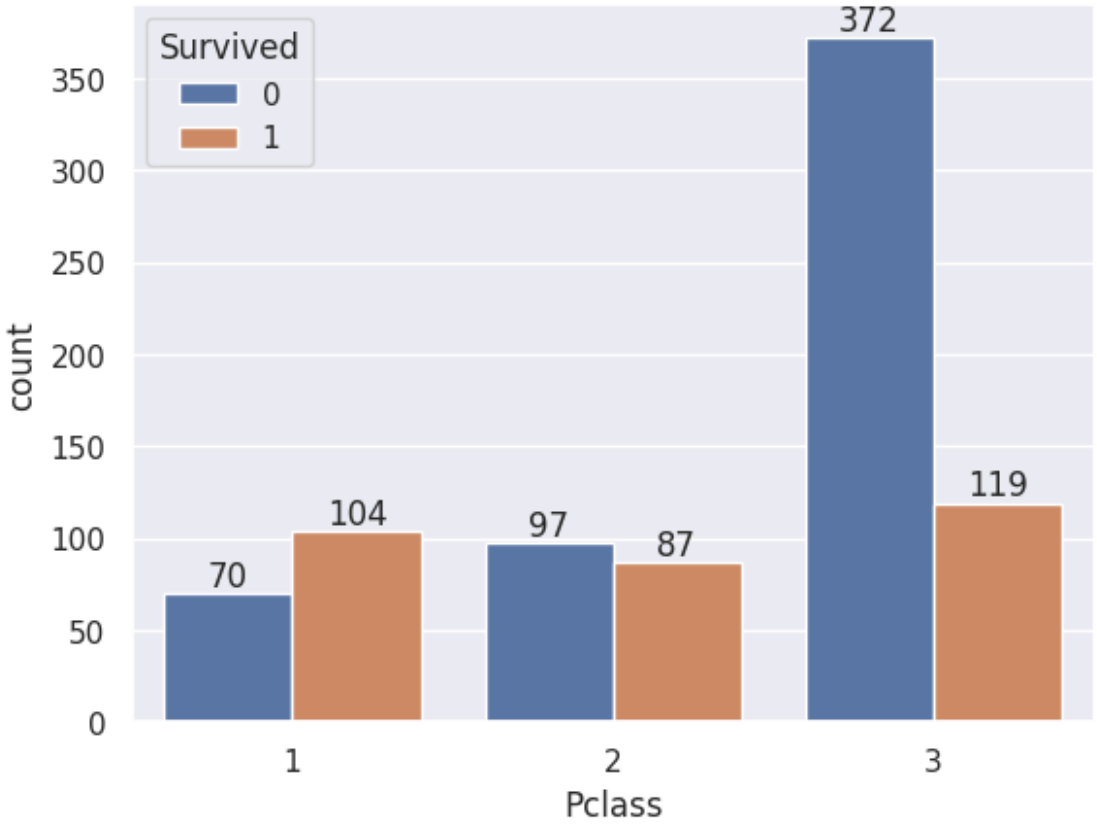



```
#use a countplot to count the number for parents/Children aboard the titanic
ax = sns.countplot(x= "Parch",data = df)
ax.bar_label(ax.containers[0])
```

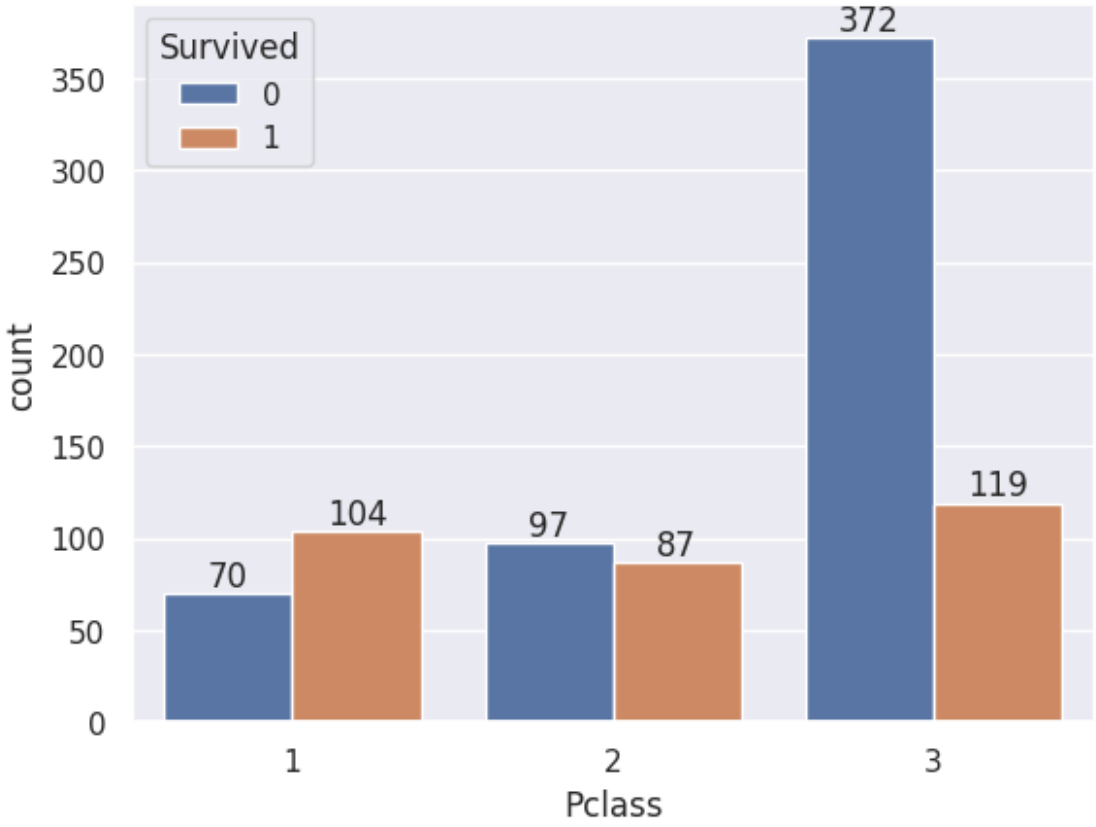
```
[Text(0, 0, '660'),
 Text(0, 0, '109'),
 Text(0, 0, '66'),
 Text(0, 0, '5'),
 Text(0, 0, '3'),
 Text(0, 0, '5'),
 Text(0, 0, '1')]
```



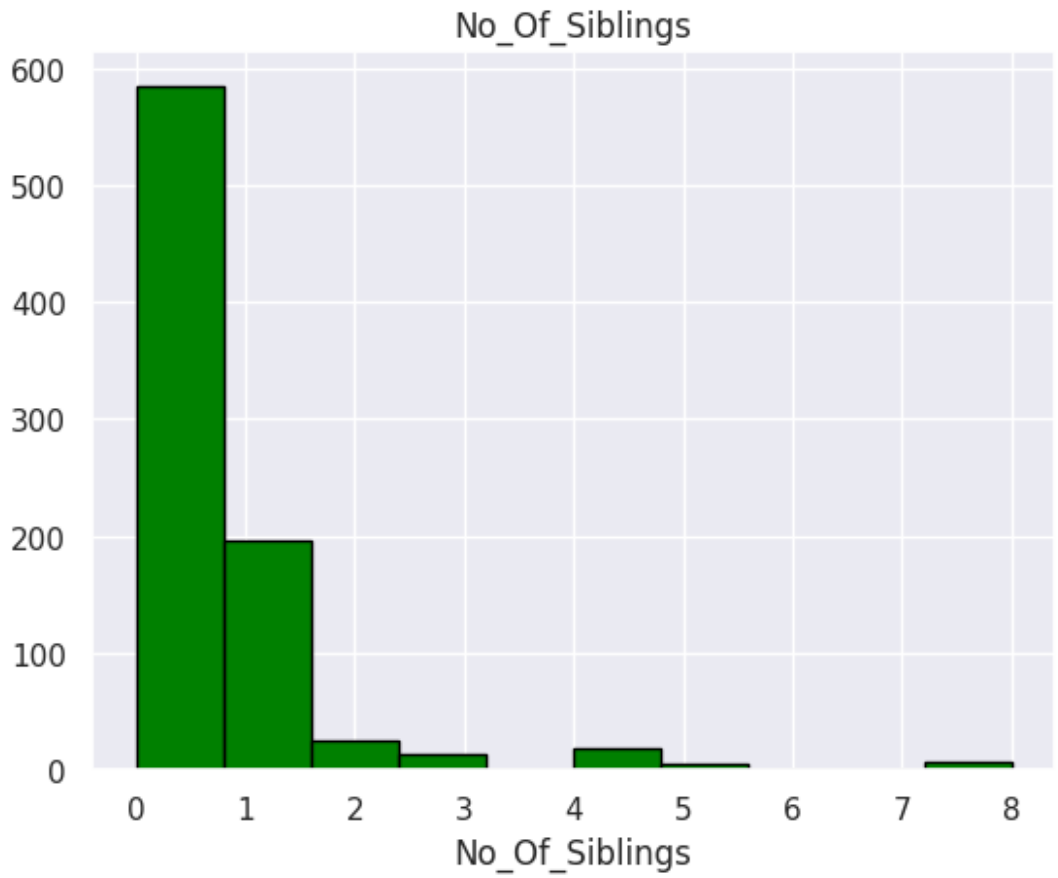

```
ax = sns.countplot(data = df, x = 'Pclass', hue = 'Survived')

for bars in ax.containers:
    ax.bar_label(bars)
```




```
ax = sns.countplot(data = df, x = 'Pclass', hue = 'Survived')

for bars in ax.containers:
    ax.bar_label(bars)
```




```
No_Of_Siblings = df['SibSp']
plt.hist(No_Of_Siblings, color='green', edgecolor='black')
plt.title('No_Of_Siblings')
plt.xlabel('No_Of_Siblings')

Text(0.5, 0, 'No_Of_Siblings')
```



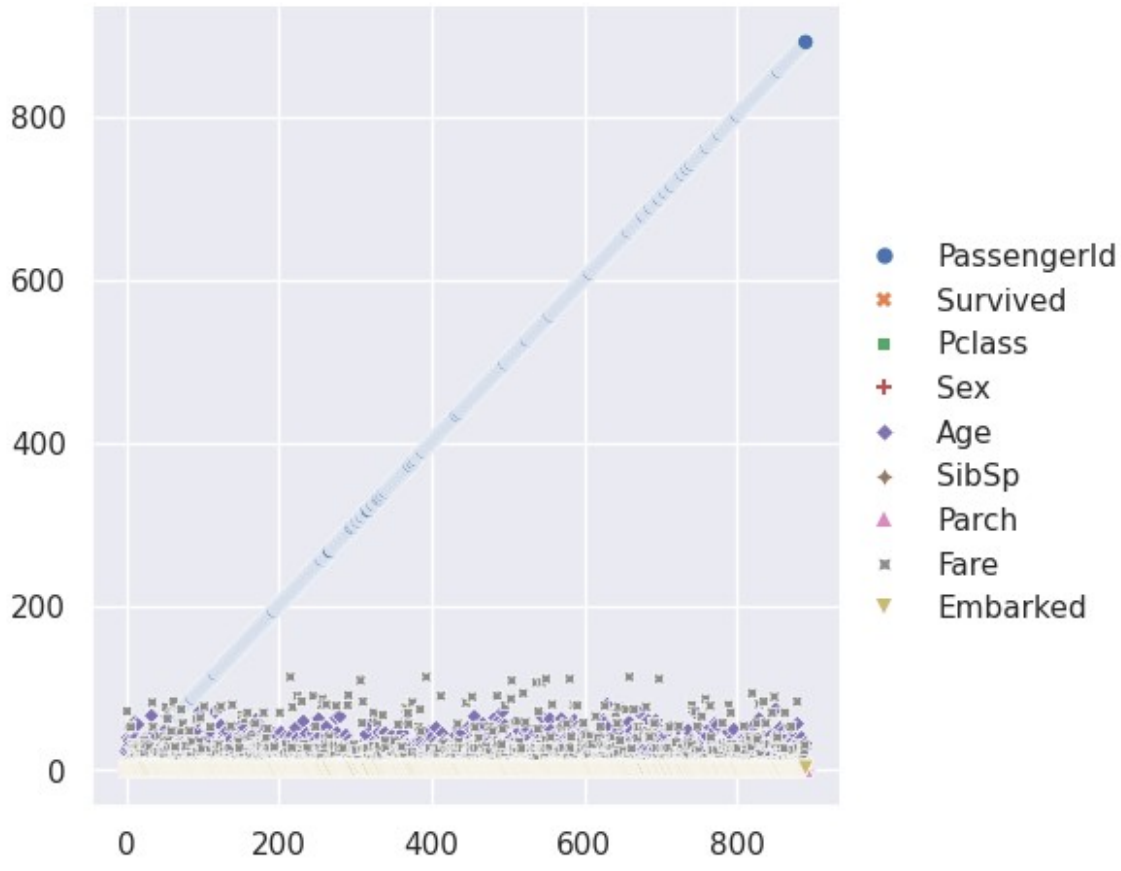

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7995e132fa90>
```


```
plt.figure(figsize =(15,5))
sns.relplot(df)
```

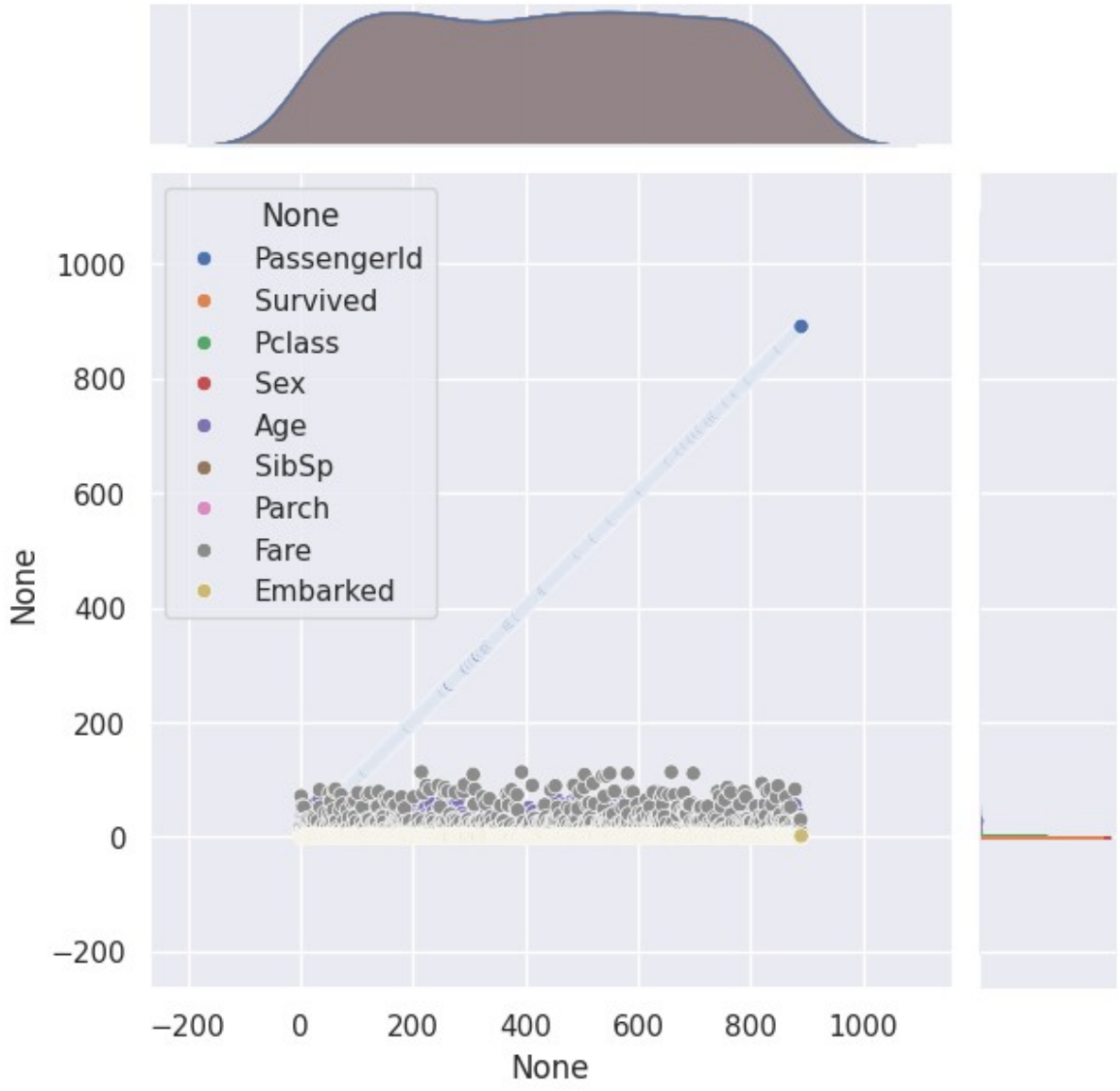
```
<seaborn.axisgrid.FacetGrid at 0x7995db3e7c70>
```

```
<Figure size 1500x500 with 0 Axes>
```



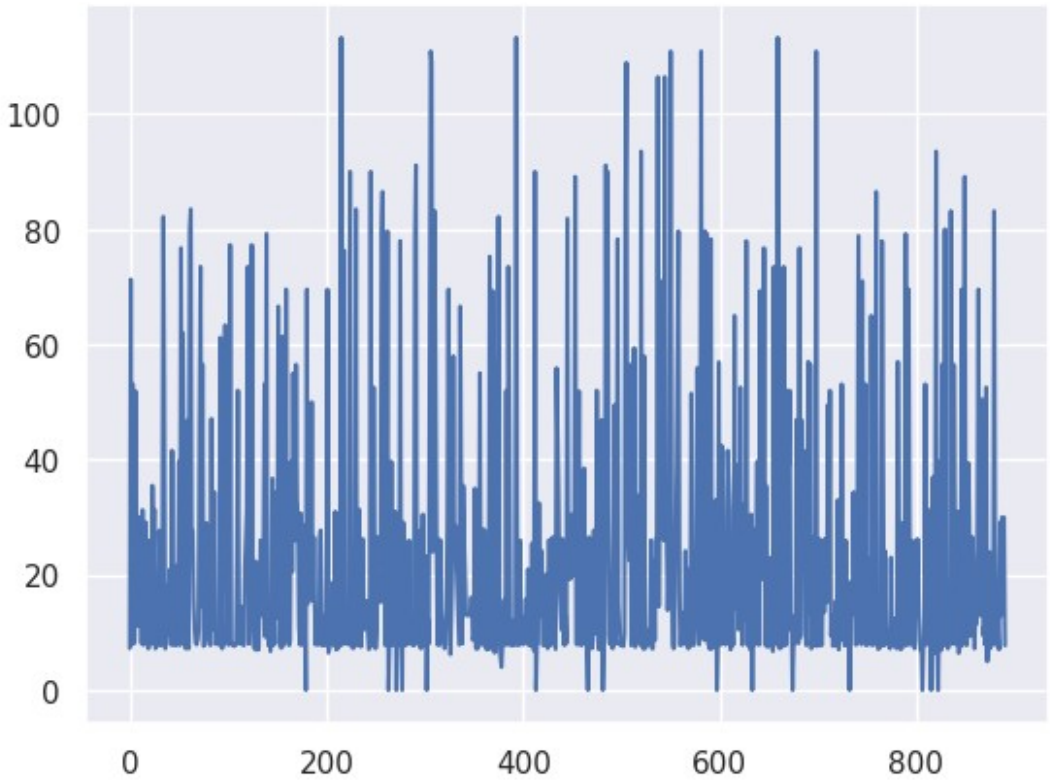

```
sns.jointplot(df)
```

```
<seaborn.axisgrid.JointGrid at 0x7995db314310>
```




```
df['Fare'].plot()
```

```
<Axes: >
```



Encoding the Categorical Columns

```
df['Sex'].value_counts()
```

```
Sex
0    563
1    286
Name: count, dtype: int64
```

```
df['Embarked'].value_counts()
```

```
Embarked
0    622
1    150
2     77
Name: count, dtype: int64
```

```
# converting categorical Columns

df.replace({'Sex':{'male':0,'female':1}, 'Embarked':
{'S':0,'C':1,'Q':2}}, inplace=True)
```

```
<ipython-input-77-f405e8b4fb18>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df.replace({'Sex':{'male':0,'female':1}, 'Embarked':
{'S':0,'C':1,'Q':2}}, inplace=True)
```

```
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 849,\n  \"fields\": [\
n    {\n      \"column\": \"PassengerId\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 258,\n        \"min\": 1,\n
\"max\": 891,\n        \"num_unique_values\": 849,\n
\"samples\": [\n          540,\n          379,\n          114\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Survived\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Pclass\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 1,\n        \"max\": 3,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          3,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Name\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 849,\n        \"samples\": [\n
\"Frolicher, Miss. Hedwig Margaritha\",\n          \"Betros, Mr.
Tannous\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sex\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Age\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 12.96006886134744,\n        \"min\": 0.42,\n        \"max\":
80.0,\n        \"num_unique_values\": 88,\n        \"samples\": [\n
64.0,\n          22.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SibSp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 8,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Parch\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 6,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Ticket\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 665,\n        \"samples\": [\n
\"113043\",\n          \"244361\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Fare\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 22.962214421541105,\n
\"min\": 0.0,\n        \"max\": 113.275,\n
\"num_unique_values\": 232,\n        \"samples\": [\n
7.8875,\n          23.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Embarked\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


Separating features & Target

```
X = df.drop(columns =
['PassengerId','Name','Ticket','Survived'],axis=1)
Y = df['Survived']

print(X)
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	0	22.000000	1	0	7.2500	0
1	1	1	38.000000	1	0	71.2833	1
2	3	1	26.000000	0	0	7.9250	0
3	1	1	35.000000	1	0	53.1000	0
4	3	0	35.000000	0	0	8.0500	0
..	...	...	...	...	...	...	...
886	2	0	27.000000	0	0	13.0000	0
887	1	1	19.000000	0	0	30.0000	0
888	3	1	29.699118	1	2	23.4500	0
889	1	0	26.000000	0	0	30.0000	1
890	3	0	32.000000	0	0	7.7500	2

```
[849 rows x 7 columns]

print(Y)

0      0
1      1
2      1
3      1
4      0
      ..
886    0
```

```
887    1
888    0
889    1
890    0
Name: Survived, Length: 849, dtype: int64
```

Splitting the data into training data & Test data

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
test_size=0.2, random_state=2)

print(X.shape, X_train.shape, X_test.shape)
```

```
(849, 7) (679, 7) (170, 7)
```

Model Training

```
model = LogisticRegression()

# training the Logistic Regression model with training data
model.fit(X_train, Y_train)
```

```
LogisticRegression()
```

Model Evaluation

```
# accuracy on training data
X_train_prediction = model.predict(X_train)

print(X_train_prediction)
```

```
[0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 0 0 0 1
0 1
 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 0
1 0
 0 0 1 1 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
0 1
 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0
0 0
 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0
0 0
 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 1 0 0 0
0 0
 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1
```

```
1 0
 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 0 1
1 0
 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0
0 1
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0
 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1
0 0
 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 1 1
0 0
 1 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0
0 0
 0 1 0 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 1
0 1
 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0
0 0
 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0
0 0
 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0
0 0
 0 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 1
0 0
 0 1 0 0 0 0 0 1 0 0 0 1 1]
```

```
from sklearn.metrics import accuracy_score

training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8026509572901326
```

```
# accuracy on test data
X_test_prediction = model.predict(X_test)

print(X_test_prediction)
```

```
[0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 0 1 1
0 1
 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
1 0
 0 1 0 1 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0
0 1
 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 1 1
0 1
 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 1 0]
```

```
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.788235294117647
```