Experiment No. 2
Analyze the Titanic Survival Dataset and apply appropriate regression technique
Date of Performance:
Date of Submission:

Aim: Analyze the Titanic Survival Dataset and apply appropriate Regression Technique.

Objective: Able to perform various feature engineering tasks, apply logistic regression on the given dataset and maximize the accuracy.

Theory:

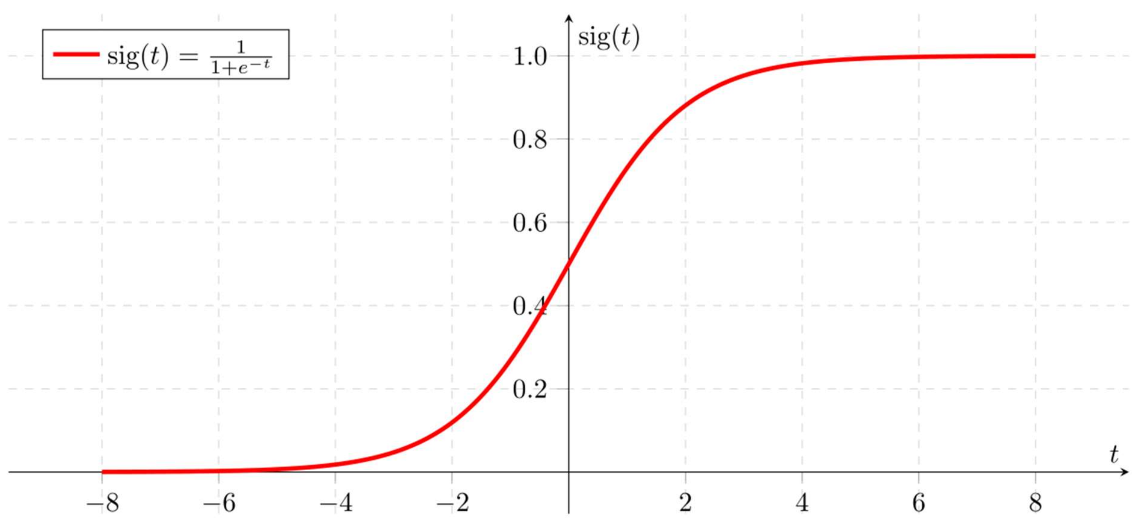
Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical and is binary in nature. In order to perform binary classification the logistic regression techniques makes use of Sigmoid function.

For example,

To predict whether an email is spam (1) or (0)

Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time.



From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

Dataset:

The sinking of the Titanic is one of the most infamous shipwrecks in history.

On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this challenge, we ask you to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

pclass: A proxy for socio-economic status (SES)

1st = Upper, 2nd = Middle, 3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way...,

Sibling = brother, sister, stepbrother, stepsister

Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way...

Parent = mother, father

Child = daughter, son, stepdaughter, stepson

Some children travelled only with a nanny, therefore parch=0 for them.

Conclusion:

1. Features selected for predicting survival in the Titanic dataset, encompassing 'gender', 'pclass', 'embarked', 'fare', 'sibsp', 'age', and 'parch', is well-justified due to their correlations with the target variable. 'Gender' plays a crucial role, reflecting historical priority in rescue efforts. 'Pclass' and 'fare' likely indicate socio-economic status, influencing access to resources during the crisis. 'Embarked' could indirectly signify factors like location or demographics. 'Sibsp', 'age', and 'parch' capture family dynamics and individual characteristics..

2. The accuracy of a classification model indicates the proportion of correctly predicted outcomes among all predictions. While a higher accuracy suggests better performance. Accuracy obtained is 85% with precision of 88% , recall of 86% ,and f1 score of 87%.Type 1 error is 15 and Type II error is 12.

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df=sns.load_dataset("titanic")

df.head()
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked
class  \
0         0       3    male  22.0      1      0   7.2500        S
Third
1         1       1  female  38.0      1      0  71.2833        C
First
2         1       3  female  26.0      0      0   7.9250        S
Third
3         1       1  female  35.0      1      0  53.1000        S
First
4         0       3    male  35.0      0      0   8.0500        S
Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

```
df.describe().T
```

```
          count       mean        std   min      25%      50%   75%
max
survived  891.0   0.383838   0.486592  0.00   0.0000   0.0000   1.0
1.0000
pclass    891.0   2.308642   0.836071  1.00   2.0000   3.0000   3.0
3.0000
age       714.0  29.699118  14.526497  0.42  20.1250  28.0000  38.0
80.0000
sibsp     891.0   0.523008   1.102743  0.00   0.0000   0.0000   1.0
8.0000
parch     891.0   0.381594   0.806057  0.00   0.0000   0.0000   0.0
6.0000
fare      891.0  32.204208  49.693429  0.00   7.9104  14.4542  31.0
512.3292
```

```
val=df.corr()['survived'].sort_values(ascending= True).values
```

```
<ipython-input-74-34287dac2f0f>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
```

```
of numeric_only to silence this warning.
  val=df.corr()['survived'].sort_values(ascending= True).values
```

```
val
```

```
array([-0.55708004, -0.33848104, -0.20336709, -0.07722109, -
0.0353225 ,
        0.08162941,  0.25730652,  1.        ])
```

```
series = pd.Series(val)
series
```

```
0   -0.557080
1   -0.338481
2   -0.203367
3   -0.077221
4   -0.035322
5    0.081629
6    0.257307
7    1.000000
dtype: float64
```

```
series = series.abs()
series.sort_values(ascending=False)
```

```
7    1.000000
0    0.557080
1    0.338481
6    0.257307
2    0.203367
5    0.081629
3    0.077221
4    0.035322
dtype: float64
```

```
df.corr()['survived'].sort_values(ascending= True)
```

```
<ipython-input-78-8ca4d6e58f13>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  df.corr()['survived'].sort_values(ascending= True)
```

```
adult_male   -0.557080
pclass       -0.338481
alone        -0.203367
age          -0.077221
sibsp        -0.035322
parch         0.081629
fare          0.257307
```

```
survived      1.000000
Name: survived, dtype: float64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

```
df.isnull().sum()
```

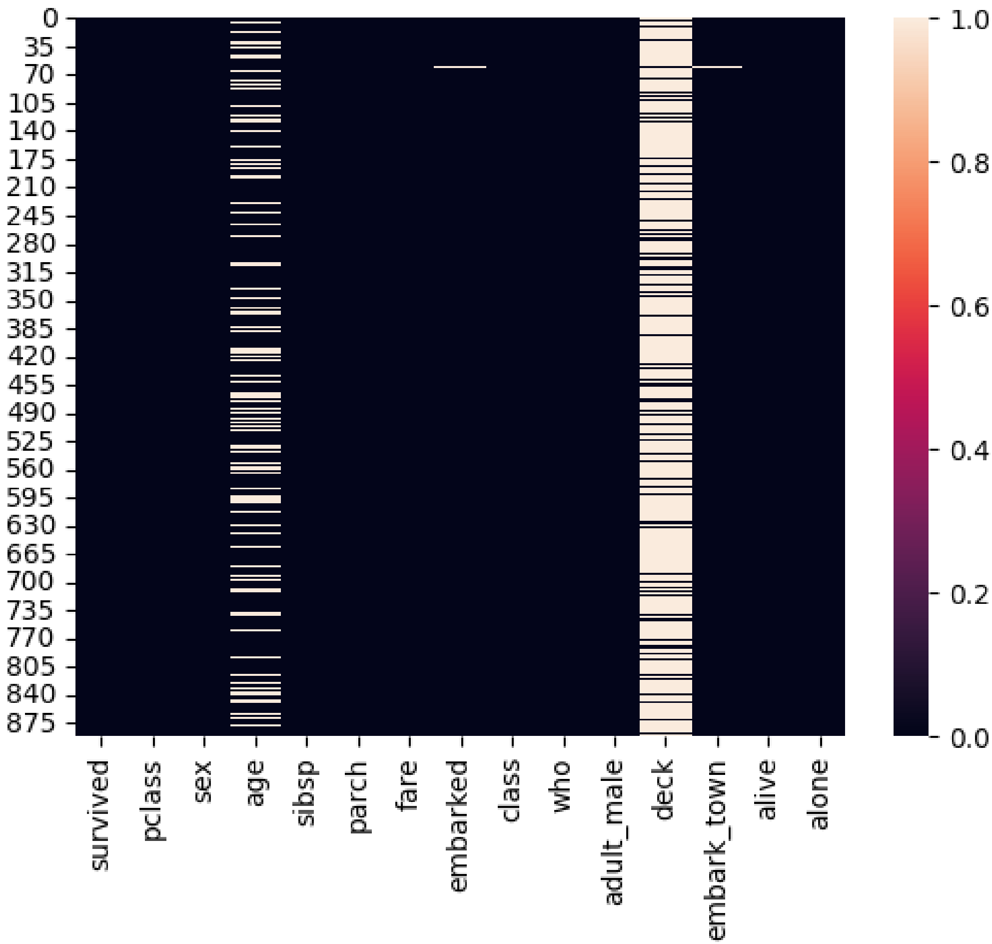
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

```
df.shape
```

```
(891, 15)
```

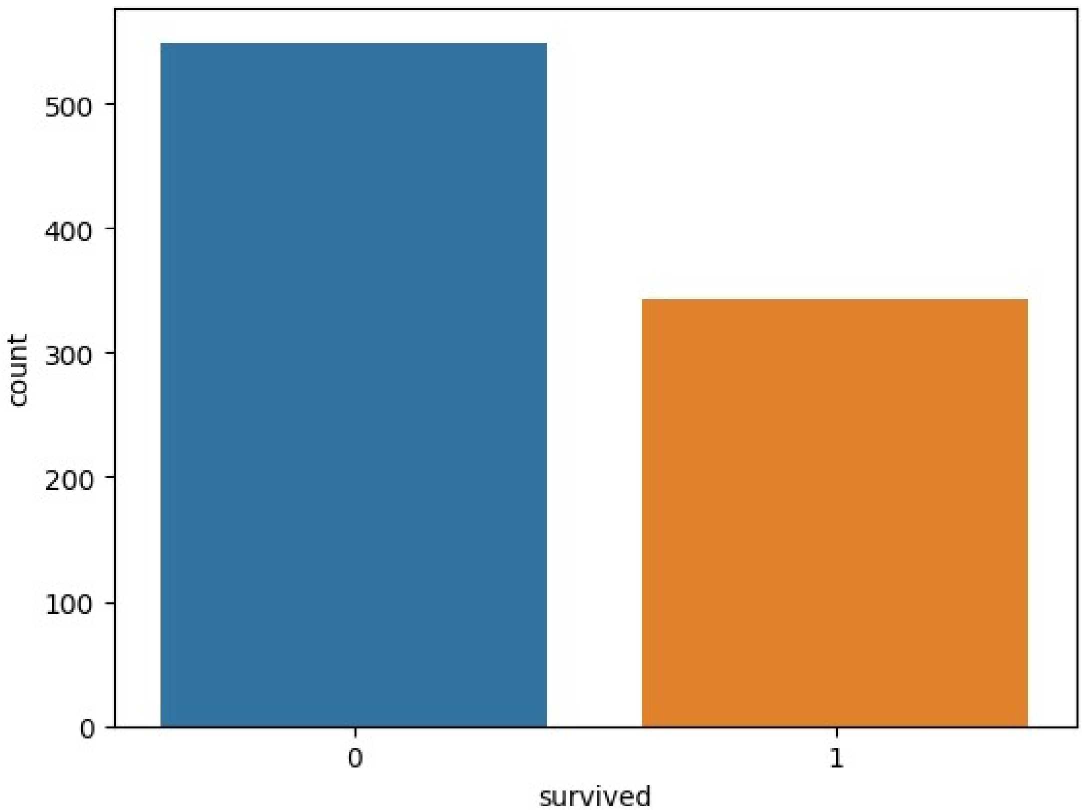
```
sns.heatmap(df.isnull())
```

```
<Axes: >
```



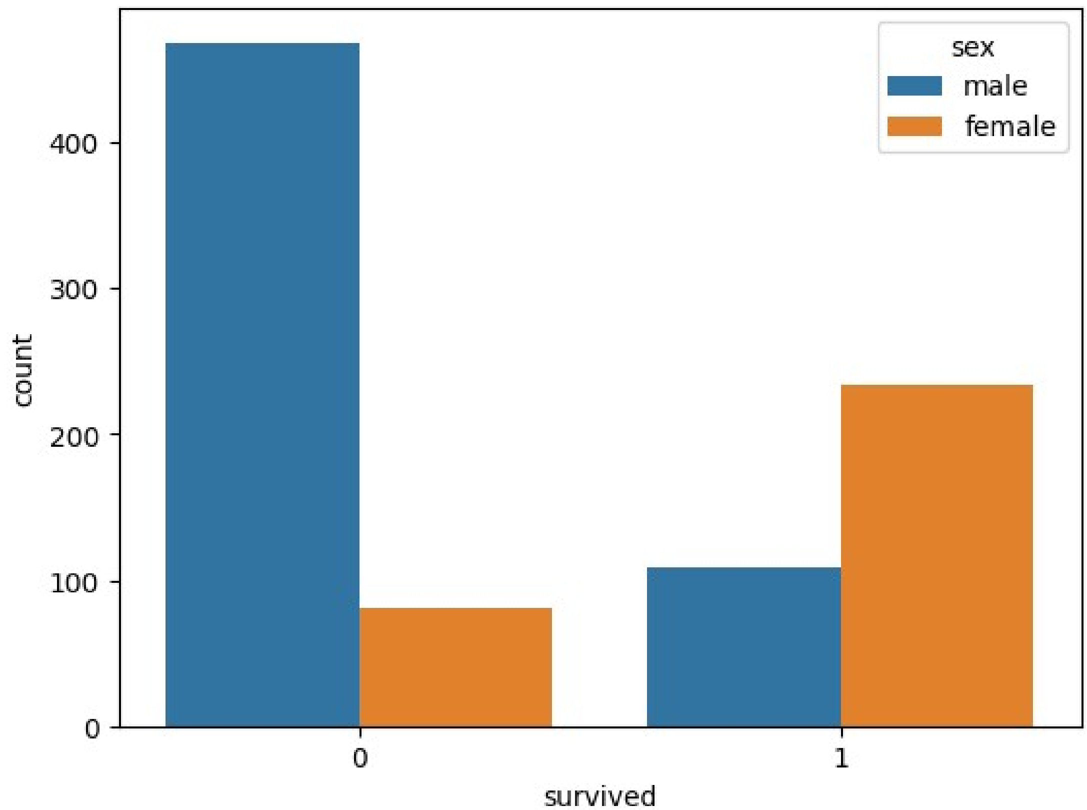

```
sns.countplot(data=df , x=df['survived'])
```

```
<Axes: xlabel='survived', ylabel='count'>
```



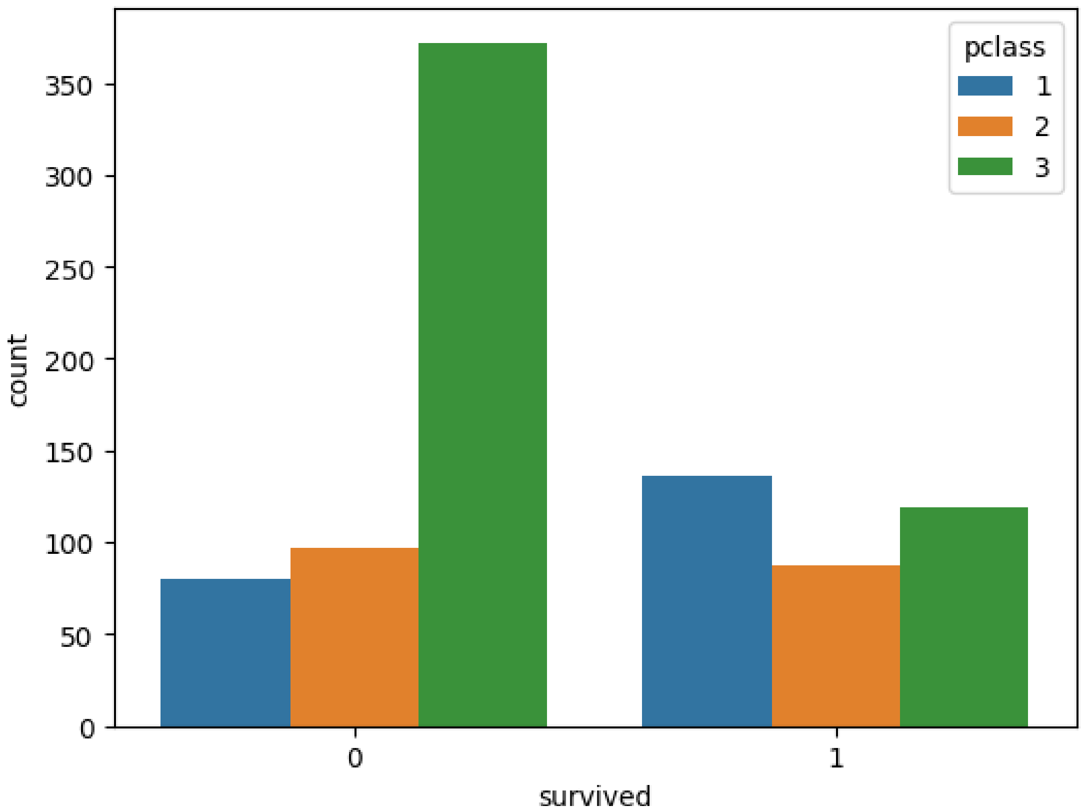

```
sns.countplot(data=df , x=df['survived'] , hue=df['sex'])
```

```
<Axes: xlabel='survived', ylabel='count'>
```



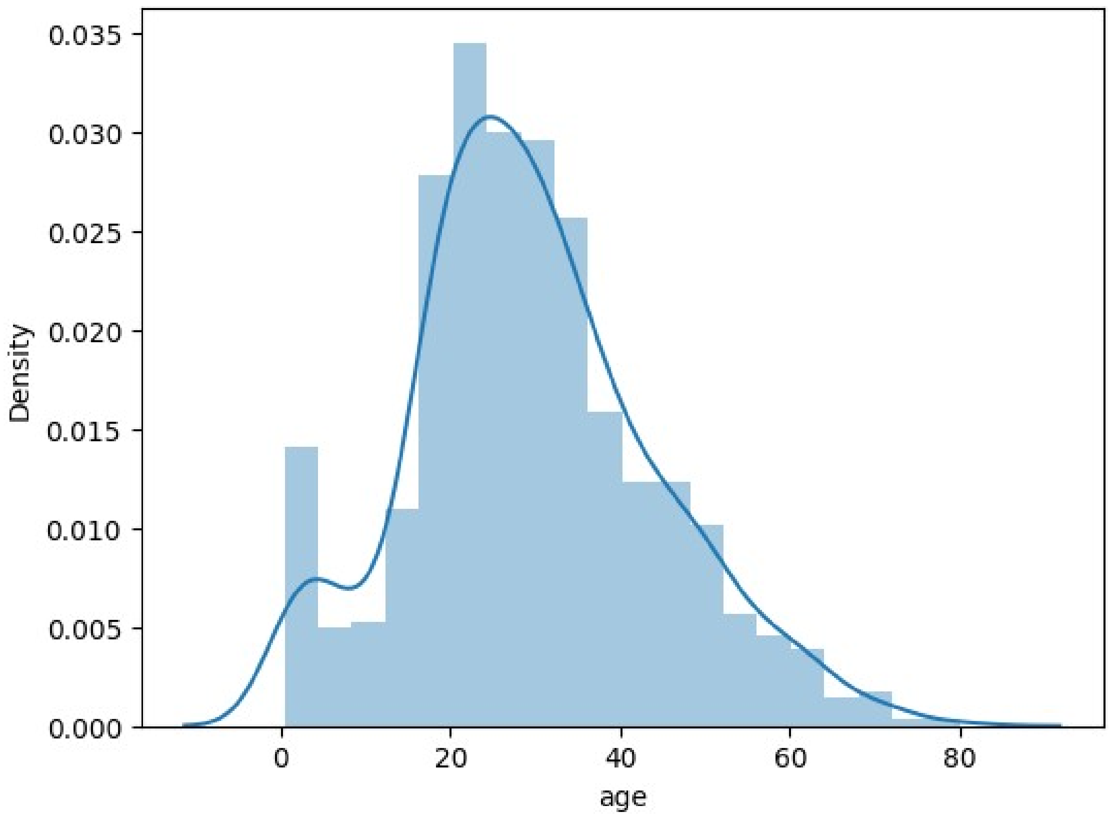

```
sns.countplot(data=df , x=df['survived'] , hue=df['pclass'])
```

```
<Axes: xlabel='survived', ylabel='count'>
```




```
sns.distplot(df['age'])
```

```
<ipython-input-86-7452d86f8334>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['age'])

<Axes: xlabel='age', ylabel='Density'>
```




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

```
df.isnull().sum()
```

```
survived        0
pclass          0
sex             0
age             0
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck          688
embark_town     2
alive           0
alone           0
dtype: int64
```

```
column_name = 'embarked'
df = df.dropna(subset=[column_name], axis=0)
```

```
df['sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
gender=pd.get_dummies(df['sex'] , drop_first=True)
```

```
df['gender']=gender
```

```
columns=['alive' , 'alone' , 'embark_town' , 'who' , 'adult_male' ,
'deck']
df.drop(columns , axis=1 , inplace=True)
```

```
df.head()
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked
class  \
0         0       3    male  22.0      1      0   7.2500        S
Third
1         1       1  female  38.0      1      0  71.2833        C
First
2         1       3  female  26.0      0      0   7.9250        S
Third
3         1       1  female  35.0      1      0  53.1000        S
First
4         0       3    male  35.0      0      0   8.0500        S
Third

   gender
0       1
1       0
2       0
3       0
4       1
```

```
df.drop('sex'  , axis=1 , inplace=True)
```

```
df.head()
```

```
   survived  pclass   age  sibsp  parch     fare embarked  class
gender
0         0       3  22.0      1      0   7.2500        S  Third
1
1         1       1  38.0      1      0  71.2833        C  First
0
2         1       3  26.0      0      0   7.9250        S  Third
0
3         1       1  35.0      1      0  53.1000        S  First
0
4         0       3  35.0      0      0   8.0500        S  Third
1
```

```
df['embarked'].unique()
```

```
array(['S', 'C', 'Q'], dtype=object)
```

```
df.dropna(axis = 1 , inplace=True)

df['class'].unique()

['Third', 'First', 'Second']
Categories (3, object): ['First', 'Second', 'Third']

mapping = {'First': 1, 'Second': 2, 'Third': 3}
df['class'] = df['class'].replace(mapping)

df.head()

   survived  pclass   age  sibsp  parch     fare embarked class
gender
0         0       3  22.0      1      0   7.2500        S     3
1
1         1       1  38.0      1      0  71.2833        C     1
0
2         1       3  26.0      0      0   7.9250        S     3
0
3         1       1  35.0      1      0  53.1000        S     1
0
4         0       3  35.0      0      0   8.0500        S     3
1

df['embarked'].unique()

array(['S', 'C', 'Q'], dtype=object)

mapping = {'S': 1, 'C': 2, 'Q': 3}
df['embarked'] = df['embarked'].replace(mapping)

df.head()

   survived  pclass   age  sibsp  parch     fare  embarked class
gender
0         0       3  22.0      1      0   7.2500         1     3
1
1         1       1  38.0      1      0  71.2833         2     1
0
2         1       3  26.0      0      0   7.9250         1     3
0
3         1       1  35.0      1      0  53.1000         1     1
0
4         0       3  35.0      0      0   8.0500         1     3
1

df.shape

(889, 9)
```

```
# x=df.iloc[: , 1:]
x=df.iloc[: , [-1 , 1 , -3 ,-4 , 4 , 2 , 3]]
# x=df.iloc[: , [4 ,-3 , -4 , 3]]
y=df.iloc[: , 0]

x
```

```
     gender  pclass  embarked     fare  parch        age  sibsp
0         1       3         1   7.2500      0  22.000000      1
1         0       1         2  71.2833      0  38.000000      1
2         0       3         1   7.9250      0  26.000000      0
3         0       1         1  53.1000      0  35.000000      1
4         1       3         1   8.0500      0  35.000000      0
..      ...     ...       ...      ...    ...        ...    ...
886       1       2         1  13.0000      0  27.000000      0
887       0       1         1  30.0000      0  19.000000      0
888       0       3         1  23.4500      2  29.699118      1
889       1       1         2  30.0000      0  26.000000      0
890       1       3         3   7.7500      0  32.000000      0

[889 rows x 7 columns]
```

```
y
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: survived, Length: 889, dtype: int64
```

```
from sklearn.model_selection import train_test_split

X_train , X_test , Y_train , Y_test = train_test_split(x , y ,
test_size=0.2 , random_state=1)

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

```
(711, 7) (178, 7)
```

```
from sklearn.linear_model import LogisticRegression

Log_Reg=LogisticRegression()

Log_Reg.fit(X_train ,Y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression()
```

```
y_pred=Log_Reg.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix , classification_report
```

```
confusion_matrix(Y_test , y_pred)
```

```
array([[90, 15],
       [12, 61]])
```

```
print(classification_report(Y_test , y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       105
           1       0.80      0.84      0.82        73

    accuracy                           0.85       178
   macro avg       0.84      0.85      0.84       178
weighted avg       0.85      0.85      0.85       178
```