**Logistic Regression** is a statistical method used for binary classification problems in machine learning. It predicts the probability of a binary outcome (e.g., yes/no, 0/1) based on one or more predictor variables.

**Key Concepts**

1. **Binary Classification**: Logistic regression is used when the target variable has two possible outcomes. For example, predicting whether an email is spam or not spam.
2. **Logistic Function**: Logistic regression uses the logistic function (also called the sigmoid function) to model the probability of the target variable. The function maps any real-valued number into a value between 0 and 1.

The logistic function is defined as:

sigmoid(z)=11+e−z\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}sigmoid(z)=1+e−z1​

where zzz is a linear combination of the input features.

1. **Linear Combination**: The model calculates a weighted sum of the input features and applies the logistic function to predict the probability of the positive class.

**Example**

Let’s say you want to predict whether a student will pass an exam based on their hours of study. Here’s a simple example with one feature (hours of study).

**Steps:**

1. **Collect Data**: Gather data with features (hours of study) and the binary target (pass/fail).

| **Hours of Study** | **Passed Exam (1)** |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |

1. **Create a Model**:
   * Compute the logistic regression coefficients (weights) using the training data.
   * The model equation will be: Probability of Passing=11+e−(β0+β1⋅Hours)\text{Probability of Passing} = \frac{1}{1 + e^{-(\beta\_0 + \beta\_1 \cdot \text{Hours})}}Probability of Passing=1+e−(β0​+β1​⋅Hours)1​

where β0\beta\_0β0​ and β1\beta\_1β1​ are the model parameters.

1. **Predict**: For a new value (e.g., 4 hours of study), use the model to predict the probability of passing.

**LOGISTIC REGRESSION**

from google.colab import files

uploaded = files.upload()

import pandas as pd

import io

titanic\_data= pd.read\_csv(io.BytesIO(uploaded['titanic\_dataset.csv']))

print(titanic\_data)

output:

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 Gender 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]

titanic\_data.info()

titanic\_data.isna().sum() // to check null and summing

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 Gender 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

PassengerId0

Survived0

Pclass0

Name0

Gender0

Age 177

SibSp0

Parch0

Ticket0

Fare0

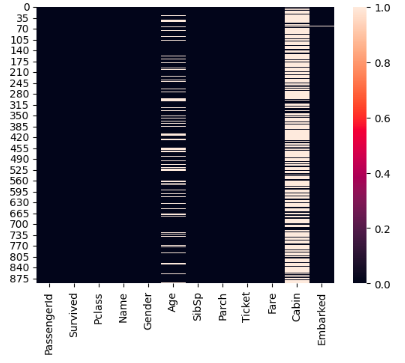
Cabin 687

Embarked 2

Now checking using heat map

Import seaborn as sea

sea.heatmap(titanic\_data.isna())



Note : you can see lot of white spaces in age and cabin. This denotes that null value is present.

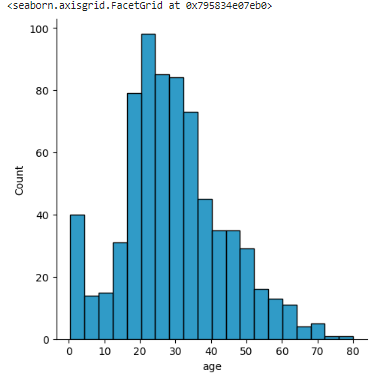
**// changing all column to small case**

titanic\_data.columns=[cols.lower() for cols in titanic\_data.columns]

titanic\_data.info()

**// to see distribution plot**

**sea**.displot(x=’age’, data=titanic\_data)



// to display mean age

mean\_age= titanic\_data['age'].mean()

print(mean\_age)

output:

29.69911764705882

**// to fill mean age in NA**

titanic\_data['age'].fillna(mean\_age, inplace=True)

// displaying with distribution plot and displaying titanic data

sea.displot(x='age', data=titanic\_data)

titanic\_data.info()

**output:**

<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 gender 891 non-null object

5 age 891 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

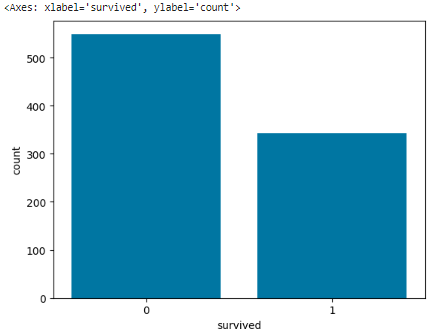
dtypes: float64(2), int64(5), object(5)

memory usage: 83.7+ KB

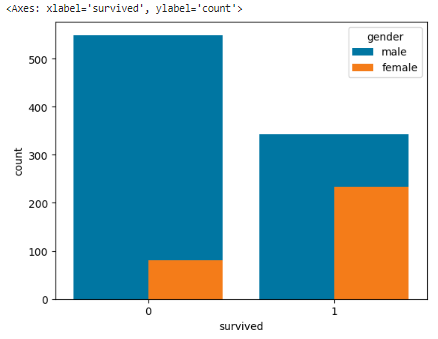
**you can see age is filled with mean value.**

**// to check how many surived**

**sea.countplot(x=survived, data= titanic\_data)**



**sea.countplot(x=survived, data= titanic\_data,hue=’gender’))**



// Now changing male to 1 and female to 2 in gender and displaying gender value

titanic\_data.replace({'gender':{'male':1,'female':2}},inplace=True)

print(titanic\_data['gender'])

output:

0 1

1 2

2 2

3 2

4 1

..

886 1

887 2

888 2

889 1

890 1

Name: gender, Length: 891, dtype: int64

// now we are going to drop certain columns and put it in input data and output data

input\_data= titanic\_data.drop(columns=['name','survived','ticket','cabin','embarked'])

output\_data=titanic\_data['survived']

print(input\_data)

passengerid pclass gender age sibsp parch fare

0 1 3 1 22.000000 1 0 7.2500

1 2 1 2 38.000000 1 0 71.2833

2 3 3 2 26.000000 0 0 7.9250

3 4 1 2 35.000000 1 0 53.1000

4 5 3 1 35.000000 0 0 8.0500

.. ... ... ... ... ... ... ...

886 887 2 1 27.000000 0 0 13.0000

887 888 1 2 19.000000 0 0 30.0000

888 889 3 2 29.699118 1 2 23.4500

889 890 1 1 26.000000 0 0 30.0000

890 891 3 1 32.000000 0 0 7.7500

[891 rows x 7 columns]

Note: The above data value is not standardized as it has more variation so we are using **standard scaler** to standardize the value and printing the input\_data

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

input\_data=scaler.fit\_transform(input\_data)

print(input\_data)

**ouput:**

[[-1.73010796 0.82737724 -0.73769513 ... 0.43279337 -0.47367361

-0.50244517]

[-1.72622007 -1.56610693 1.35557354 ... 0.43279337 -0.47367361

0.78684529]

[-1.72233219 0.82737724 1.35557354 ... -0.4745452 -0.47367361

-0.48885426]

...

[ 1.72233219 0.82737724 1.35557354 ... 0.43279337 2.00893337

-0.17626324]

[ 1.72622007 -1.56610693 -0.73769513 ... -0.4745452 -0.47367361

-0.04438104]

[ 1.73010796 0.82737724 -0.73769513 ... -0.4745452 -0.47367361

-0.49237783]]

**// to check how many columns present in input and output data**

print(input\_data.shape)

print(output\_data.shape)

**output:**

(891, 7)

(891,)

**// for training and spliting**

from sklearn.model\_selection import train\_test\_split // for training and spliting

input\_data\_train,input\_data\_test,output\_data\_train, output\_data\_test=train\_test\_split(input\_data,output\_data,test\_size=0.3)

print(input\_data\_train.shape)

print(output\_data\_train.shape)

print(input\_data\_test.shape)

print(output\_data\_test.shape)

Output:

(623, 7)

(623,)

(268, 7)

(268,)

// importing Logistic Regression algorithm

from sklearn.linear\_model import LogisticRegression

model= LogisticRegression()

model.fit(input\_data\_train,output\_data\_train)

**// Predicting out input\_data\_test with logistic model**

predicted\_survival=model.predict(input\_data\_test)

//now machine learning has predicted.

// Now checking machine learning model prediction with output\_data

// using confusion matrix for checking

from sklearn.metrics import confusion\_matrix

confusion\_matrix(output\_data\_test, predicted\_survival)

output:

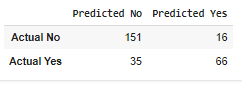
array([[151, 16],

[ 35, 66]])

//printing the confusion matrix with label and index

pd.DataFrame(confusion\_matrix(output\_data\_test,predicted\_survival),columns=['Predicted No', 'Predicted Yes'],index=['Actual No','Actual Yes'])

**output:**



// Printing accuracy

from sklearn.metrics import accuracy\_score

accuracy\_info=accuracy\_score(output\_data\_test,predicted\_survival)

accuracy\_info

0.8097014925373134

// it is giving 80 % accuracy

// After increasing test size it is 2% increased

0.8268156424581006