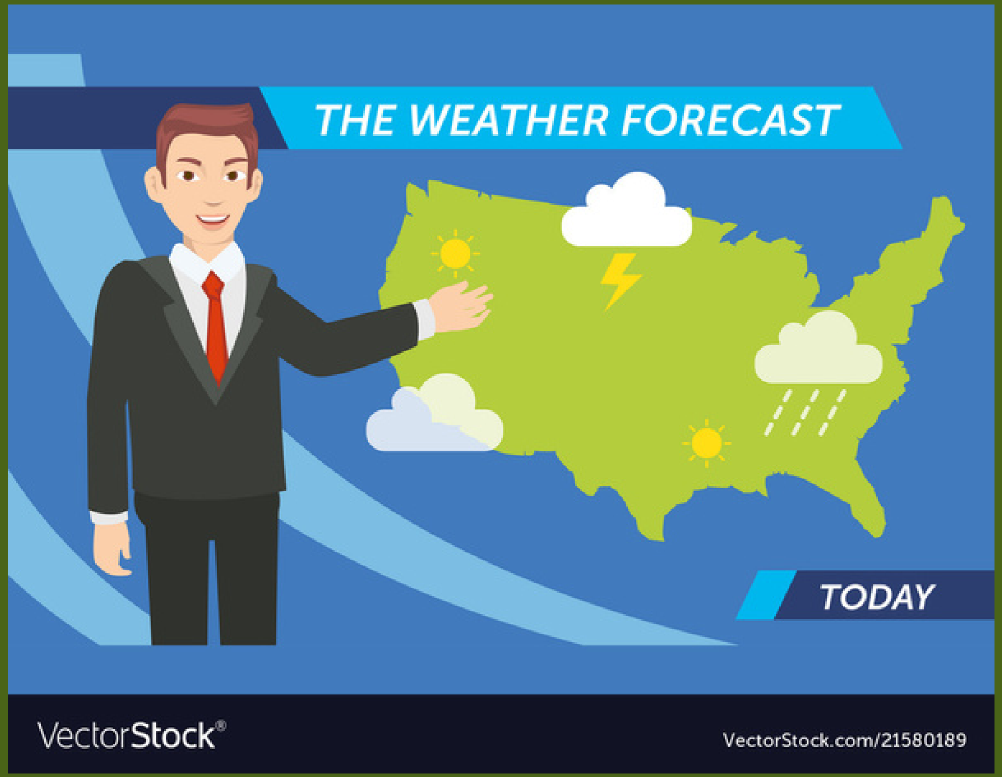
Linear & Logistic Regression

Regression



Weather forecasting

Sales Revenue

Housing Price

Others -
Life Expectancy,
Crop production,
Insurance Claim , etc.

Questions -

1. What will be the temperature tomorrow.
2. What will be the house price based on essential factors.
3. What is a prediction of future Sales revenue?
4. What is the relationship between drug dosage and blood pressure of patients.

ANSWER -

Regression

What is Regression?

A statistical measure that attempts to see how strongly one dependent variable is correlated with a number of other variables that are constantly changing (independent variable).

Predict the value of a dependent variable (Y) based on the value of an independent variable (X1, X2, ….).

Regression is widely used with a continuous quantity for prediction and forecasting purpose.

Dependent Variable (Y) & Independent Variable (X)

- The dependent variable is what you want to use the model to explain or predict.
- The values of this variable depend on other variables. It is the outcome that you're studying.
- It's also known as the response variable, outcome variable, and left-hand variable. Statisticians commonly denote them using a Y.

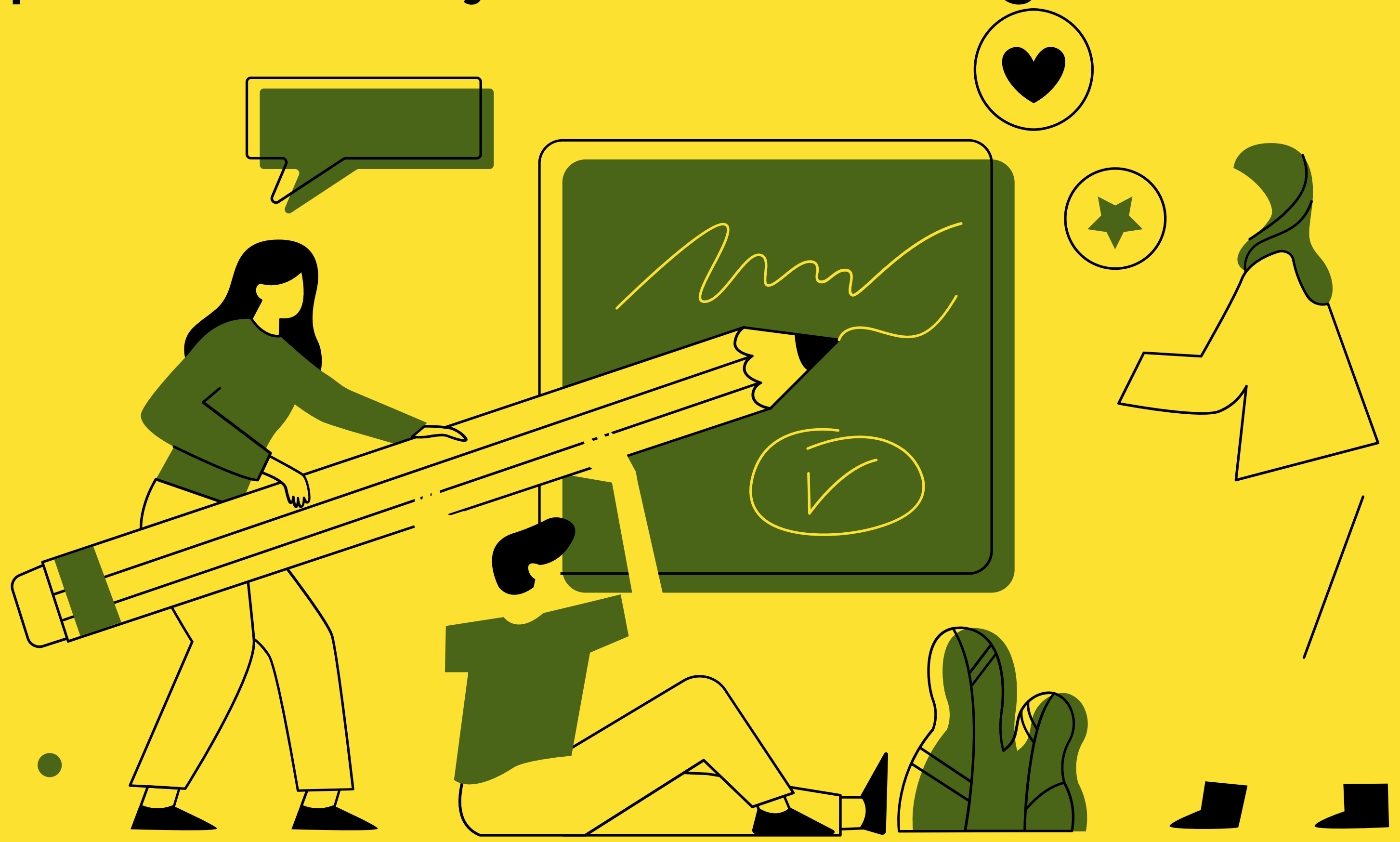
- Independent variables are the ones that we used to predict the response variable (dependent variable).
- These variables are independent.
- They stand alone and other variables in the model do not influence them.
- Independent variables are also known as predictors, input variables, X-variables, and right-hand variables—because they appear on the right side of the equals sign in a regression equation.

Types Of Regression

- Linear Regression
- Logistic Regression

Linear Regression

Linear regression is a method to predict dependent variable(Y) based on values of independent variable (X). We use it where we want to predict some continuous quantity.
Linear regression is a popular technique for predictive analysis and modelling.

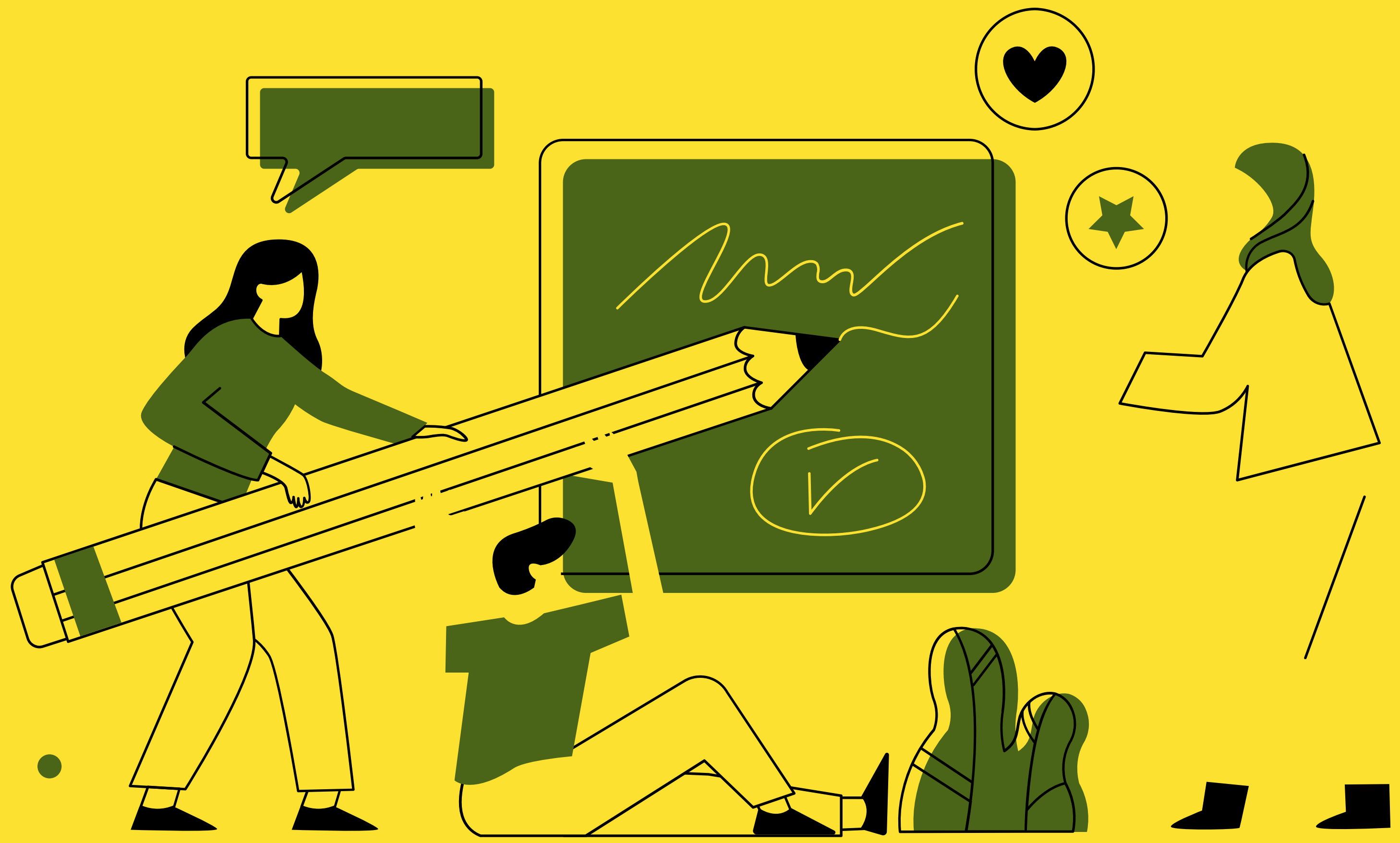


Examples -

- The weight of the person is linearly related to their height.
- The effect of fertilizer and water on crop yields.
- The effect that different training regimens have on player performance.
- Analyzing insurance claim is higher in older age people.

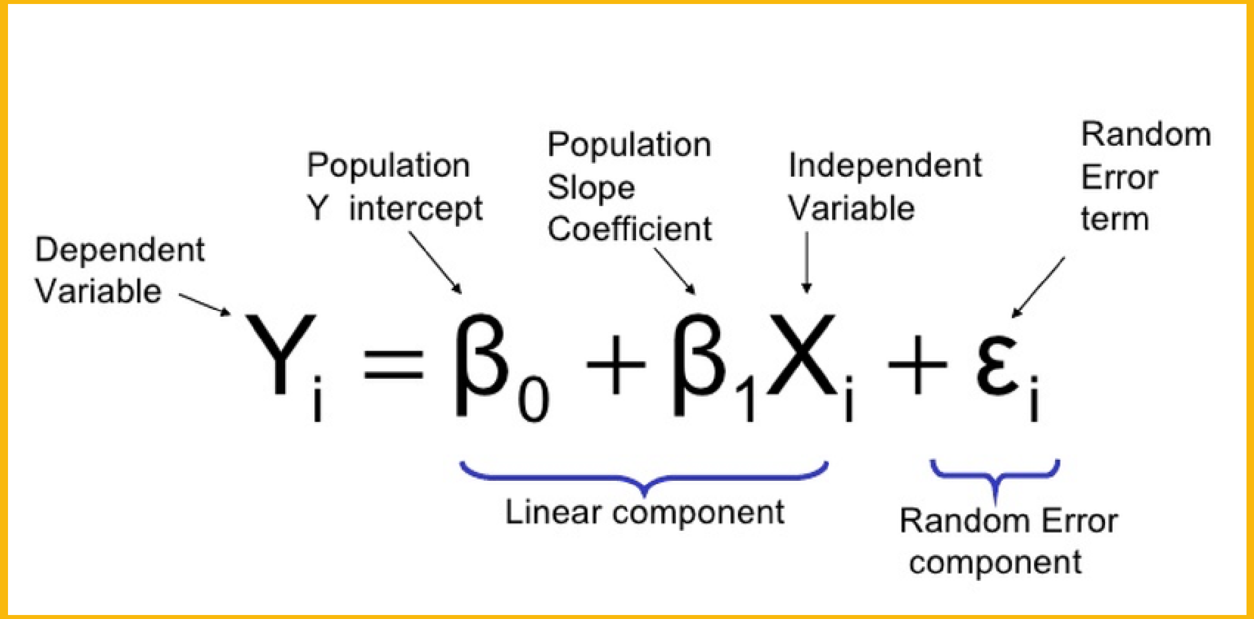
etc.

When will I use Linear Regression ?



- **Linear relationship:** Relationship between response and feature variables should be linear.
- **Little or no multi-collinearity:** Multicollinearity occurs when the features (or independent variables) are not independent of each other.
- **Little or no auto-correlation:** Autocorrelation occurs when the residual errors are not independent of each other.
- **Homoscedastic :** It refers to a condition in which the error term, in a regression model is constant.

Formula



$$\underbrace{Y_i}_{\text{}} = \underbrace{\beta_0 + \beta_1 X_i}_{\text{Linear component}} + \underbrace{\varepsilon_i}_{\text{Random Error component}}$$

Dependent Variable: Y_i

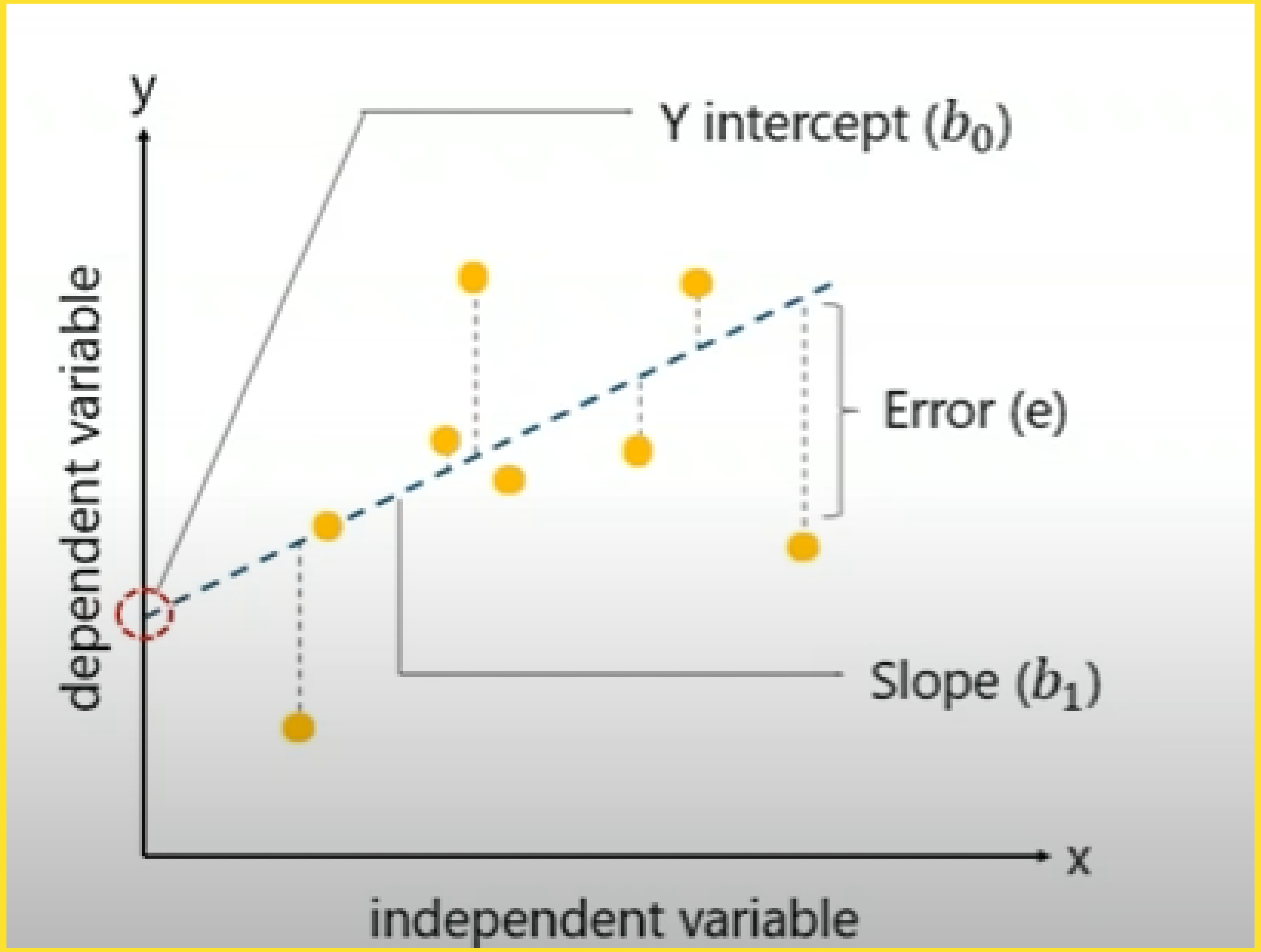
Population Y intercept: β_0

Population Slope Coefficient: β_1

Independent Variable: X_i

Random Error term: ε_i

Graph Understanding



Mathematical Implementation

01

02

03

04

Step 1

Get the data in X - table and Y- table

Step 2

Find the value of X^2, & Y^2

Step 3

From the table find -Σx , Σy, Σxy, Σx^2, Σy^2

Step 4

Find this

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma x y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma x y) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

Finally put it in this -

$$Y = a + bX$$

Python Implementation

01

Choose dataset and perform data preprocessing

Before implementing linear regression or any modal , we first look into our dataset.

In the real world, data is usually incomplete:

- **it lacks different values,**
- **it has mistakes, or**

it only contains aggregate data.

Steps in Data Preprocessing

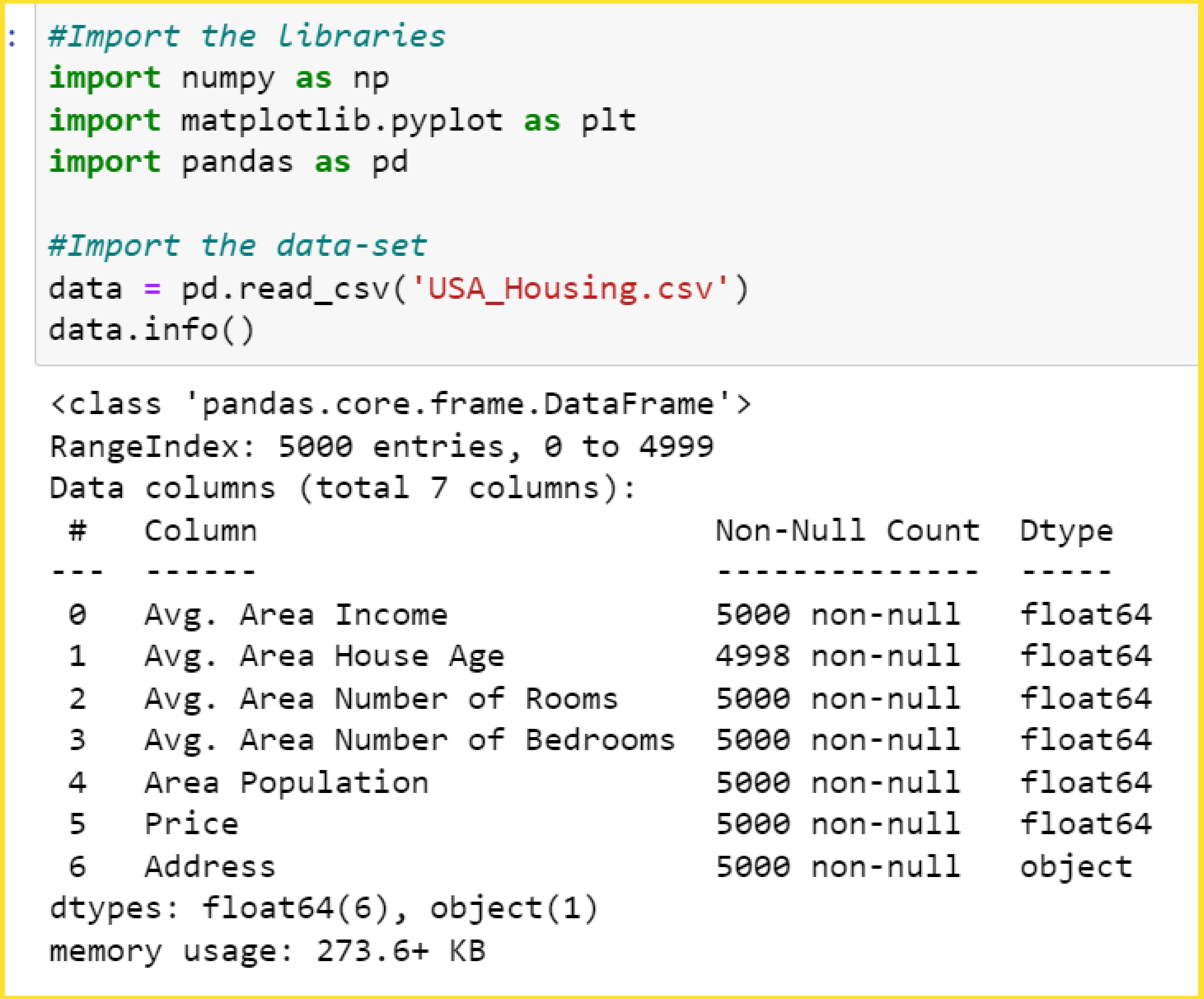
Step 1: Import the libraries
Step 2: Import the data-set
Step 3: Check out the missing values
Step 4: Encode the Categorical data
Step 5: Prepare for Splitting the dataset

Data preprocessing is a proven method of resolving such issues.

We are trying to make a model who can predict HOUSE PRICING .

We have taken USA_Housing data in csv format and imported required libraries .

Here we have 2 missing value in Avg. Area House Age



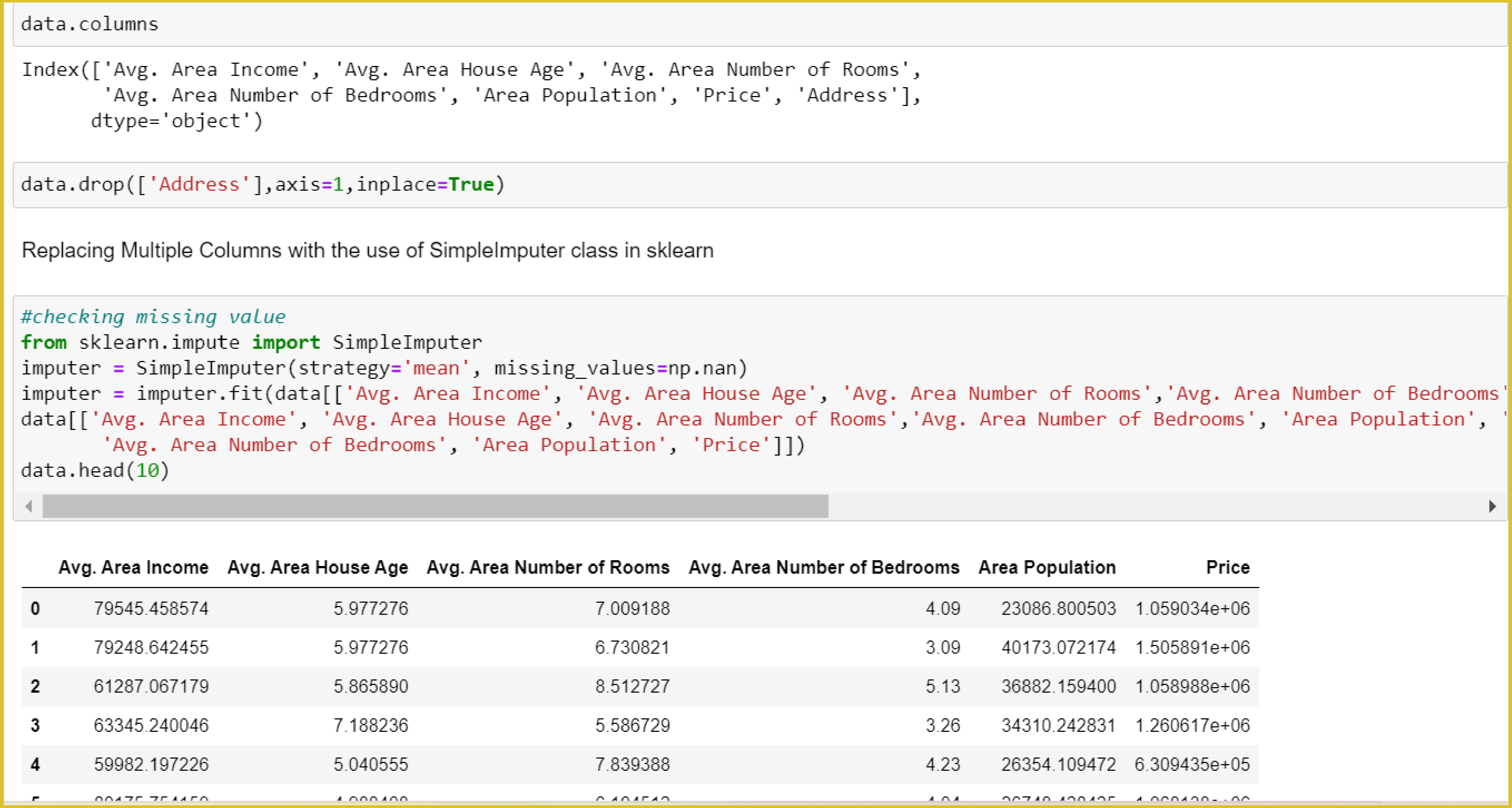
```
#Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Import the data-set
data = pd.read_csv('USA_Housing.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           4998 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

We have dropped the 'Address' column because Simple Imputer doesn't work with non-numeric.

Here in Avg. Area House Age column first two values are the mean value.

We dropped 0 and replaced it with mean value.



```
data.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

```
data.drop(['Address'],axis=1,inplace=True)
```

Replacing Multiple Columns with the use of SimpleImputer class in sklearn

```
#checking missing value
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
imputer = imputer.fit(data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area Number of Bedrooms'
data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area Number of Bedrooms', 'Area Population', '
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price']])
data.head(10)
```

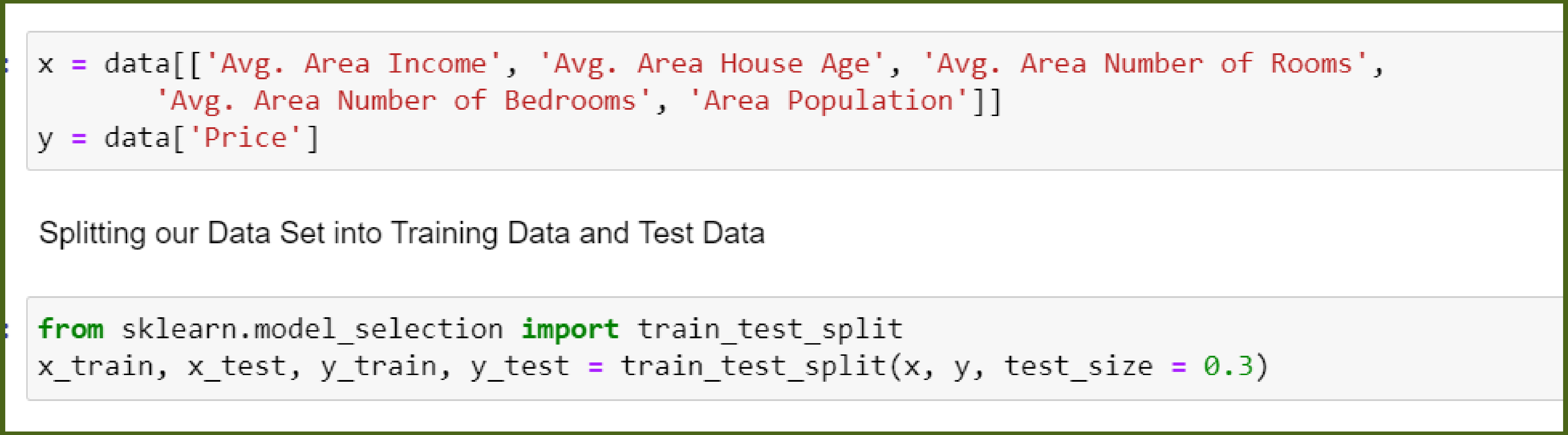
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
0	79545.458574	5.977276	7.009188	4.09	23086.800503	1.059034e+06
1	79248.642455	5.977276	6.730821	3.09	40173.072174	1.505891e+06
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05

02

Splitting our Data Set into Training Data and Test Data

Split our data into an x-array (which contains the data that we will use to make predictions) and a y-array (which contains the data that we are trying to predict).

Training data is the initial dataset you use to teach a machine learning application to recognize patterns or perform to your criteria, while testing data is used to evaluate your model's accuracy.



```
x = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
y = data['Price']
```

Splitting our Data Set into Training Data and Test Data

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)
```

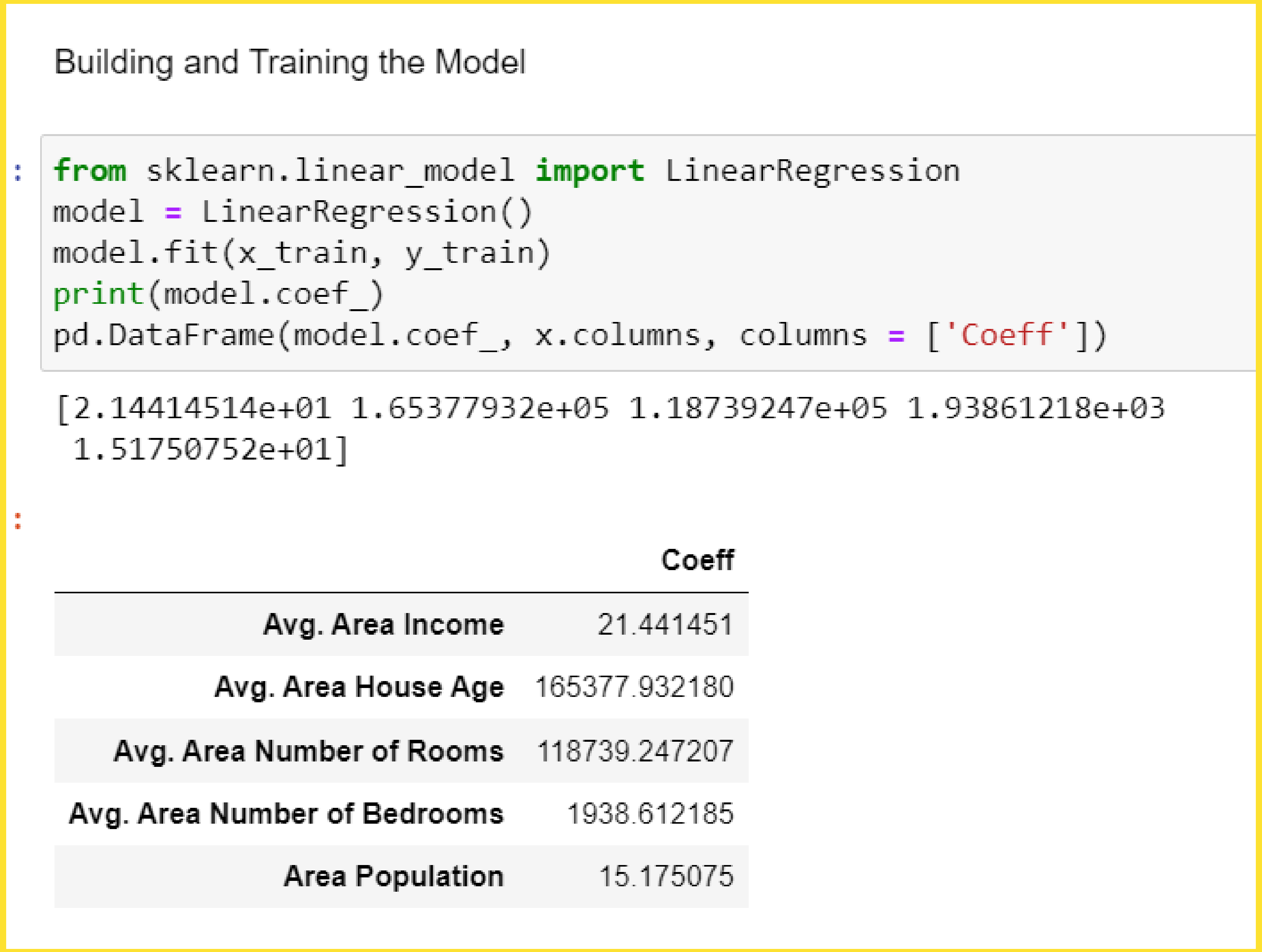
Scikit-learn (Sklearn) is most efficient library for statistical modeling including classification, regression, clustering and dimensionality reduction in Python.

03

Building And Training Model

- **First import the Linear Regression estimator from scikit-learn**
- **set it to a variable called model**
- **use scikit-learn’s fit method to train this model on our training data.**

A nicer way to view the coefficients is by placing them in a DataFrame.



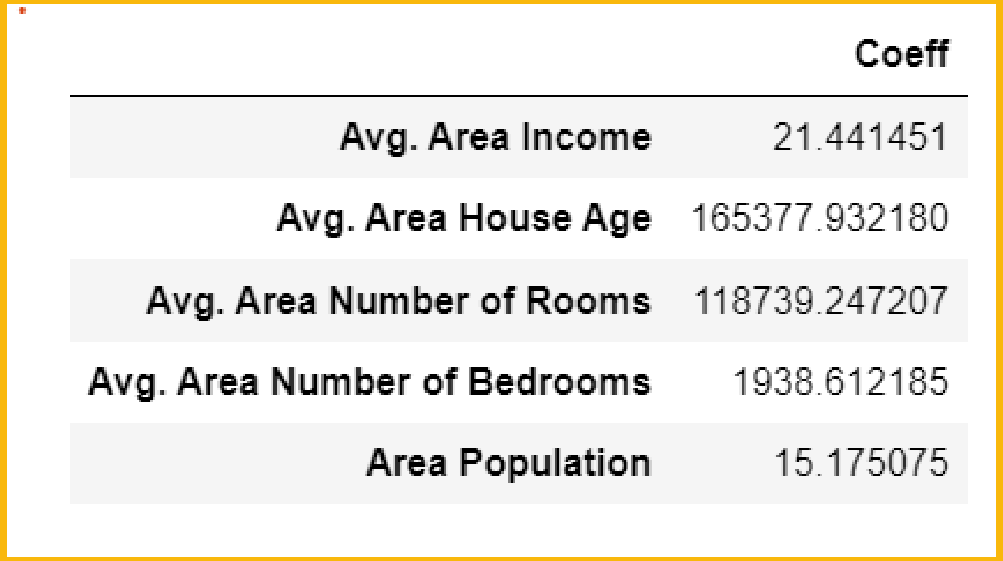
Building and Training the Model

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
print(model.coef_)
pd.DataFrame(model.coef_, x.columns, columns = ['Coeff'])
```

```
[2.14414514e+01 1.65377932e+05 1.18739247e+05 1.93861218e+03
 1.51750752e+01]
```

	Coeff
Avg. Area Income	21.441451
Avg. Area House Age	165377.932180
Avg. Area Number of Rooms	118739.247207
Avg. Area Number of Bedrooms	1938.612185
Area Population	15.175075

Output Interpretation



	Coeff
Avg. Area Income	21.441451
Avg. Area House Age	165377.932180
Avg. Area Number of Rooms	118739.247207
Avg. Area Number of Bedrooms	1938.612185
Area Population	15.175075

You can easily describe the coefficients like this:

"For every one-unit increase in [X variable], the [y variable] increases by [coefficient] when all other variables are held constant."

Area Population variable which has a coefficient of approximately 15.

That means if we hold all other variables constant, then a one-unit increase in Area Population will result in a 15-unit increase in the predicted variable - in this case, Price.

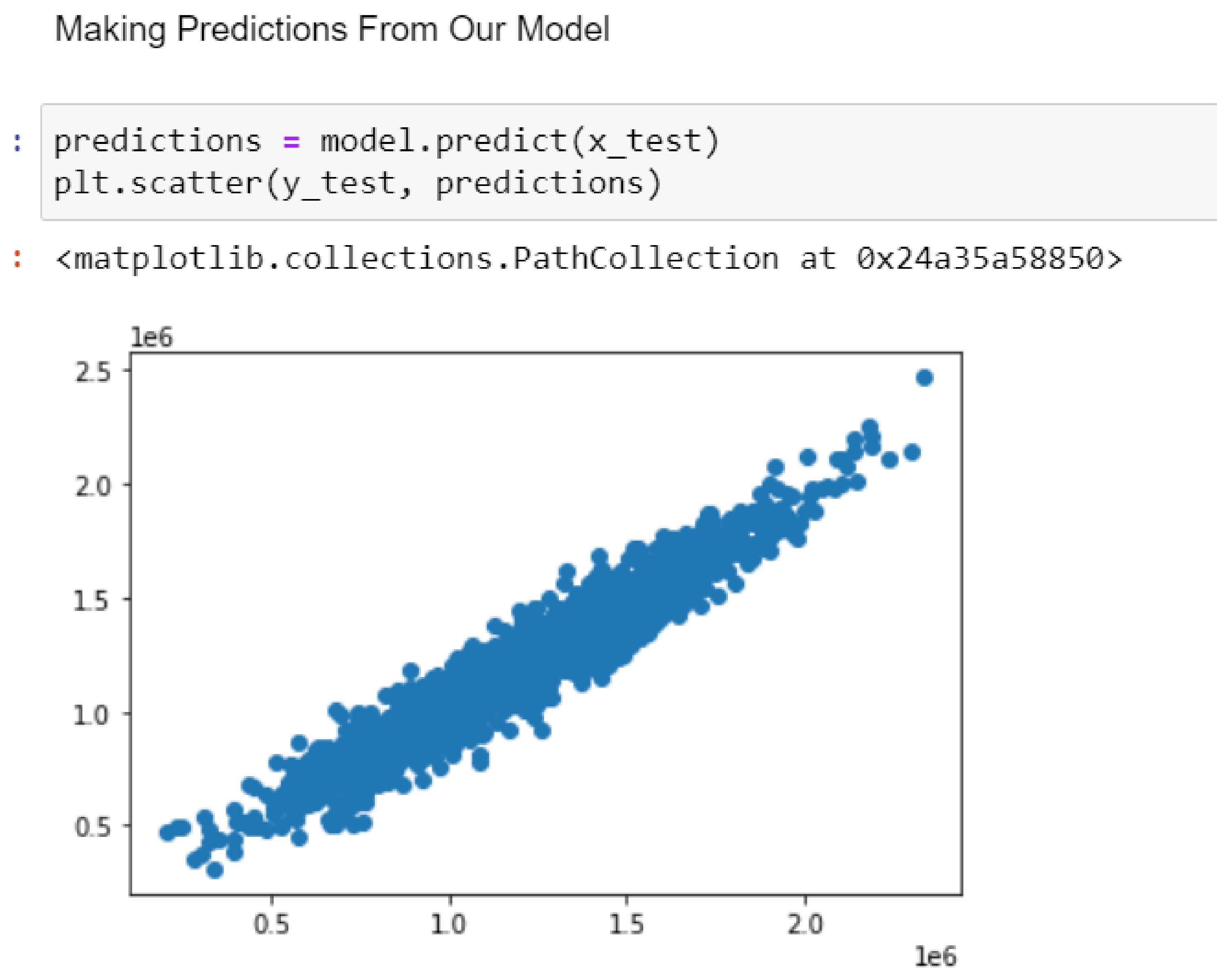
In another word, large coefficients on a specific variable mean that that variable has a large impact on the value of the variable we're trying to predict.

Similarly, small values have a small impact.

04 Making Predictions From Our Model

- **We call the predict method on the model variable that we created earlier**
- **Since the predict variable is designed to make predictions, it only accepts an x-array parameter.**
- **It will generate the y values for us.**

As you can see, our predicted values are very close to the actual values for the observations in the data set. A perfectly straight diagonal line in this scatterplot would indicate that our model perfectly predicted the y-array values.



Making Predictions From Our Model

```
predictions = model.predict(x_test)
plt.scatter(y_test, predictions)
```

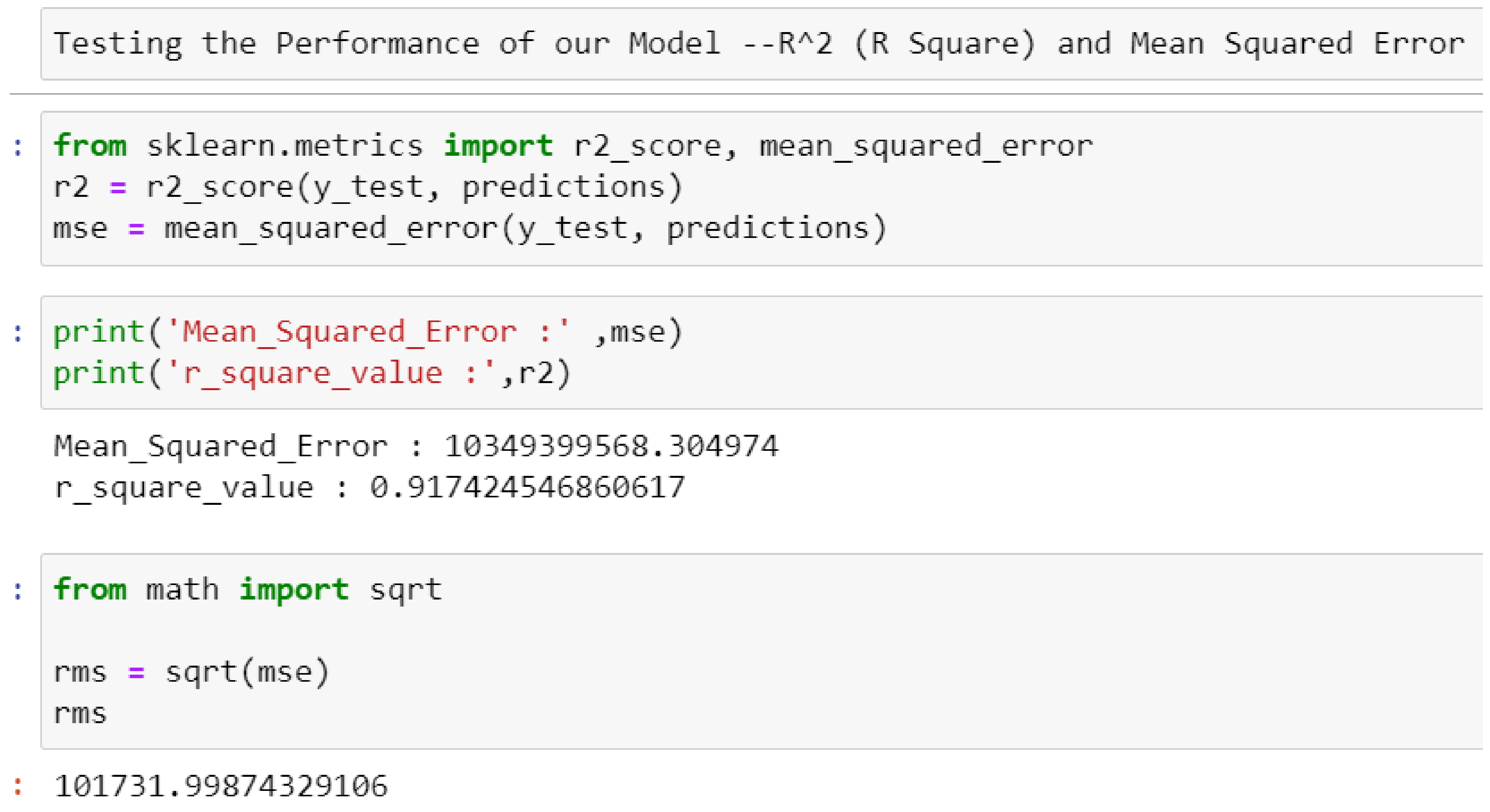
```
<matplotlib.collections.PathCollection at 0x24a35a58850>
```

05 Testing the Performance of our Model

To evaluate the accuracy of the model we use

1. **R-Squared**
2. **Mean absolute error**
3. **Mean squared error**
4. **Root mean squared error**

if all values are low ,its good for our model



```
Testing the Performance of our Model --R^2 (R Square) and Mean Squared Error
```

```
: from sklearn.metrics import r2_score, mean_squared_error
  r2 = r2_score(y_test, predictions)
  mse = mean_squared_error(y_test, predictions)
```

```
: print('Mean_Squared_Error :' ,mse)
  print('r_square_value :',r2)

  Mean_Squared_Error : 10349399568.304974
  r_square_value : 0.917424546860617
```

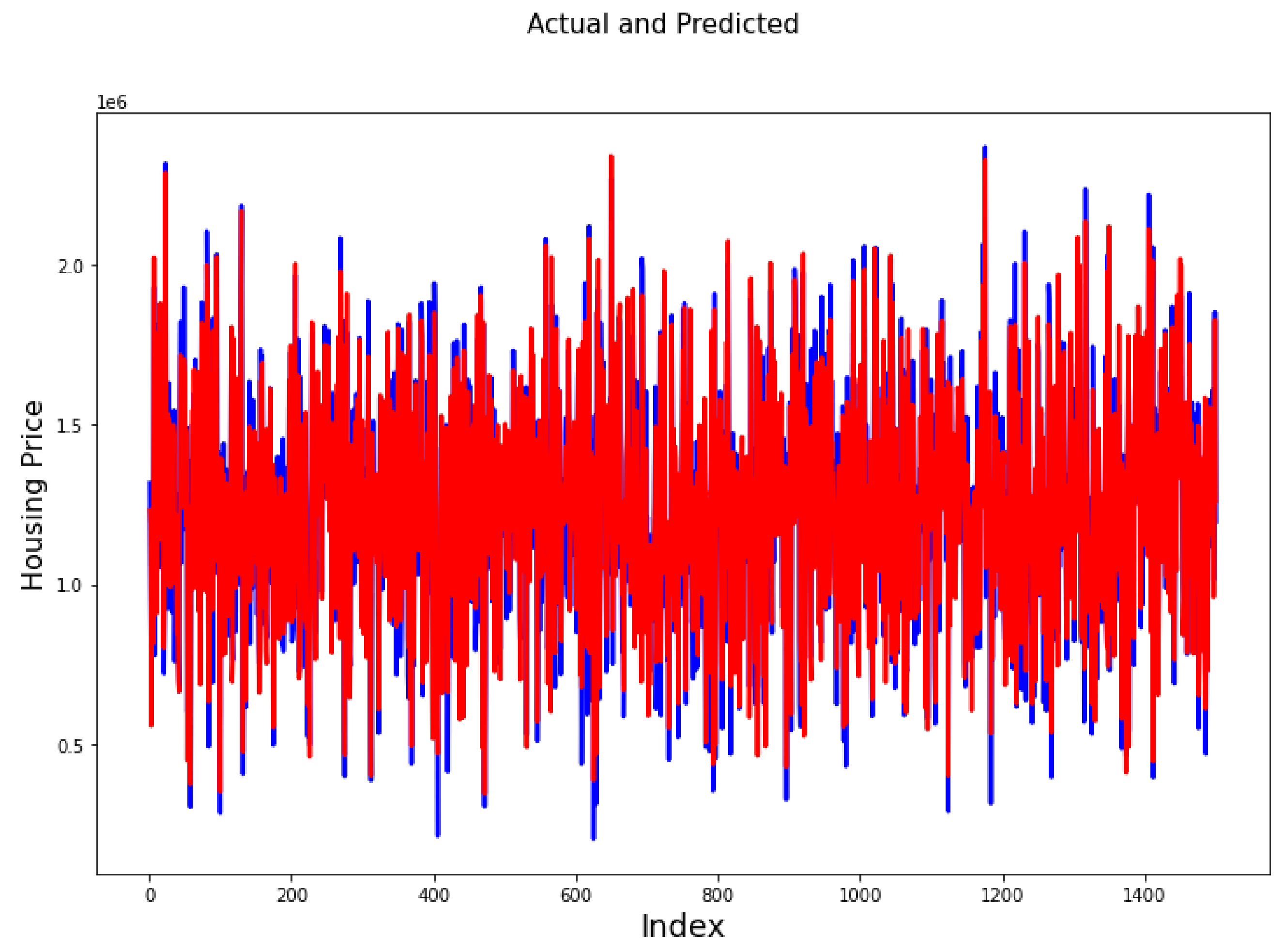
```
: from math import sqrt

  rms = sqrt(mse)
  rms

: 101731.99874329106
```

R^2 : Best possible score is 1.0 and here we got 0.91 .
So our modal works perfectly.

06

Difference Between Predicted and Actual Price (Accuracy Checking)

```
]: c = [i for i in range(1,1501,1)] # generating index
   fig = plt.figure(figsize=(12,8))
   plt.plot(c,y_test, color="blue", linewidth=2.5, linestyle="-") #Plotting Actual
   plt.plot(c,predictions, color="red",  linewidth=2.5, linestyle="-") #Plotting predicted
   fig.suptitle('Actual and Predicted', fontsize=15)              # Plot heading
   plt.xlabel('Index', fontsize=18)                               # X-label
   plt.ylabel('Housing Price', fontsize=16)

]: Text(0, 0.5, 'Housing Price')
```



R Square	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error
R-squared represents the proportion of the variance in the dependent variable which is explained by the linear regression model. It is a scale-free i.e. the value of R square will be less than one.	The Mean absolute error represents the average of the absolute difference between the actual and predicted values in the dataset. It measures the average of the residuals in the dataset.	Mean Squared Error represents the average of the squared difference between the original and predicted values in the data set. It measures the variance of the residuals.	Root Mean Squared Error is the square root of Mean Squared error. It measures the standard deviation of residuals.
A higher value of R square is considered desirable.	The lower value of MAE implies higher accuracy of a regression model.	The lower value of MSE, implies higher accuracy of a regression model.	The lower value of RMSE implies higher accuracy of a regression model.
$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$	$MAE = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}\rvert$ Where, $\hat{y}$ – predicted value of y $\bar{y}$ – mean value of y	$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$	$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$

Logistic Regression

Linear regression is a method to predict dependent variable(Y) , given a set of independent variable (X) such that the dependent variable is categorical.
Logistic regression is a popular technique for a prediction about a categorical variable

The output of a logistic regression model is in binary ('yes' or 'no' , 0 or 1).

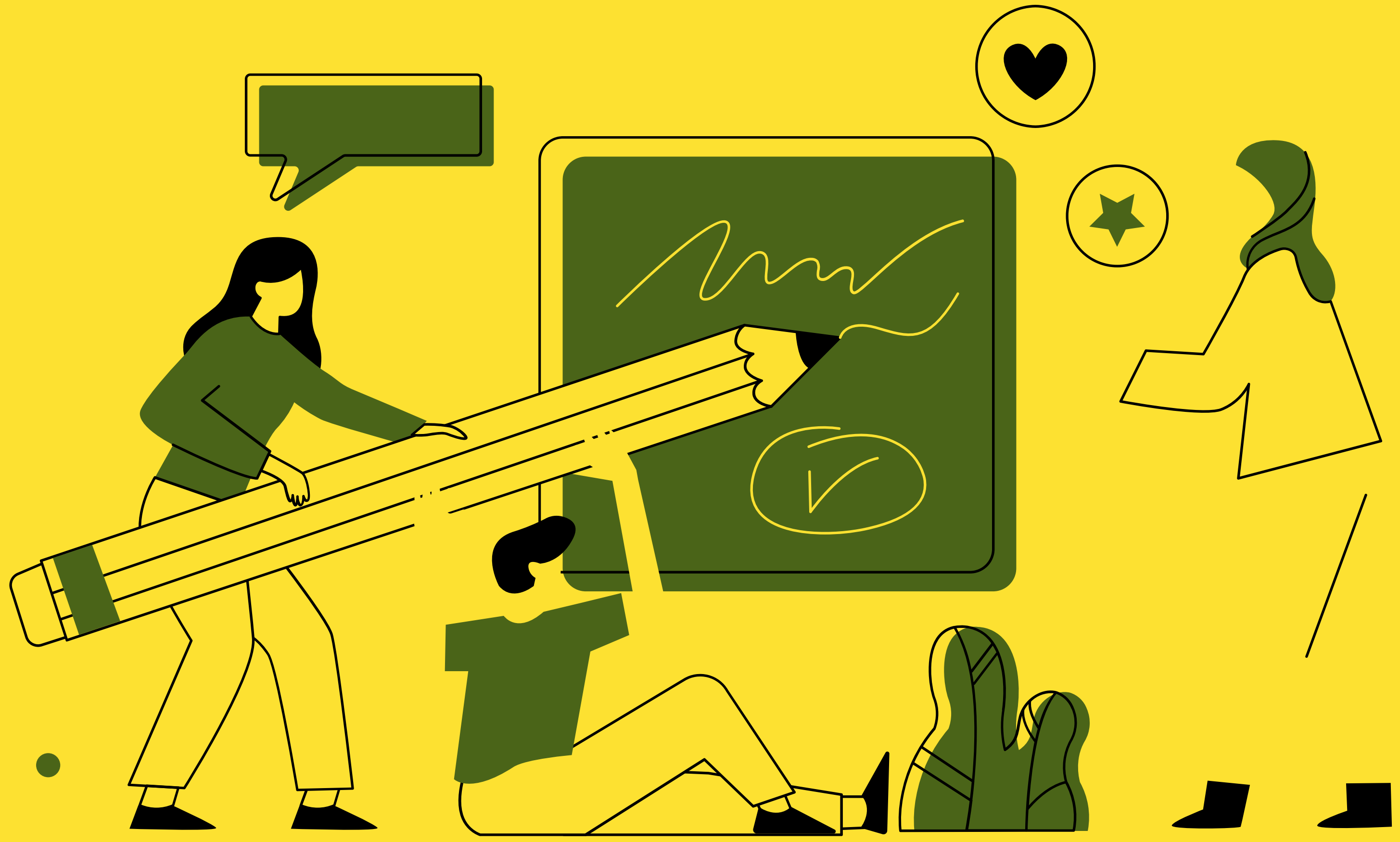
In another word the output is the probability of our input belonging to the class labeled with 1. And the complement of our model's output is the probability of our input belonging to the class labeled with 0.

Examples -

- To predict whether an email is spam (1) or (0)
- To predict whether the tumor is malignant (1) or not (0)
- To predict whether a political candidate will win or lose an election
- To predict whether a high school student will be admitted or not to a particular college.

etc.

When will I use Logistics Regression ?



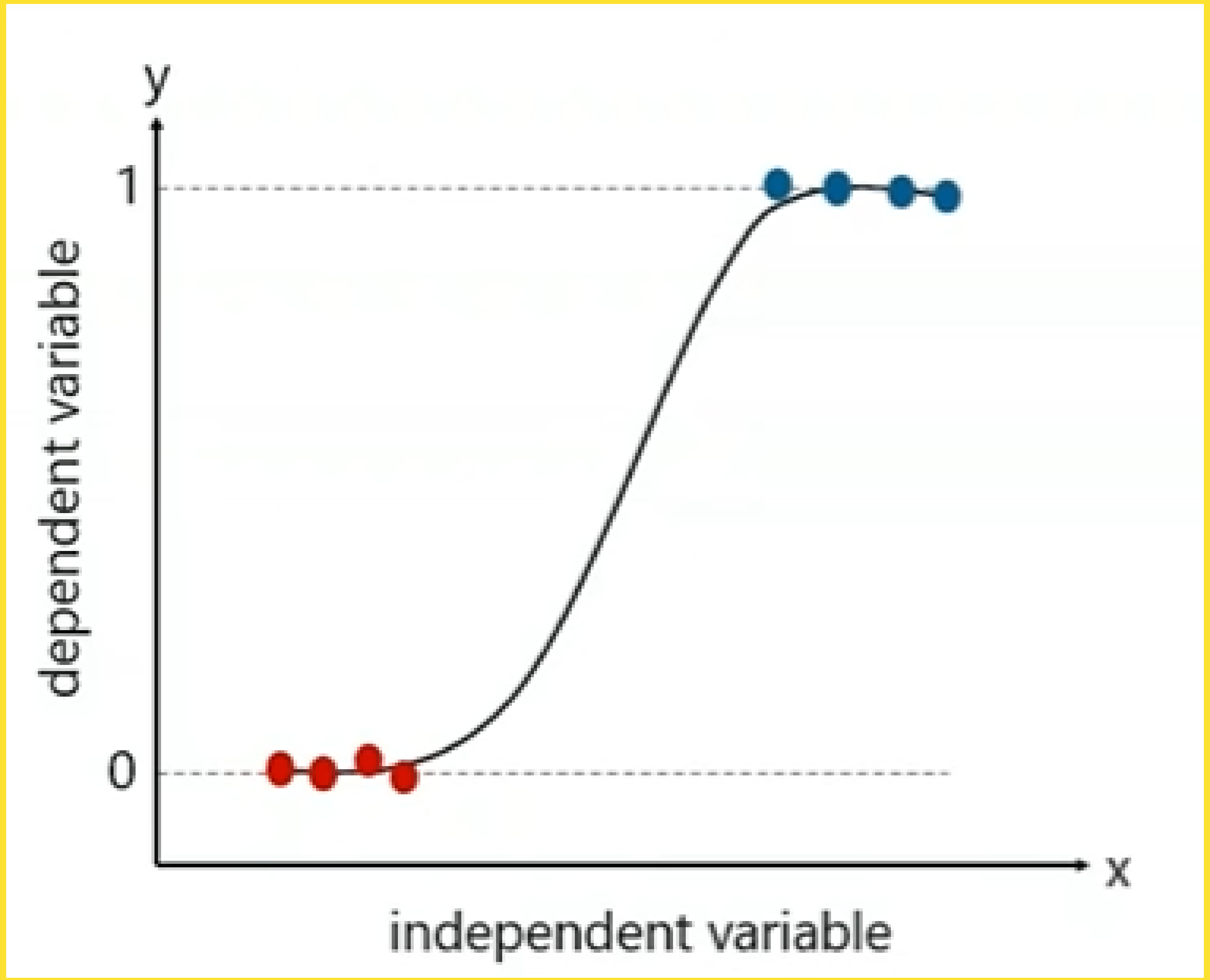
- Independence of errors - no room for errors
- Linearity in the logit (log-odds) - the outcome and each continuous independent variable is linear
- Absence of multicollinearity - (Multicollinearity happens when independent variables in the regression model are highly correlated to each other)
- Lack of strongly influential outliers (outlier means either much larger or much smaller data than all the other values)

Formula

$$\log\left(\frac{Y}{1-Y}\right) = C + B1X1 + B2X2 + \ldots.$$

- Y is the probability of an event to happen which you are trying to predict
- x1, x2 are the independent variables which determine the occurrence of an event i.e. Y
- C is the constant term which will be the probability of the event happening when no other factors are considered

Graph Understanding



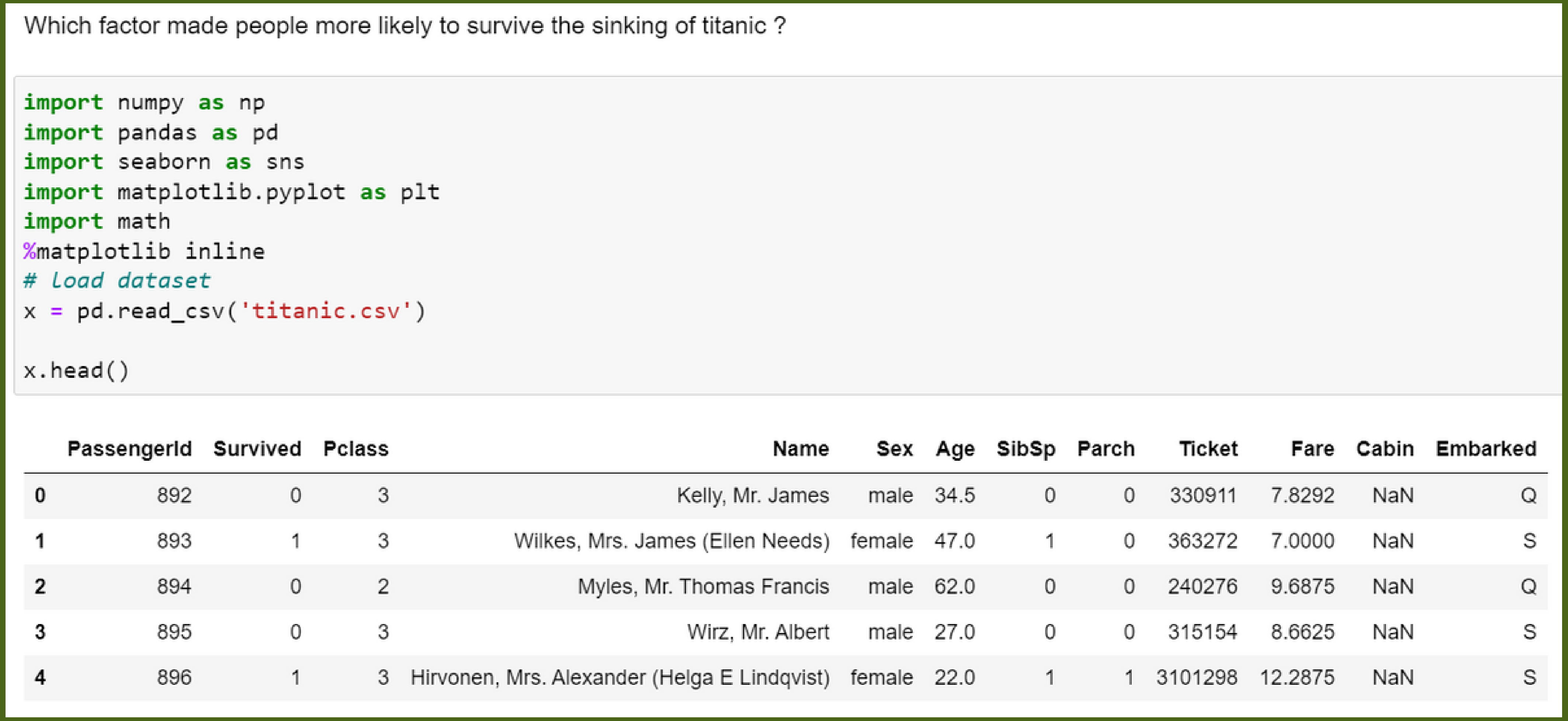
This Graph called Sigmoid Graph.

Python Implementation

01

Collecting Data - Importing libraries

We have taken Titanic data in csv format and imported required libraries .



Which factor made people more likely to survive the sinking of titanic ?

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
%matplotlib inline
# Load dataset
x = pd.read_csv('titanic.csv')

x.head()
```

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

Which factor made people more likely to survive the sinking of titanic ?

02

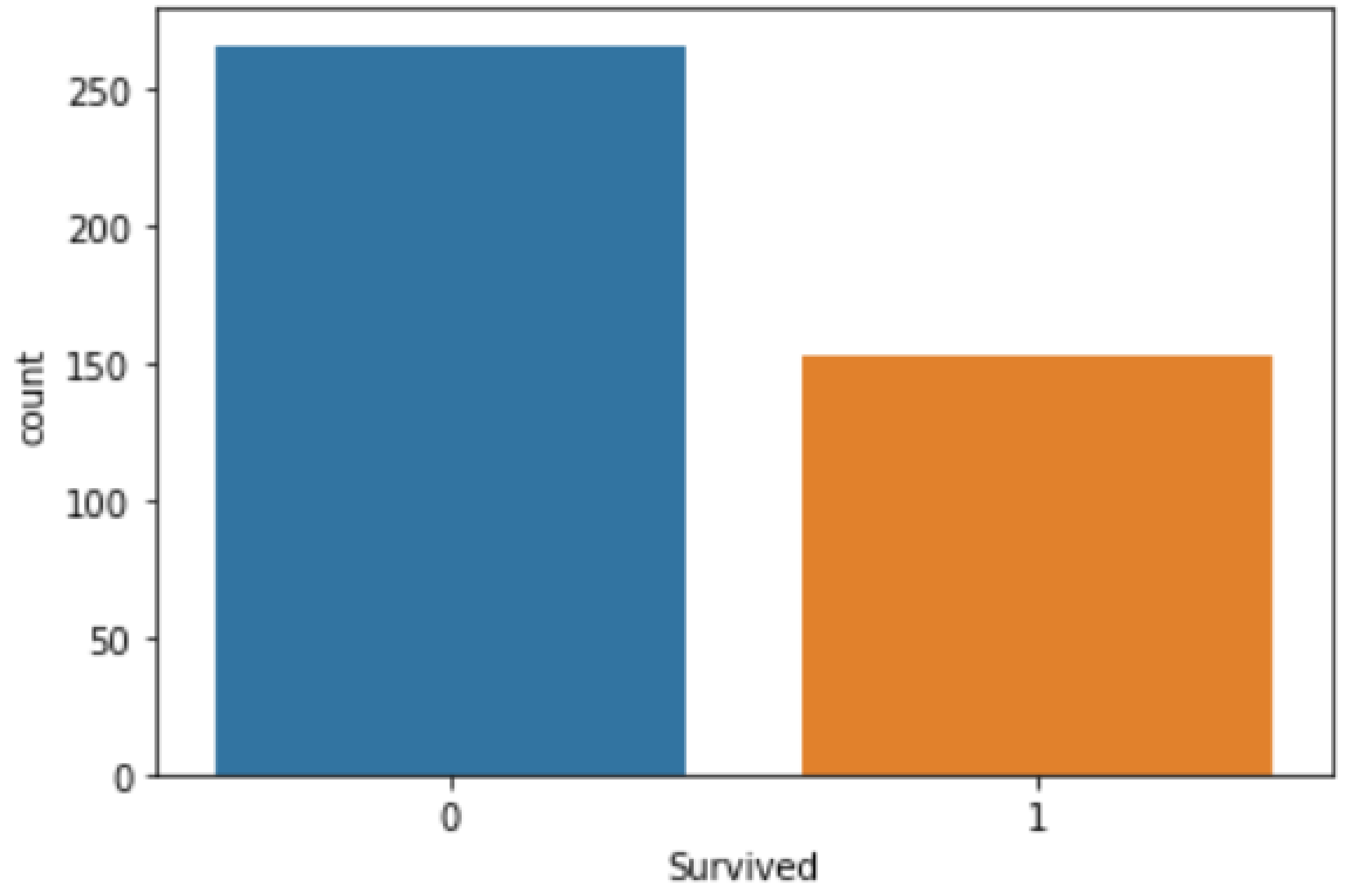
Analyse Data - Using Count plot

Plotting graph to show the survival on x-axis and count on y- axis .

0 - did not survive,
1- survived

```
sns.countplot(x='Survived', data=x)
```

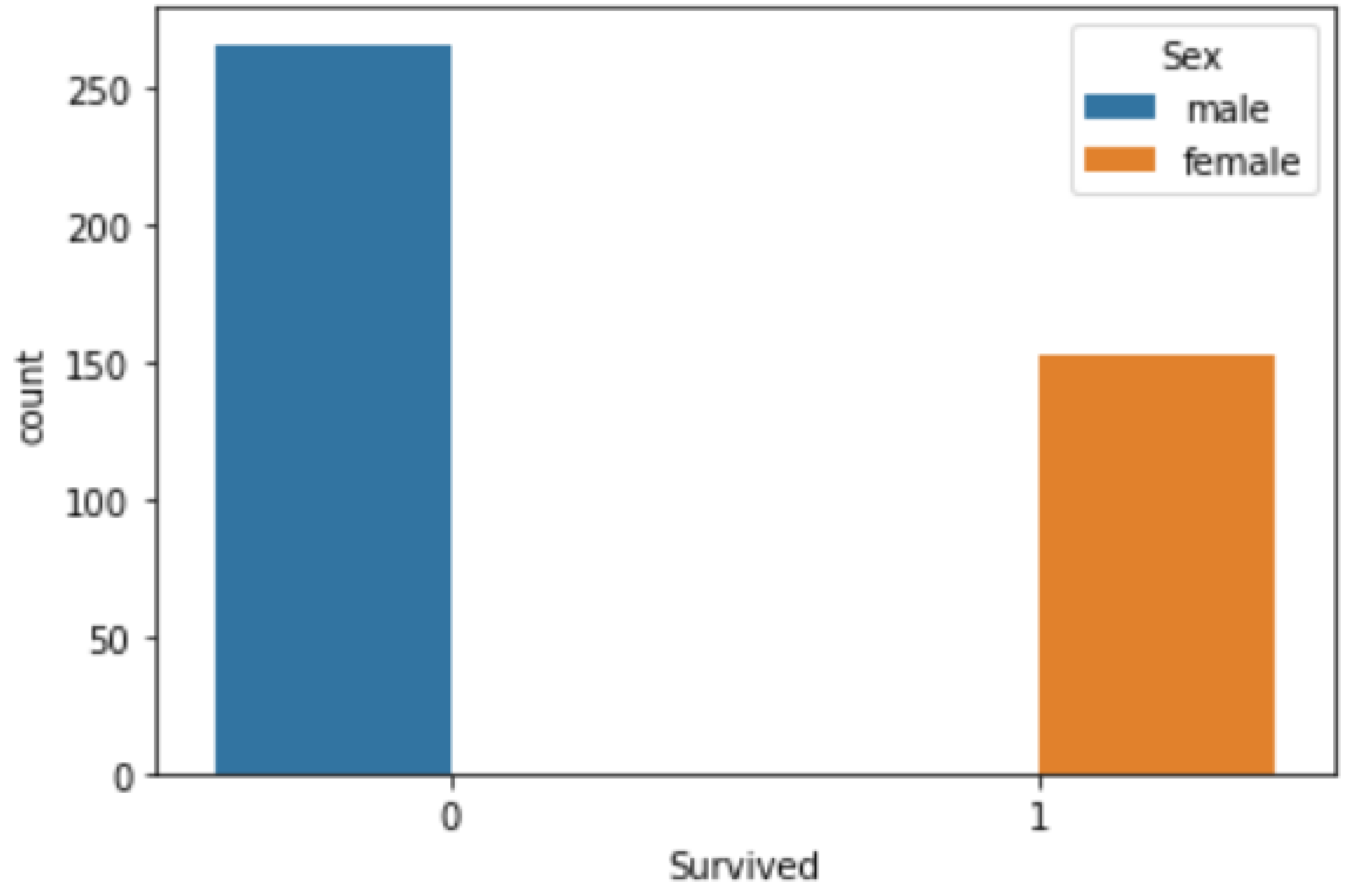
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



Analyzing the gender survival.

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

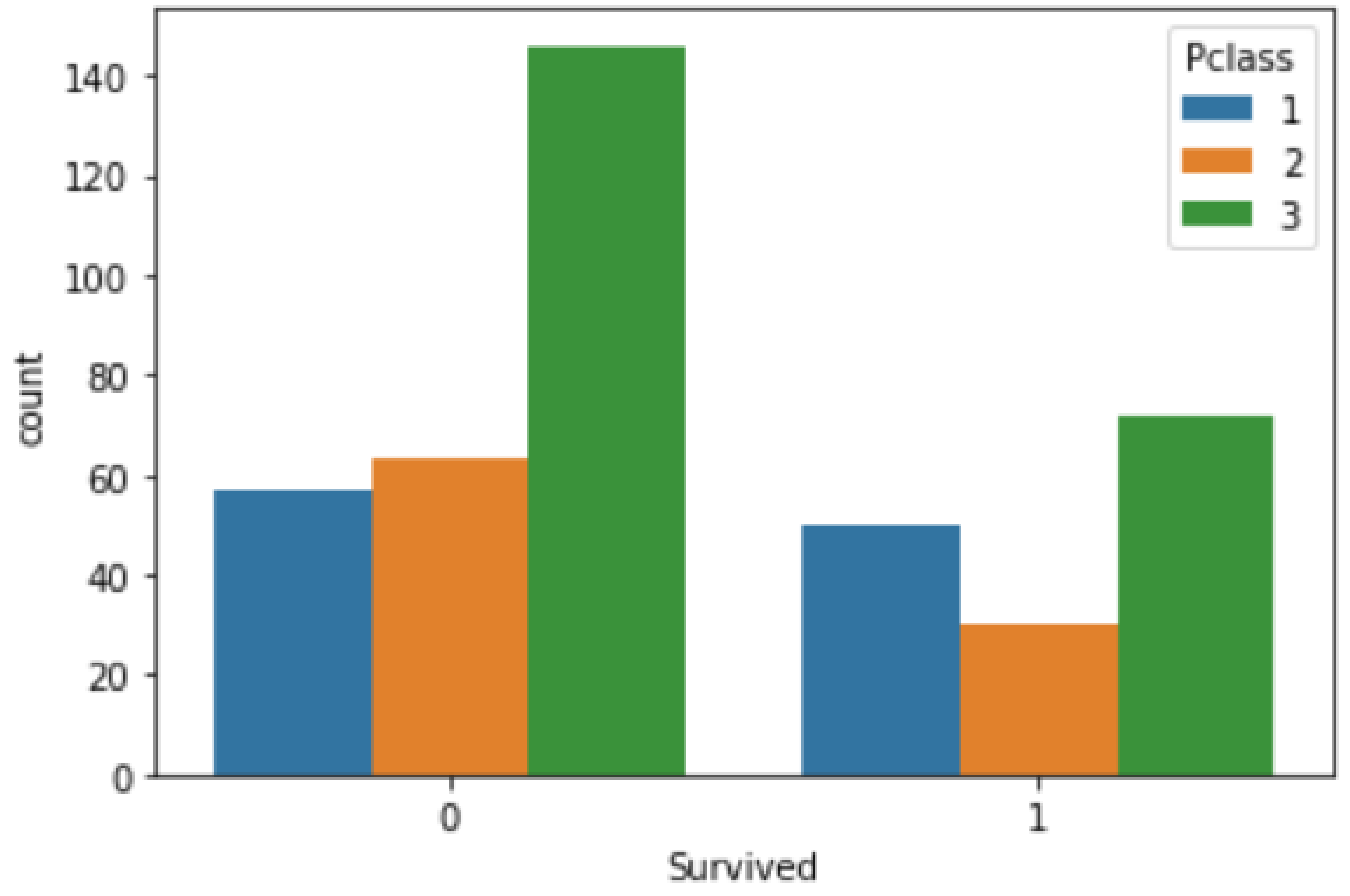
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



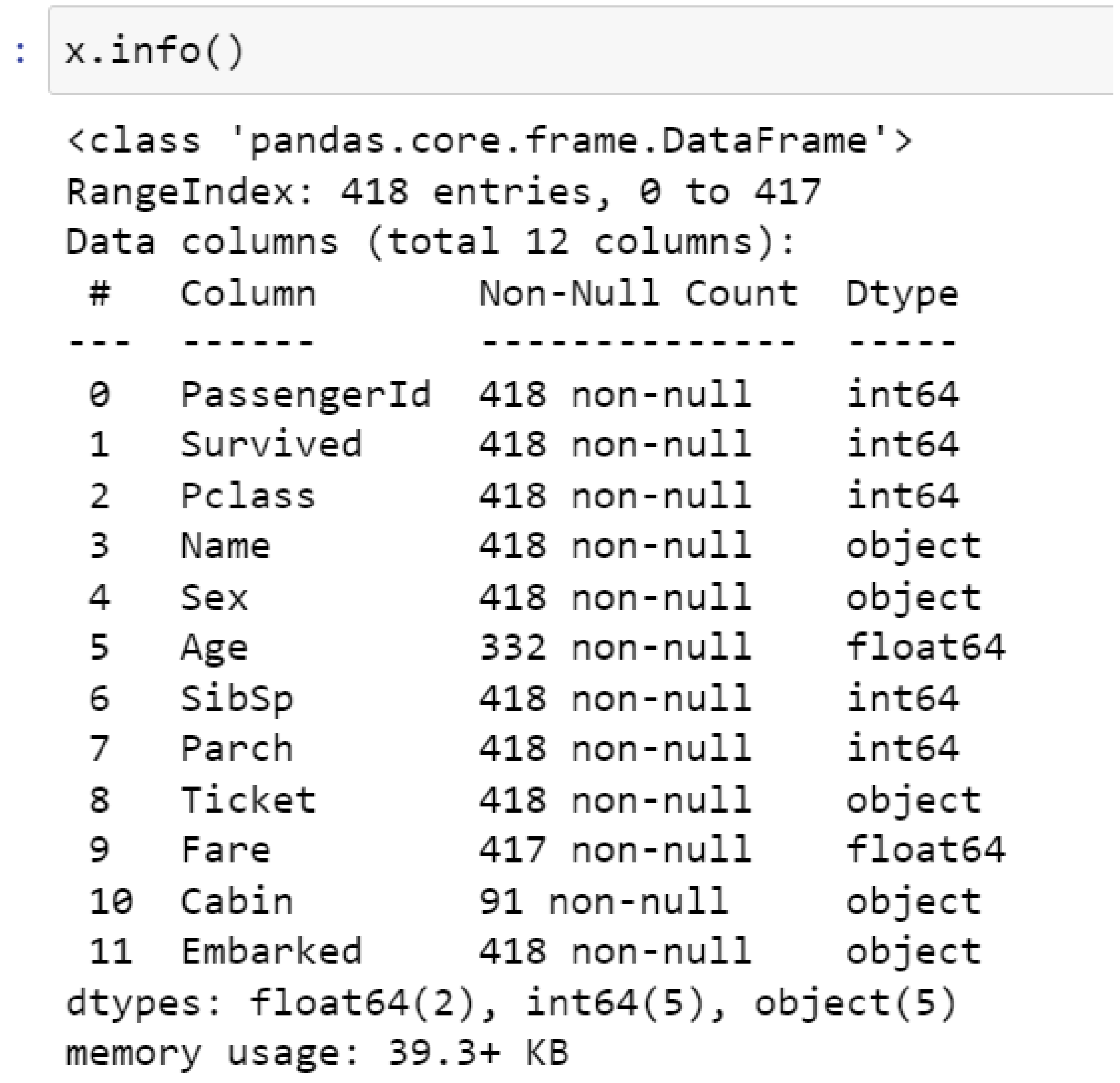
Analyzing the Passenger class wise survival count.

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

```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



We will do data wrangling (cleaning the data) with cabin and embarked column.

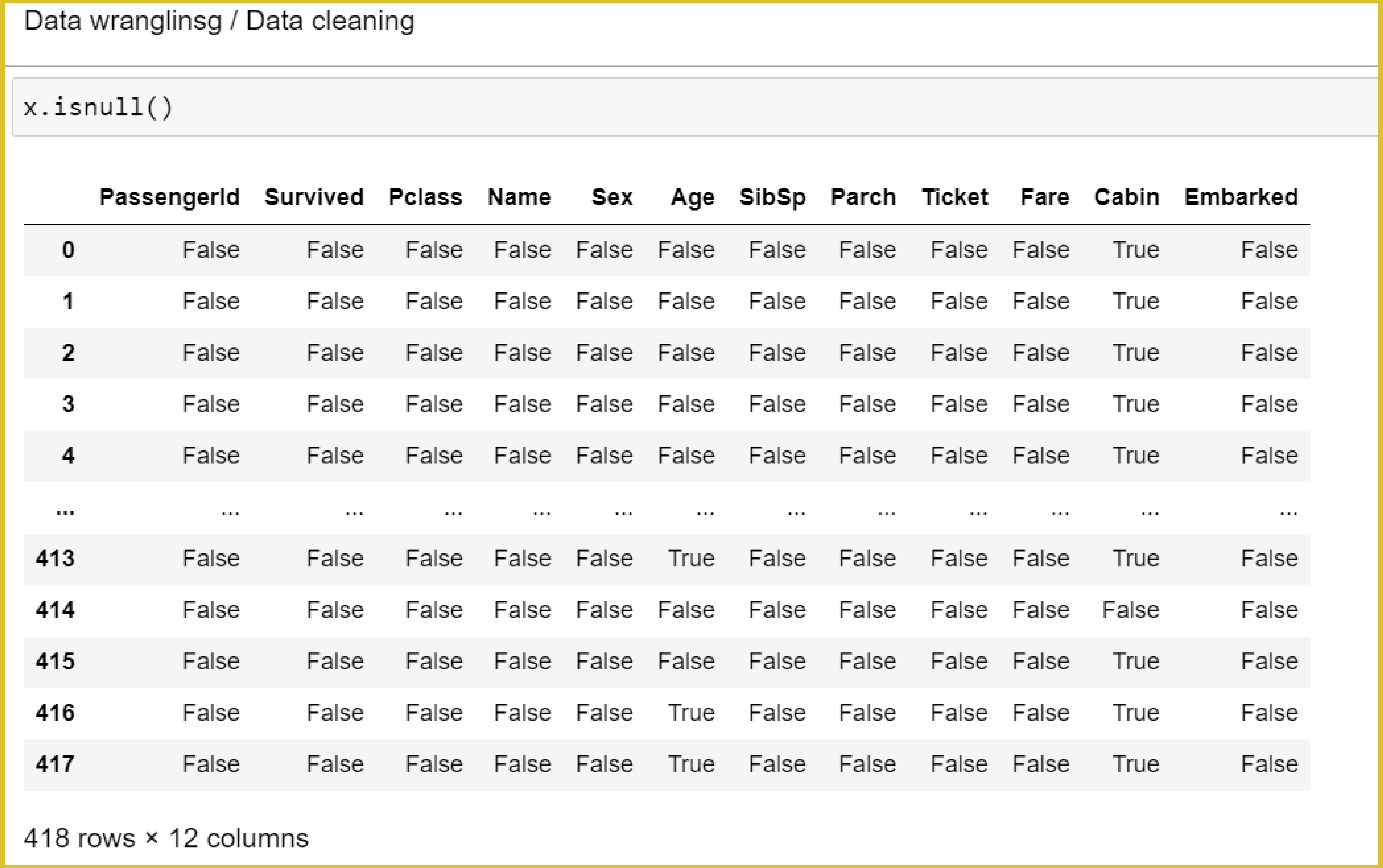


```
x.info()
```

```
<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
```

03

Data Cleaning -
Removing all null data and unnecessary data



Data wranglinsg / Data cleaning

```
x.isnull()
```

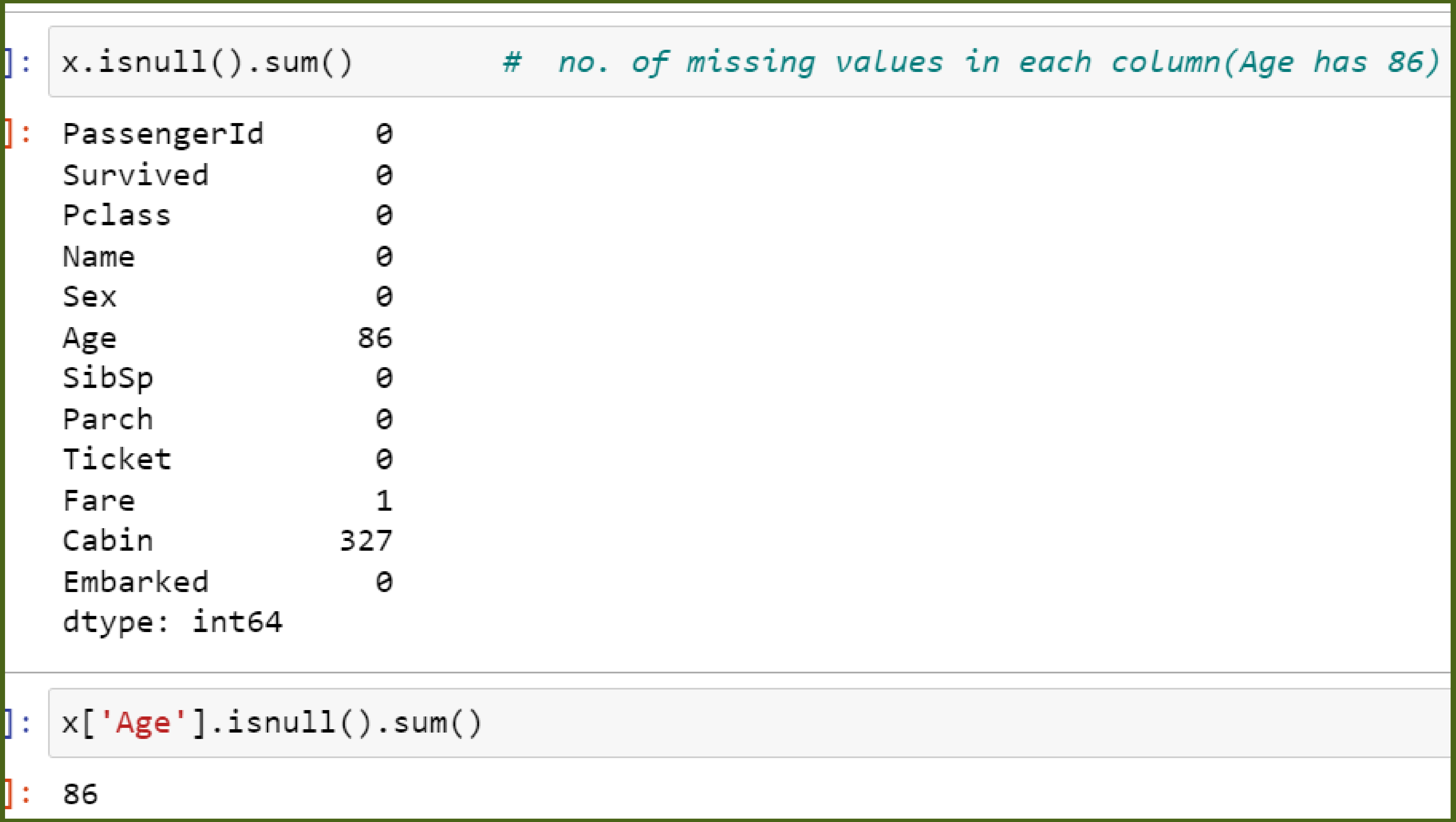
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
413	False	False	False	False	False	True	False	False	False	False	True	False
414	False	False	False	False	False	False	False	False	False	False	False	False
415	False	False	False	False	False	False	False	False	False	False	True	False
416	False	False	False	False	False	True	False	False	False	False	True	False
417	False	False	False	False	False	True	False	False	False	False	True	False

418 rows × 12 columns

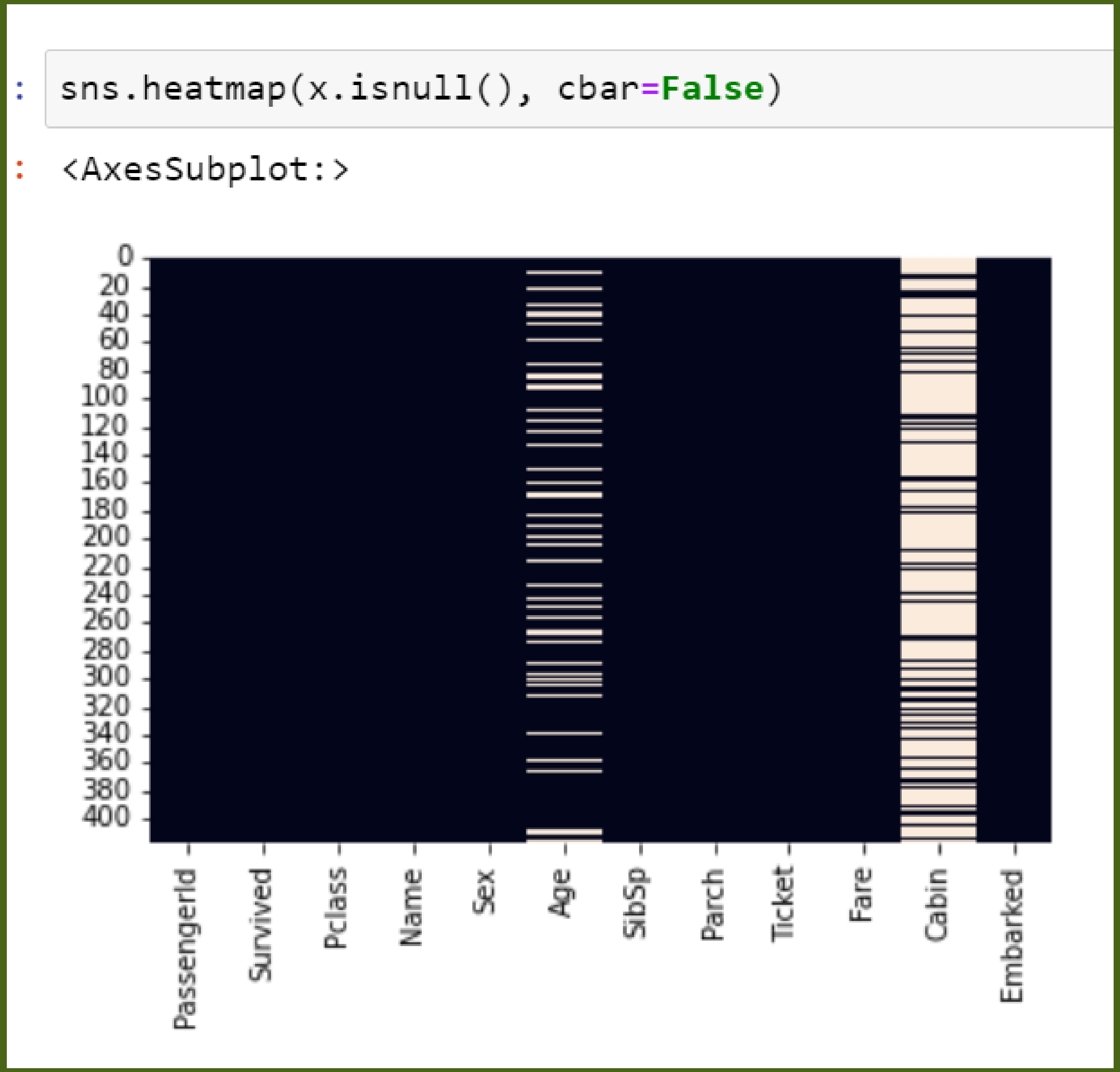
false - not null,
true - null

in age and cabin column we have many null values.

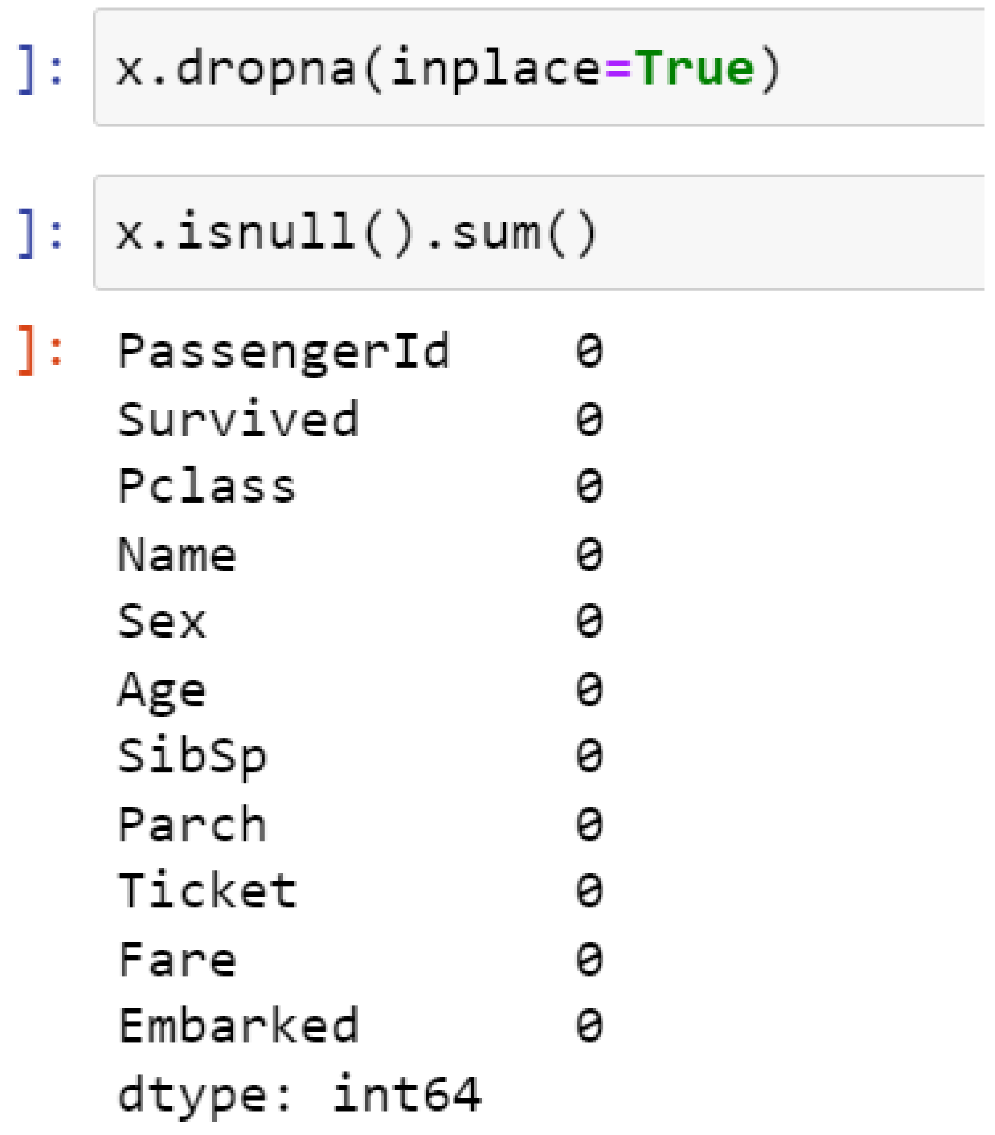
representing the missing data in heatmap.



```
]: x.isnull().sum()       #  no. of missing values in each column(Age has 86)

]: 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

]: x['Age'].isnull().sum()

]: 86
```



```
: sns.heatmap(x.isnull(), cbar=False)

: <AxesSubplot:>
```



```
]: x.dropna(inplace=True)
```

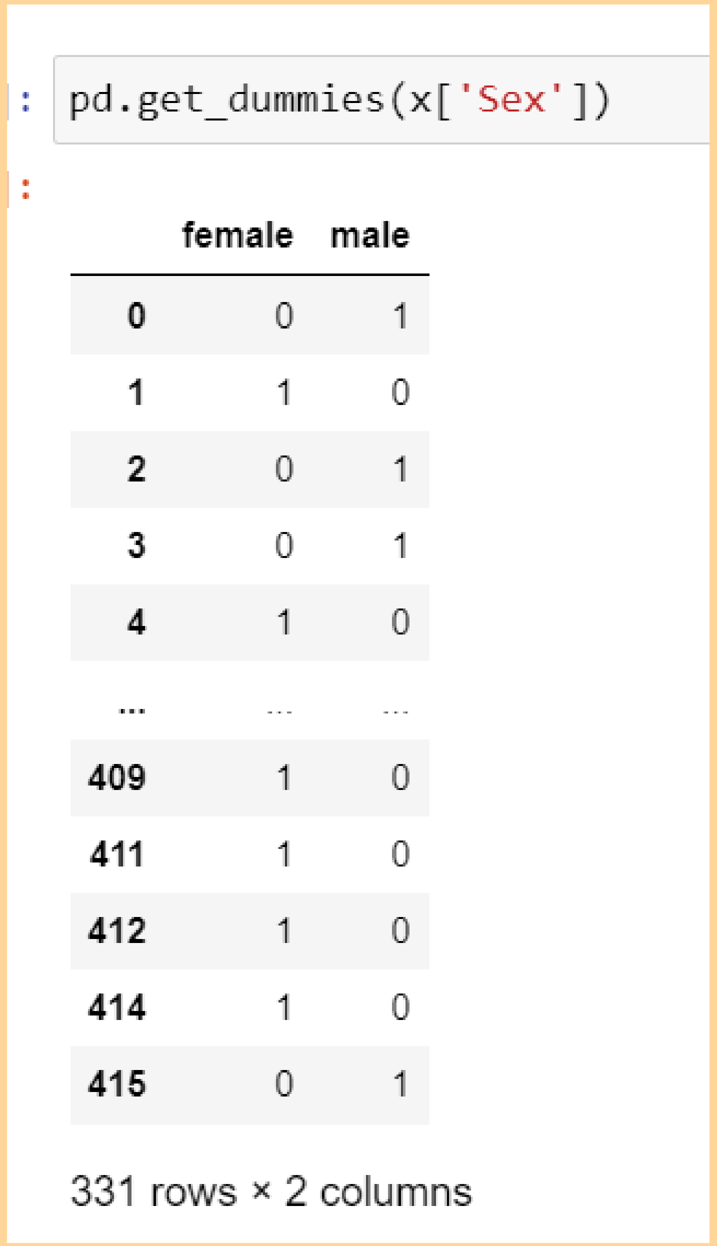
```
]: x.isnull().sum()
```

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

we have the null values and then cabin column and get this at end.

Handling Categorical Data With Dummy Variables

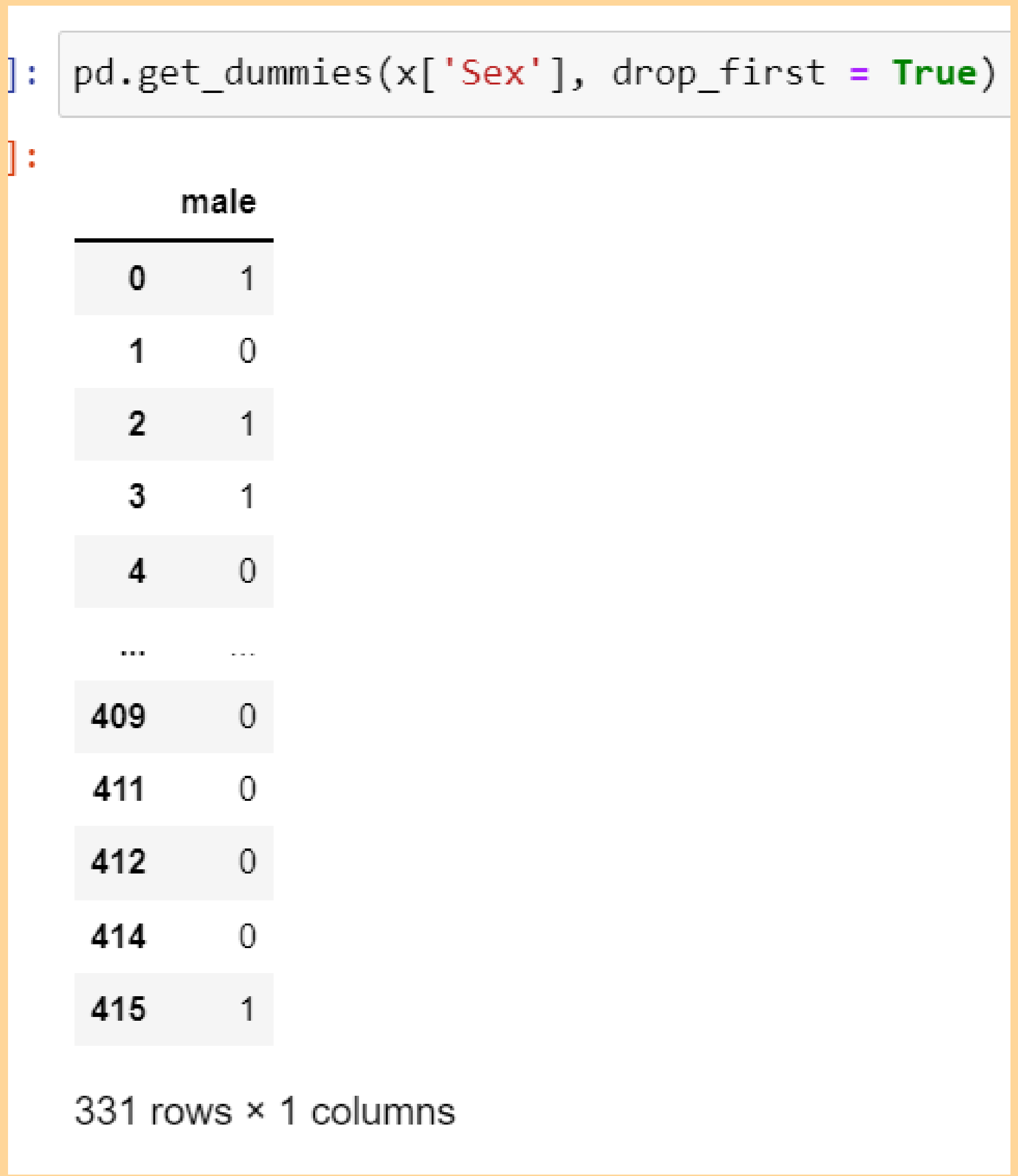
we need to find a way to numerically work with observations that are not naturally numerical .



```
: pd.get_dummies(x['Sex'])
:
```

	female	male
0	0	1
1	1	0
2	0	1
3	0	1
4	1	0
...	...	...
409	1	0
411	1	0
412	1	0
414	1	0
415	0	1

331 rows × 2 columns



```
]: pd.get_dummies(x['Sex'], drop_first = True)
]:
```

	male
0	1
1	0
2	1
3	1
4	0
...	...
409	0
411	0
412	0
414	0
415	1

331 rows × 1 columns

Like -

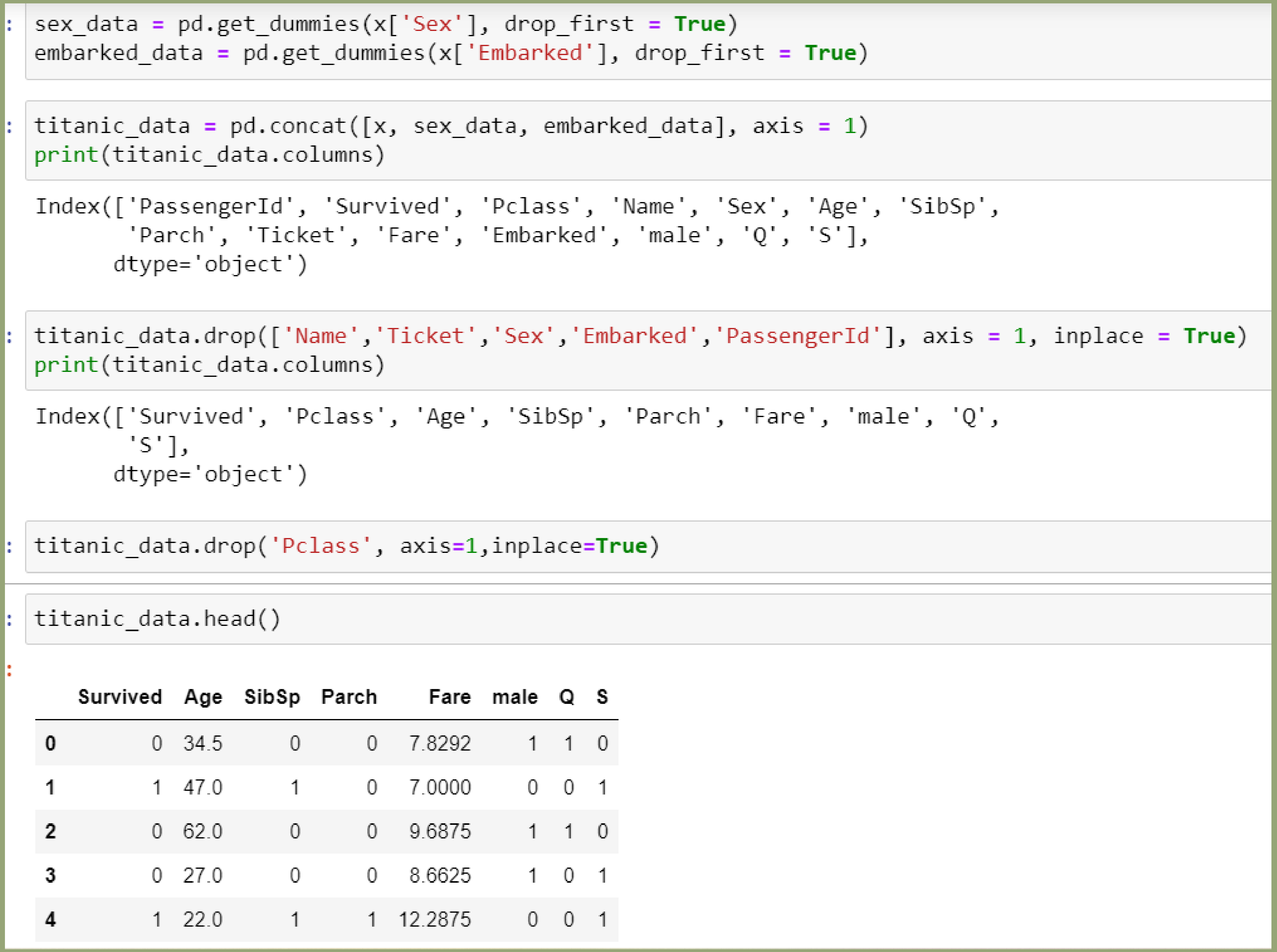
Sex column(male,female)

Embarked column(contains a single letter which indicates which city the passenger departed from).

we use get_dummies() from pandas which makes it easy to create dummy variables.

It will create a new column for each value in the DataFrame column.

After concatenating, we can now drop the original Sex and Embarked columns from the DataFrameand also other columns (like Name , PassengerId, Ticket) which are not predictive of Titanic crash survival rates.



```
sex_data = pd.get_dummies(x['Sex'], drop_first = True)
embarked_data = pd.get_dummies(x['Embarked'], drop_first = True)
```

```
titanic_data = pd.concat([x, sex_data, embarked_data], axis = 1)
print(titanic_data.columns)
```

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

```
titanic_data.drop(['Name','Ticket','Sex','Embarked','PassengerId'], axis = 1, inplace = True)
print(titanic_data.columns)
```

```
Index(['Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'male', 'Q',
       'S'],
      dtype='object')
```

```
titanic_data.drop('Pclass', axis=1,inplace=True)
```

```
titanic_data.head()
```

	Survived	Age	SibSp	Parch	Fare	male	Q	S
0	0	34.5	0	0	7.8292	1	1	0
1	1	47.0	1	0	7.0000	0	0	1
2	0	62.0	0	0	9.6875	1	1	0
3	0	27.0	0	0	8.6625	1	0	1
4	1	22.0	1	1	12.2875	0	0	1

04

Train and Test Data -
Build the model on train data and predict the model on test data.

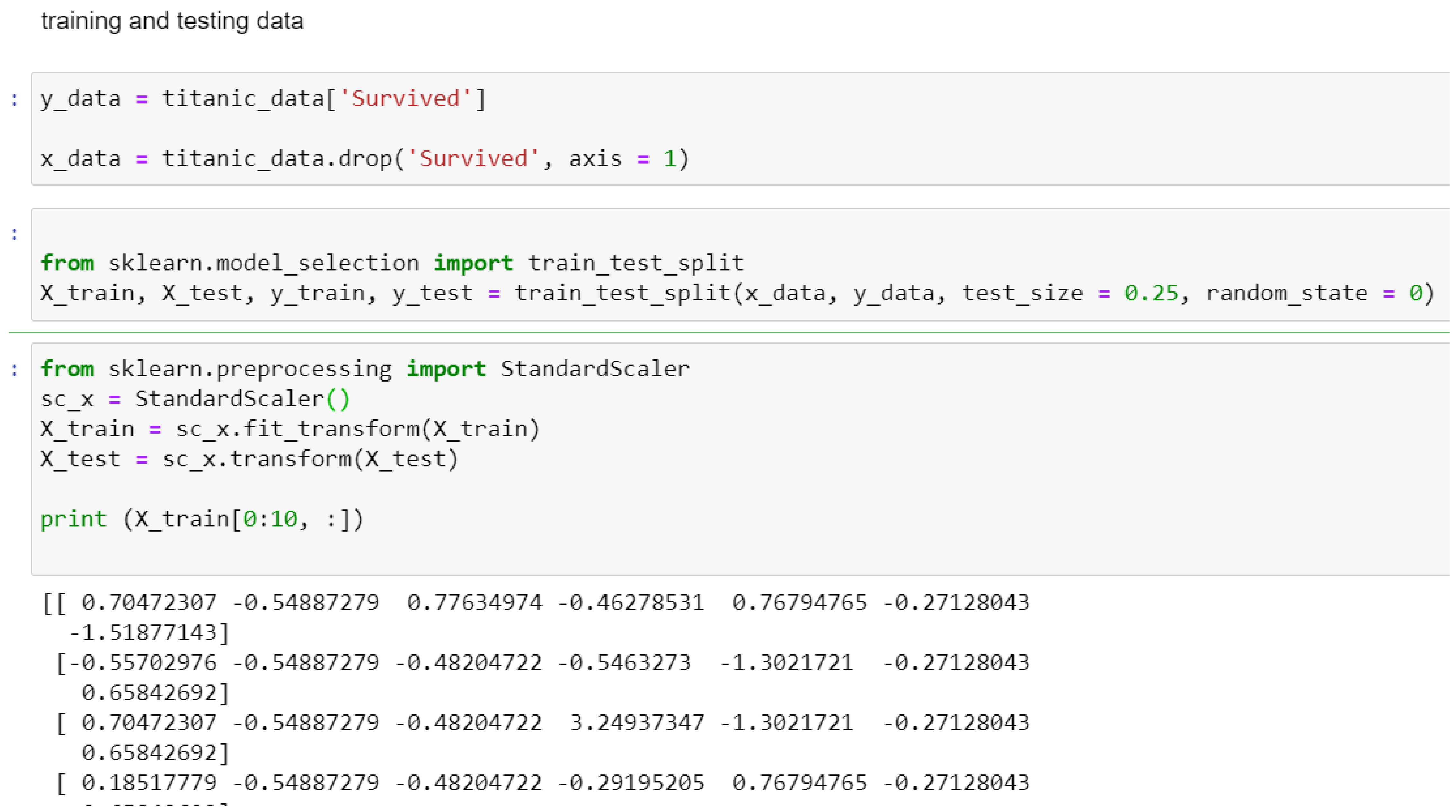
fit_transform() method on our training data and transform() method on our test data.

We use Scikit-learn library and the methods of class sklearn.preprocessing.StandardScaler() so could use almost together while scaling or standardizing our training and test data.

StandardScaler performs the task of Standardization because our dataset contains variable values that are different in scale.

The fit method is calculating the mean and variance of each of the features present in our data. The transform method is transforming all the features using the respective mean and variance.

We are splitting the dataset to train and test. 75% of data is used for training the model and 25% of it is used to test the performance of our model.



training and testing data

```
y_data = titanic_data['Survived']

x_data = titanic_data.drop('Survived', axis = 1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0.25, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)

print (X_train[0:10, :])
```

```
[[ 0.70472307 -0.54887279  0.77634974 -0.46278531  0.76794765 -0.27128043
  -1.51877143]
 [-0.55702976 -0.54887279 -0.48204722 -0.5463273  -1.3021721  -0.27128043
   0.65842692]
 [ 0.70472307 -0.54887279 -0.48204722  3.24937347 -1.3021721  -0.27128043
   0.65842692]
 [ 0.18517779 -0.54887279 -0.48204722 -0.29195205  0.76794765 -0.27128043
```

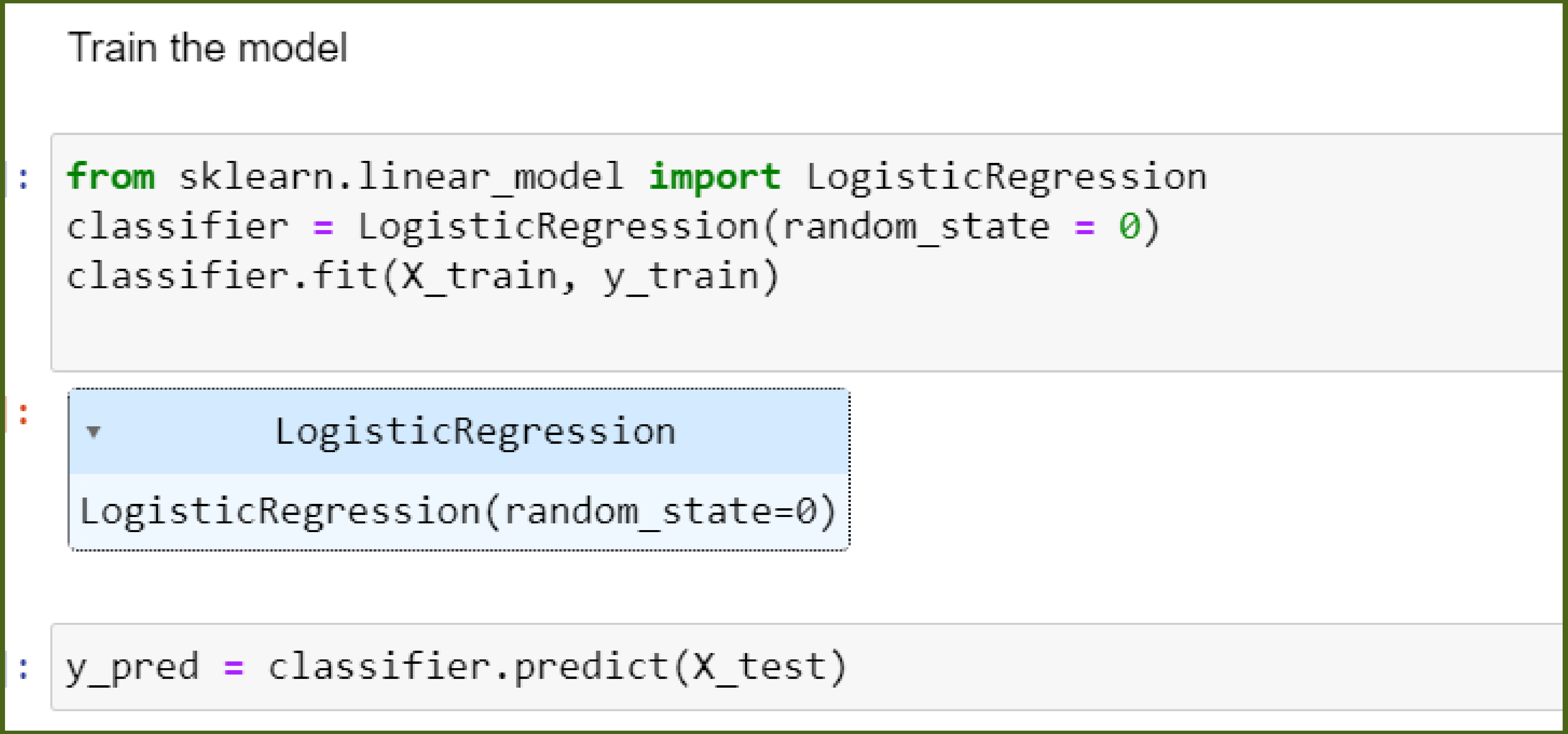
Here first we split the data nd created the model.

Then scaled down inout values

In the output-
We have values which are scaled and now there in the -1 to 1. Hence, each feature will contribute equally to decision making i.e. finalizing the hypothesis.

Finally, we are training our Logistic Regression model.

Here we appliying the Logistic regression to our data and predicting the value.


Train the model

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

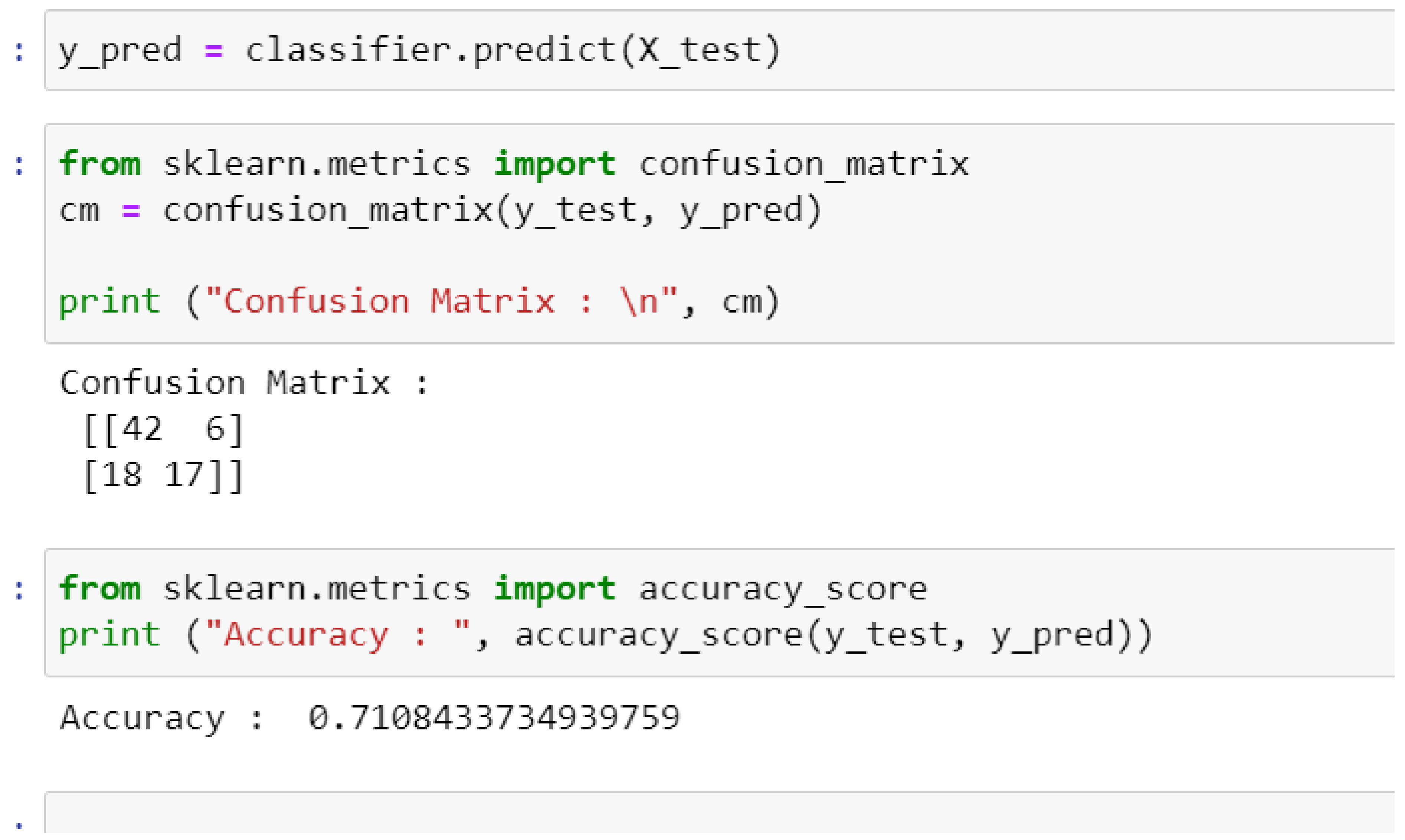
```
▾ LogisticRegression
LogisticRegression(random_state=0)
```

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

05

Accuracy Check-

Using confusion matrix here. But can also use classification prediction.



```
: y_pred = classifier.predict(X_test)
```

```
: from sklearn.metrics import confusion_matrix
  cm = confusion_matrix(y_test, y_pred)

  print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[42  6]
 [18 17]]
```

```
: from sklearn.metrics import accuracy_score
  print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

```
Accuracy :  0.7108433734939759
```

Final step -

we are predicting and getting accuray for the Test data

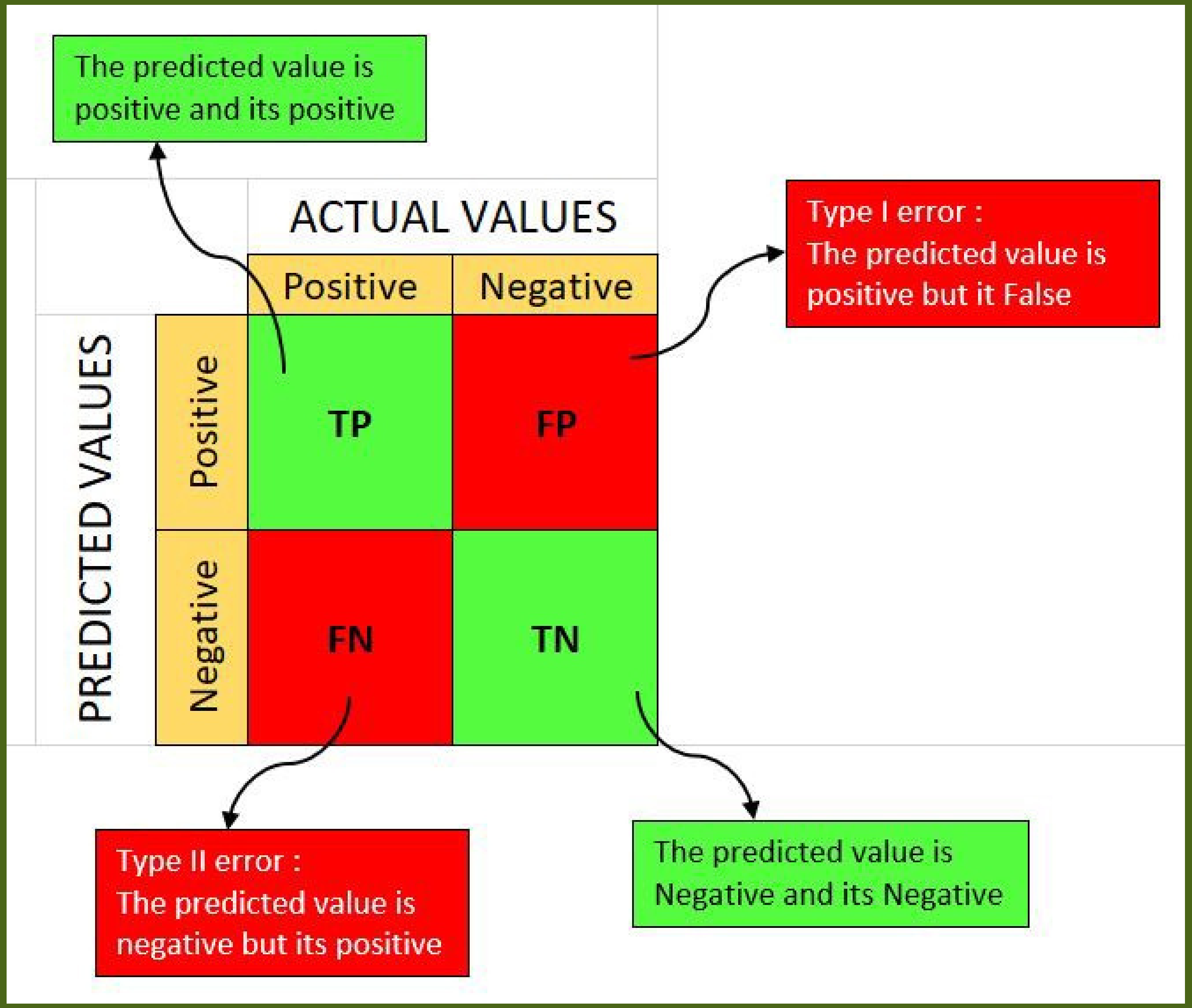
here Actual value(no) and Predicted value(no) value is 42 and Actual value(yes) and Predicted value(yes) value is 17.

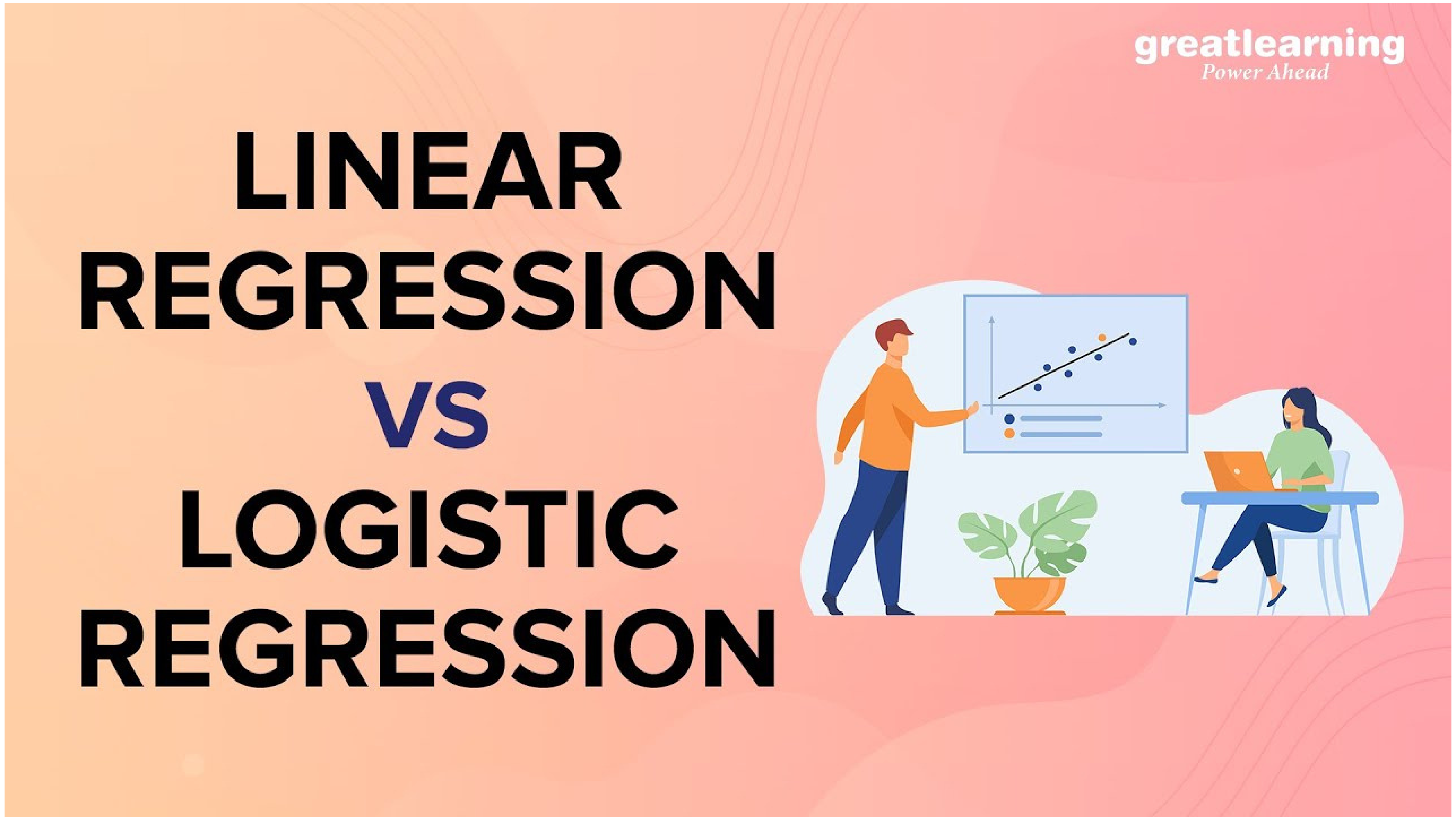
We have model accuracy as 71% .

What is Confusion Matrix ?

A confusion matrix is a performance measurement technique for Machine learning classification. It is a kind of table which helps you to the know the performance of the classification model on a set of test data for that the true values are known.

TP: True Positive: Predicted values correctly predicted as actual positive
FP: Predicted values incorrectly predicted an actual positive. i.e., Negative values predicted as positive
FN: False Negative: Positive values predicted as negative
TN: True Negative: Predicted values correctly predicted as an actual negative


The predicted value is positive and its positive
ACTUAL VALUES
Positive
Negative
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type I error : The predicted value is positive but it False
Type II error : The predicted value is negative but its positive
The predicted value is Negative and its Negative


greatlearning
Power Ahead
LINEAR REGRESSION
VS
LOGISTIC REGRESSION

LINEAR REGRESSION	**LOGISTIC REGRESSION**
A regression analysis which is used to predict the value of a variable based on the value of another variable.	**A statistical model that predicts the probability of an outcome that can only have two values**
Used to solve regression problems	**Used to solve classification problems (binary classification)**
Estimates the dependent variable when there is a change in the independent variable	**Calculates the possibility of an event occurring**
Output value is continuous & Uses a straight line	**Output value is discrete & Uses an S curve or sigmoid function**
Ex: predicting the GDP of a country, predicting product price, predicting the house selling price, score prediction	**Ex: predicting whether an email is spam or not, predicting whether the credit card transaction is fraud or not, predicting whether a customer will take a loan or no**

Thank you!