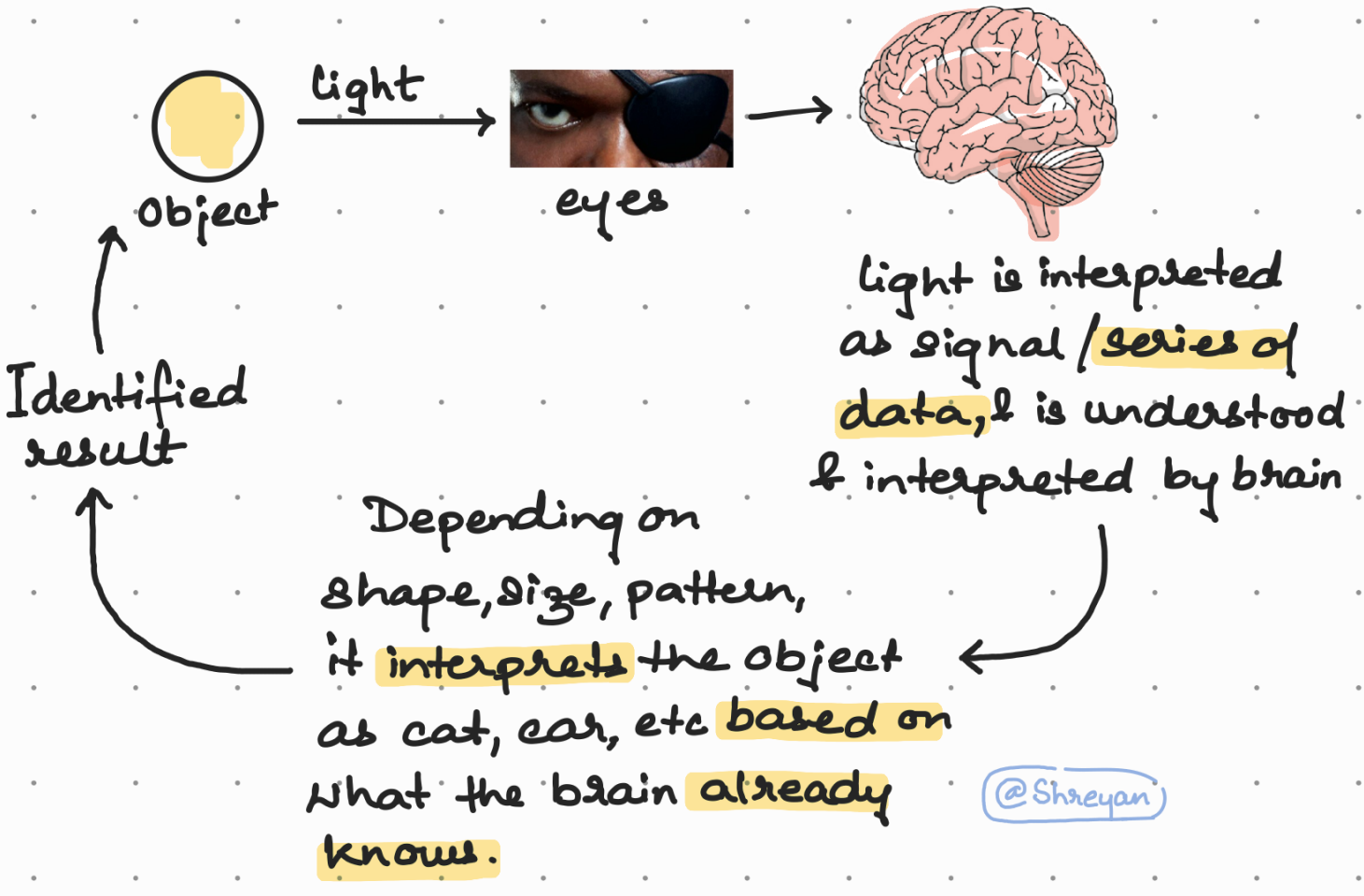
The Machine Learning Journey

Chapter - 02

Simple Linear Regression

Lets talk about our brain first!

How does your brain analyses data received from your sense organs?



A machine learning model also does the same thing.

Data is given to model → Uses mathematical formula and fits the data into that formula

if succesful → model will understand relationship b/w pieces of data.

if not succesful → results will be not accurate or with errors

Regression →

Statistically speaking, 'regression' means measure of relation between variables and pieces of data.

There are 2 objectives →

- ϕ To establish relationship b/w 2 variables.
- ϕ Predict new observations.

Variables →

- A variable is nothing but data.
- Now we know, that a data frame contains data in form of rows x columns.
- Here, the columns are variables.
- These are classified into 2 types -

ϕ Dependent variables : (y)

- the variable whose value is to forecast or predict.
- also called 'response variable' or 'target variable'.
- in equations, it is represented with the letter 'y'.

ϕ Independent variables : (x)

- used to calculate the value of another variable.
- also called 'features' or 'regressors'.
- in equations, it is represented by letter 'x'.

Linear Regression:

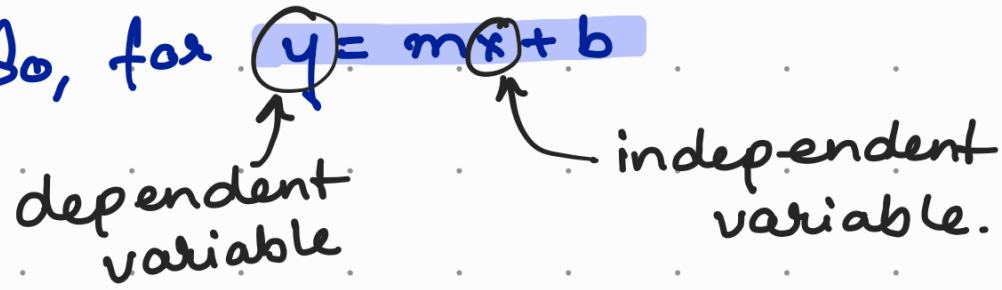
- It is a machine learning model that depends on the linear relationship b/w a dependent variable and one or more independent variables.

- Linear relationship indicates values (data points) lie on straight lines.

- if there is one independent variable then it is called Simple Linear Regression, or least Squares regression.

  for more than one independent variable, it is called Multiple Linear Regression.

So, for $y = mx + b$



In statistics, we write the same equation as —

$$y = B_0 + B_1 x$$

B_1 → slope

B_0 → intercept (distance on y-axis)

@Shreyan

SORRY ! BUT ITS ENTIRELY MATHEMATICS 😬

→

Let us take a linear equation—

$$y = 4 + 2x$$

Comparing with the master eqn.,

$$b_0 = 4 \quad , \quad b_1 = 2$$

So, by substituting 'x', we can get the value of 'y', hence—

$x \rightarrow$ independent var.

$y \rightarrow$ dependent var.

if, $x = 0, \; y = 4$

$x = 2, \; y = 8$

$x = 4, \; y = 12$

$x = 6, \; y = 16$

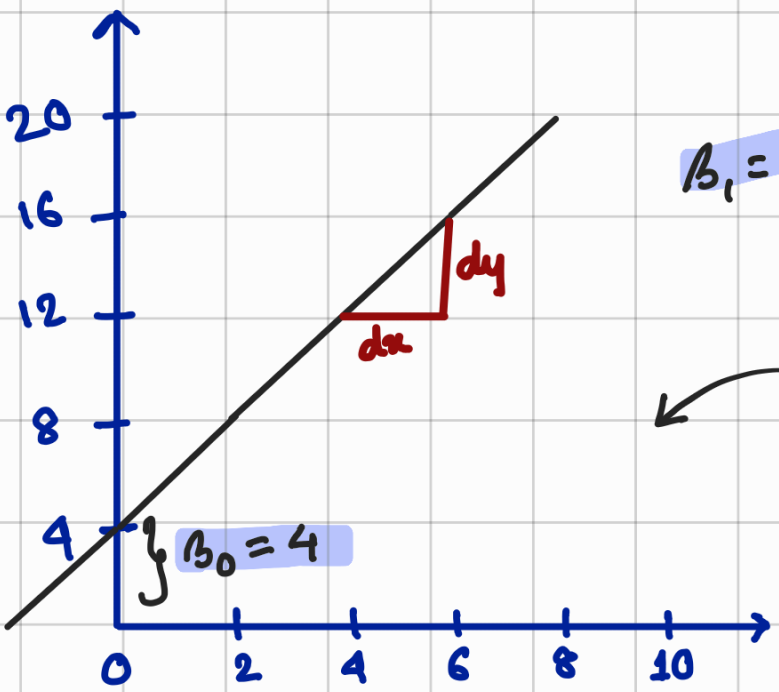
So, if x is increased in steps of 2,

y values increases in steps of 4.

$$\therefore \boxed{\text{Slope } (b_1) = \frac{\text{deviation in } y}{\text{deviation in } x}} = \frac{dy}{dx}$$

$$\therefore b_1 = \frac{4}{2} = 2$$

and b_0 = distance on y-axis where line crosses = 4.



$$b_1 = \frac{dy}{dx} = \frac{4}{2} = 2$$

Here, as we see the relation b/w x and y are linear, so we can apply Linear Regression model to analyse the data.

The R-squared value :

Now, when you plot the dataset, the data points may not be exactly on the straight line. There will be deviation. This is what the concept of R-squared value, handles.

The R^2-value is thus also known as the co-effecient of determination.

Concept →

- R^2 values ranges from 0–1.
  - > 0 : There is a huge variation b/w the predicted and the actual values.
  - > 1 : minimal or no variation b/w the values.
- A higher R^2 value indicates a better fit for the model.

Let us take an example, where you are trying to predict a student's test score based on the no. of hours they studied.

Step 1 : Suppose we have data on study hours vs. test scores.

Step 2 : We fit a linear regression model to this data.

Step 3: The R-squared value is calculated.

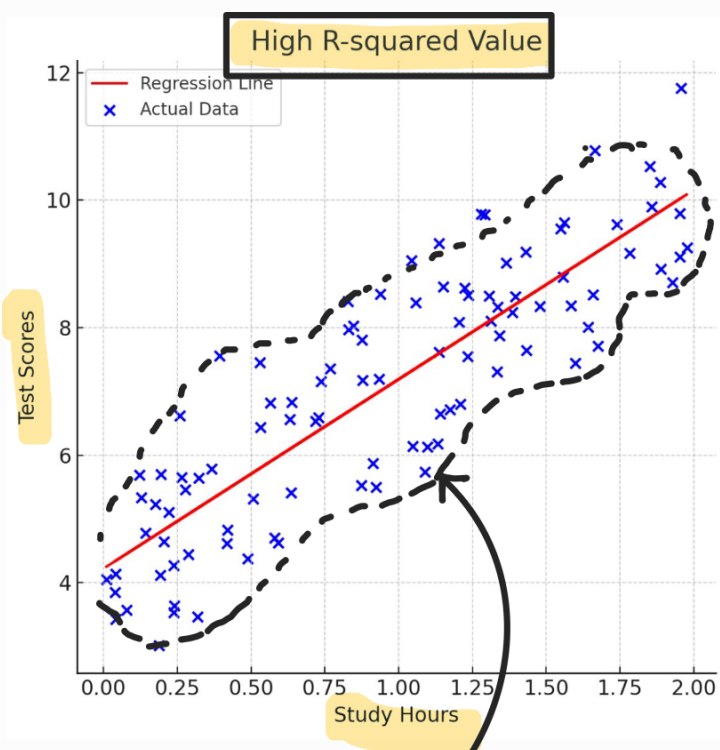
$$R^2 = 1 - \frac{SS_{Res}}{SS_{Tot}} = 1 - \frac{\Sigma_i (y_i - \hat{y}_i)^2}{\Sigma_i (y_i - \bar{y})^2}$$

- SS_{Res} → Sum of squares of residuals
- SS_{Tot} → Total sum of squares
  - y_i → actual observed value.
  - $\hat{y}_i$ → predicted value from reg. model.
  - $\bar{y}$ → mean of all observed values (y_i).

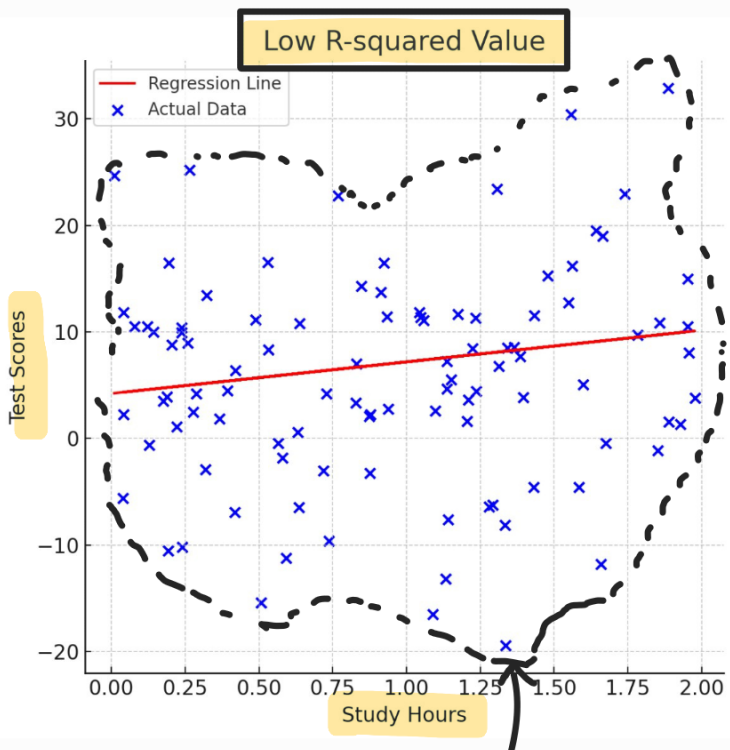
@Shreyan

Good fit → If the points passes through most of the line or, are close to the line, it is a high R^2-value.

Poor fit → if the points are scattered distantly from the line, it is a low R^2-value.



most-points are near the lines hence it is a high R^2-value.



data points are too widely scattered, hence a low R^2-value.

Predict the price of a house based on area 3500 & 5000 sq.ft from training the given dataset.

In [1]:
```
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LinearRegression
# import warnings
# warnings.filterwarnings('ignore')
```

In [2]:
```
#load the data into dataframe

df= pd.read_csv("homeprices.csv")
df
```

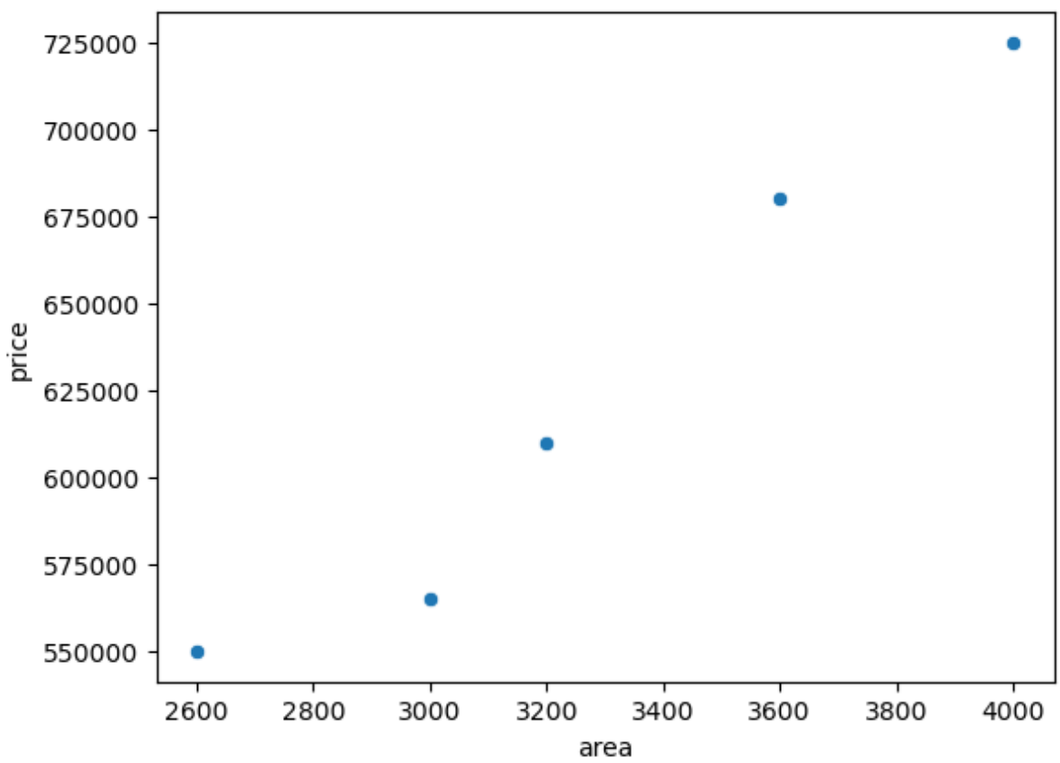
Out[2]:

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

Let us draw a scatter plot to see the relationship between the area and the price values. This can be done using the scatterplot() function of the seaborn module -

In [3]:
```
#plot a scatter plot
sns.scatterplot(data=df, x='area', y='price')
```

Out[3]:
```
<Axes: xlabel='area', ylabel='price'>
```



We can understand by observing the dots in the scatter plot that they can be connected more or less using a straight line. Hence we can apply a Simple Linear Regression ML model on this. We make this model available to us in the form of a 'reg' object. We can now call any methods od the LinearRegression class usinf this object 'reg'.

Now the next step is to train the ML model, by calling 'fit()' method on the data -

reg.fit(x,y)

Here, 'x' indicates independent variable i.e df.area. This should be supplied in the form of 2D array to the fit() method. 'y' indicates the dependent variable, i.e - df.price.

Now the question is - How to convert the df.area column data into a 2D array ? Well there are 2 ways - the first way is to convert the df.area column into an array by using values attribute as -

***df.area.values* #gives 1D array**

Now convert the 1D into a 2D array by using reshape() method of numpy arrays as -

In [4]:
```
reg = LinearRegression()
df.area.values #gives 1D array
```

Out[4]:
```
array([2600, 3000, 3200, 3600, 4000])
```

In [5]:
```
df.area.values.reshape(-1,1)
```

Out[5]:
```
array([[2600],
       [3000],
       [3200],
       [3600],
       [4000]])
```

In [6]:
```
#Another way of converting the df.area value into a 2D array is to take
reg.fit(df[['area']], df.price) #fitting means training
```

Out[6]:
```
LinearRegression
LinearRegression()
```

Fitting means the model has been trained with data, which means that the model tried to fit the data in the form of a straight line. This indicates that there is a linear relationship existing between the 'area' and 'price' columns.

Now let us test it with a data that is not there in the dataset -

In [7]:
```
#predict the price of a 3500 sqft house
reg.predict([[3500]])
```

Out[7]:
```
array([655873.28767123])
```

In [8]:
```
#find the coefficient, this is slope m
reg.coef_
```

Out[8]:
```
array([135.78767123])
```

In [9]:
```
#find the intercept, this is b
reg.intercept_
```

Out[9]:
```
180616.43835616432
```

In [10]:
```
# if we substitute m and b values in y=mx+b,
# we get the predicted value as above

y=135.78767123*3500 + 180616.43835616432
y
```

Out[10]:
```
655873.2876611643
```

Trying for another value - for 5000 sqft area

In [11]:
```
reg.predict([[5000]])
```

Out[11]:
```
array([859554.79452055])
```

Finding the accuracy level of the model by finding r squared values

In [12]:
```
from sklearn.metrics import r2_score

y_original = df.price
y_predicted = reg.predict(df[['area']])

R_square = r2_score(y_original, y_predicted)
print('r squared value', R_square)
```

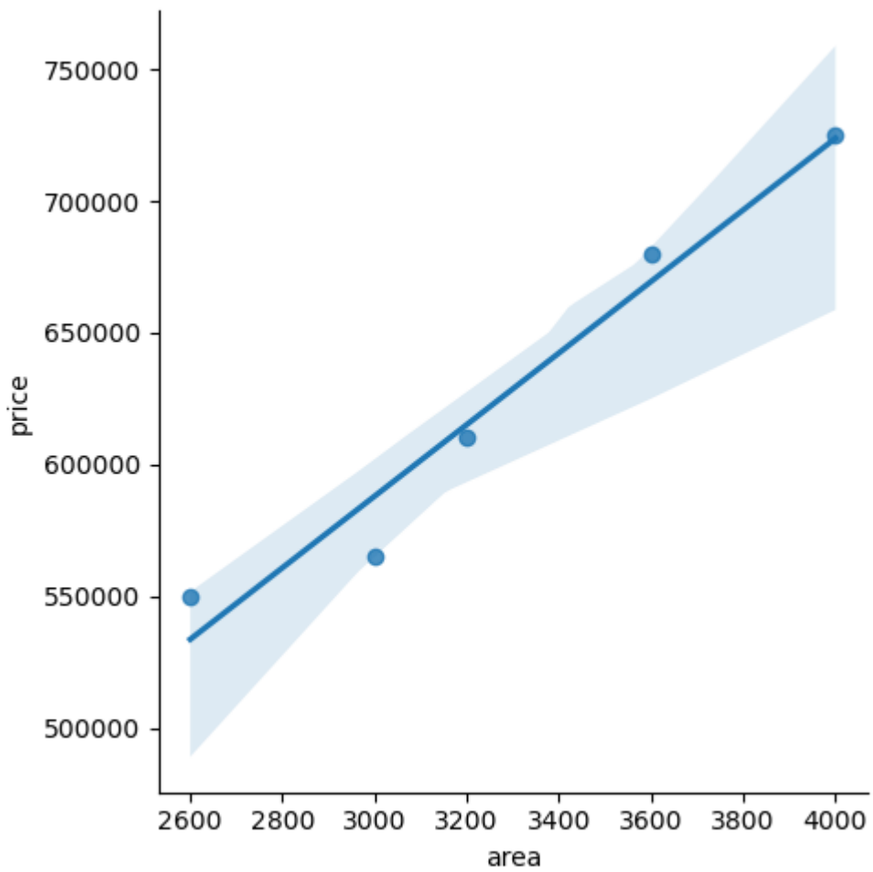
```
r squared value 0.9584301138199486
```

r-squared value is 0.95843... , hence indicating a 95.8% accuracy for this model which is quite good.

Let us check the scatter plot with a regression line using lmplot(). lmplot() will draw the straight line that best fits the data points.

In [13]:
```
sns.lmplot(data=df, x='area', y='price')
```

Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x7f54507b4dd0>
```



Splitting data into train and test :

By general practice, we divide the train-test data into 70:30, or we can use 80:20.

@Shreyan

To split the data into train and test –

```
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,
                                random_state = 0)
```

Here, test_size = 0.3, this indicates that the test size is 30%, and the remaining 70% will be used for training.

random_state = 0 → it will take seed value as 0 and create some random numbers as - 4, 1, 2, 7, 3, which means 4th, 1st, 2nd, 7th, 3rd rows are selected for testing purpose.

we can also use –

```
x_train, x_test, y_train, y_test =
        train_test_split(x, y, train_size = 0.7)
```

To find : per capita income of Canada for 2022, 2023 based on dataset provided.

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv("canada_per_capita_income.csv")
```

In [3]:
```
x = dataset.iloc[:,0:1].values
x
```

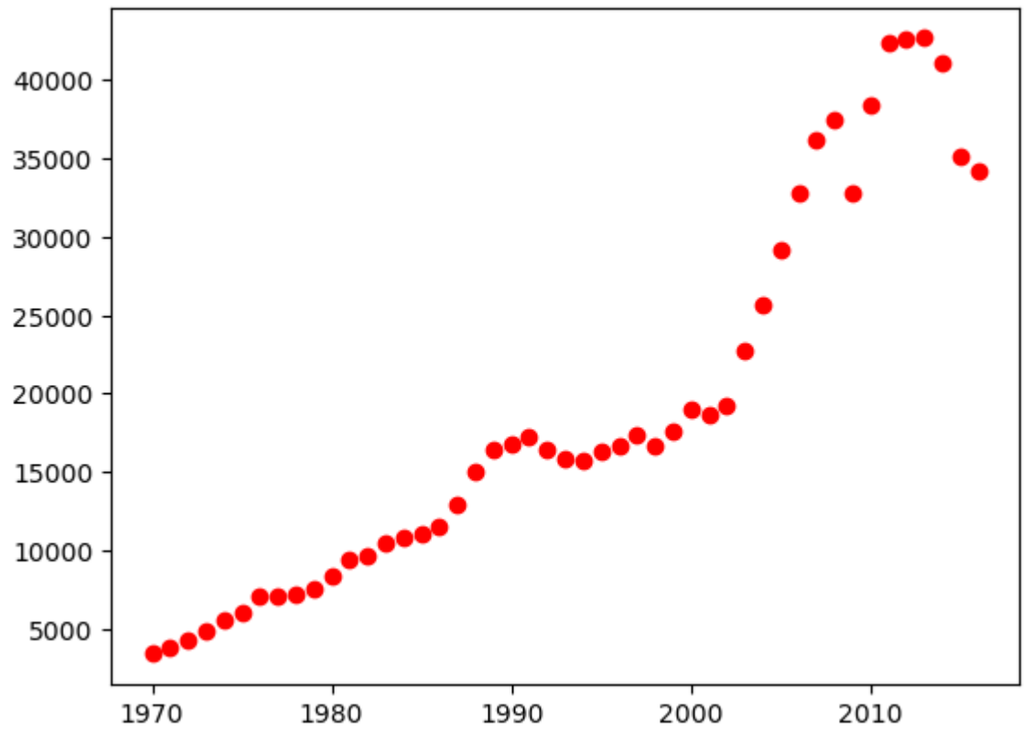
Out[3]:
```
array([[1970],
       [1971],
       [1972],
       [1973],
       [1974],
       [1975],
       [1976],
       [1977],
       [1978],
       [1979],
       [1980],
       [1981],
       [1982],
       [1983],
       [1984],
       [1985],
       [1986],
       [1987],
       [1988],
       [1989],
       [1990],
       [1991],
       [1992],
       [1993],
       [1994],
       [1995],
       [1996],
       [1997],
       [1998],
       [1999],
       [2000],
       [2001],
       [2002],
       [2003],
       [2004],
       [2005],
       [2006],
       [2007],
       [2008],
       [2009],
       [2010],
       [2011],
       [2012],
       [2013],
       [2014],
       [2015],
       [2016]])
```


In [4]:
```
y = dataset.iloc[:, -1].values
y
```

Out[4]:
```
array([ 3399.299037,  3768.297935,  4251.175484,  4804.463248,
        5576.514583,  5998.144346,  7062.131392,  7100.12617 ,
        7247.967035,  7602.912681,  8355.96812 ,  9434.390652,
        9619.438377, 10416.53659 , 10790.32872 , 11018.95585 ,
       11482.89153 , 12974.80662 , 15080.28345 , 16426.72548 ,
       16838.6732  , 17266.09769 , 16412.08309 , 15875.58673 ,
       15755.82027 , 16369.31725 , 16699.82668 , 17310.75775 ,
       16622.67187 , 17581.02414 , 18987.38241 , 18601.39724 ,
       19232.17556 , 22739.42628 , 25719.14715 , 29198.05569 ,
       32738.2629  , 36144.48122 , 37446.48609 , 32755.17682 ,
       38420.52289 , 42334.71121 , 42665.25597 , 42676.46837 ,
       41039.8936  , 35175.18898 , 34229.19363 ])
```

In [5]:
```
# check the distribution of data is linear or not
plt.scatter(x,y,color='red')
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x7fd2694c5250>
```



In [6]:
```
#take 70% of data for training and 30% for testing
#random_state indicates the random seed used in splitting the 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,
```

In [7]:
```
#train using SLR
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train, y_train)
```

Out[7]:
```
▾ LinearRegression

LinearRegression()
```


In [8]:
```
#make prediction based on the test data
y_pred = reg.predict(x_test)
y_pred
```

Out[8]:
```
array([22865.78838995, 21142.72401552,  3912.08027117, 13388.93433056,
       18558.12745387, 15111.998705  ,   465.95152231, 34065.70682378,
        1327.48370952, 23727.32057717,  2189.01589674, 30619.57807491,
       -1257.11285213, 16835.06307943,  3050.54808396])
```

In [9]:
```
#find the r-squared value by comparing test data and predicted data
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[9]:
```
0.8433026110551844
```

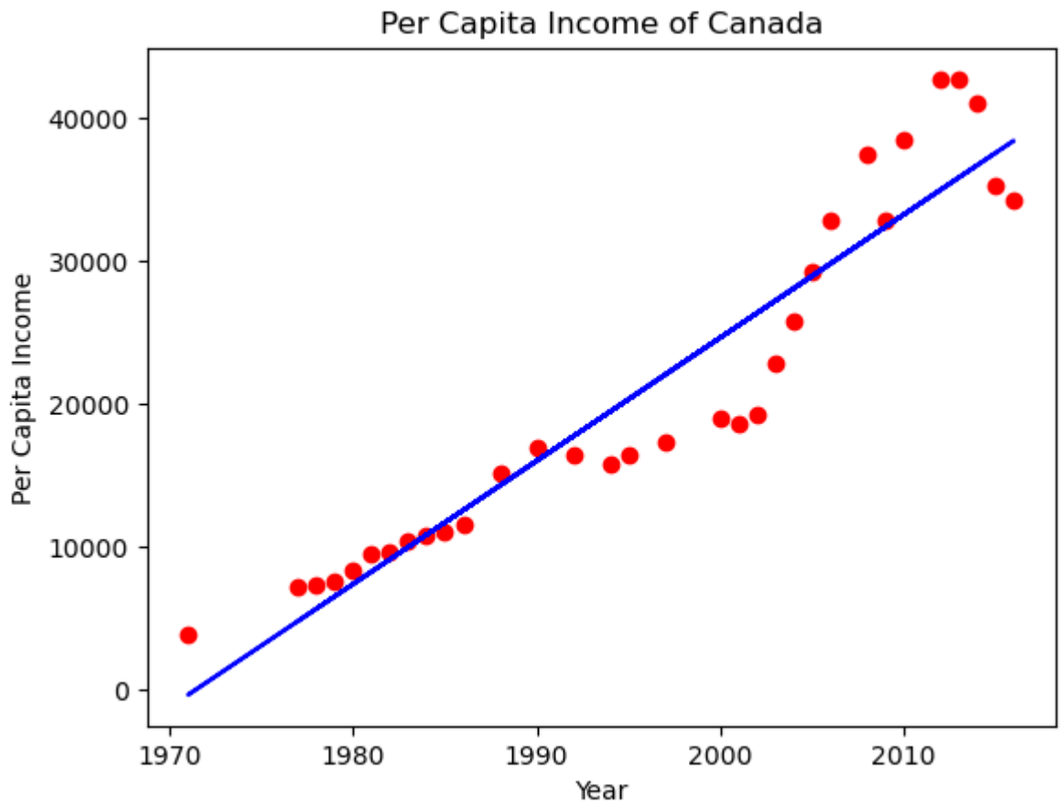
In [10]:
```
#another way to know the score of a linear reg model
reg.score(x_test, y_test)
```

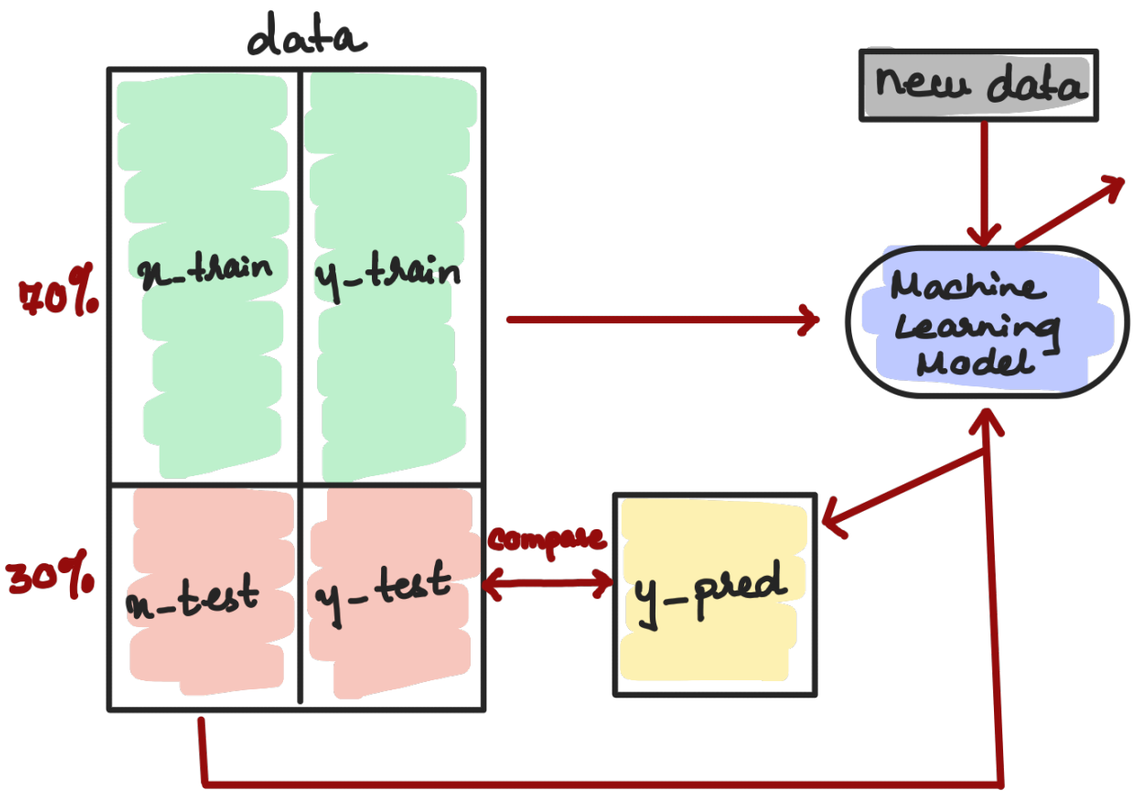
Out[10]:
```
0.8433026110551844
```

In [11]:
```
reg.predict([[2022],[2023]])
```

Out[11]:
```
array([43542.56088317, 44404.09307038])
```

In [14]:
```
plt.scatter(x_train, y_train, color='red')
plt.plot(x_train, reg.predict(x_train),color='blue')
plt.title("Per Capita Income of Canada")
plt.xlabel('Year')
plt.ylabel('Per Capita Income')
plt.show()
```





Data Splitting & using in the Machine Learning model.

@Shreyan

Shreyan Basu Ray

OPENSOURCE MAINTAINER @ SAGE.AI

SUPPORT ME → github.com/sponsors/Shreyan1