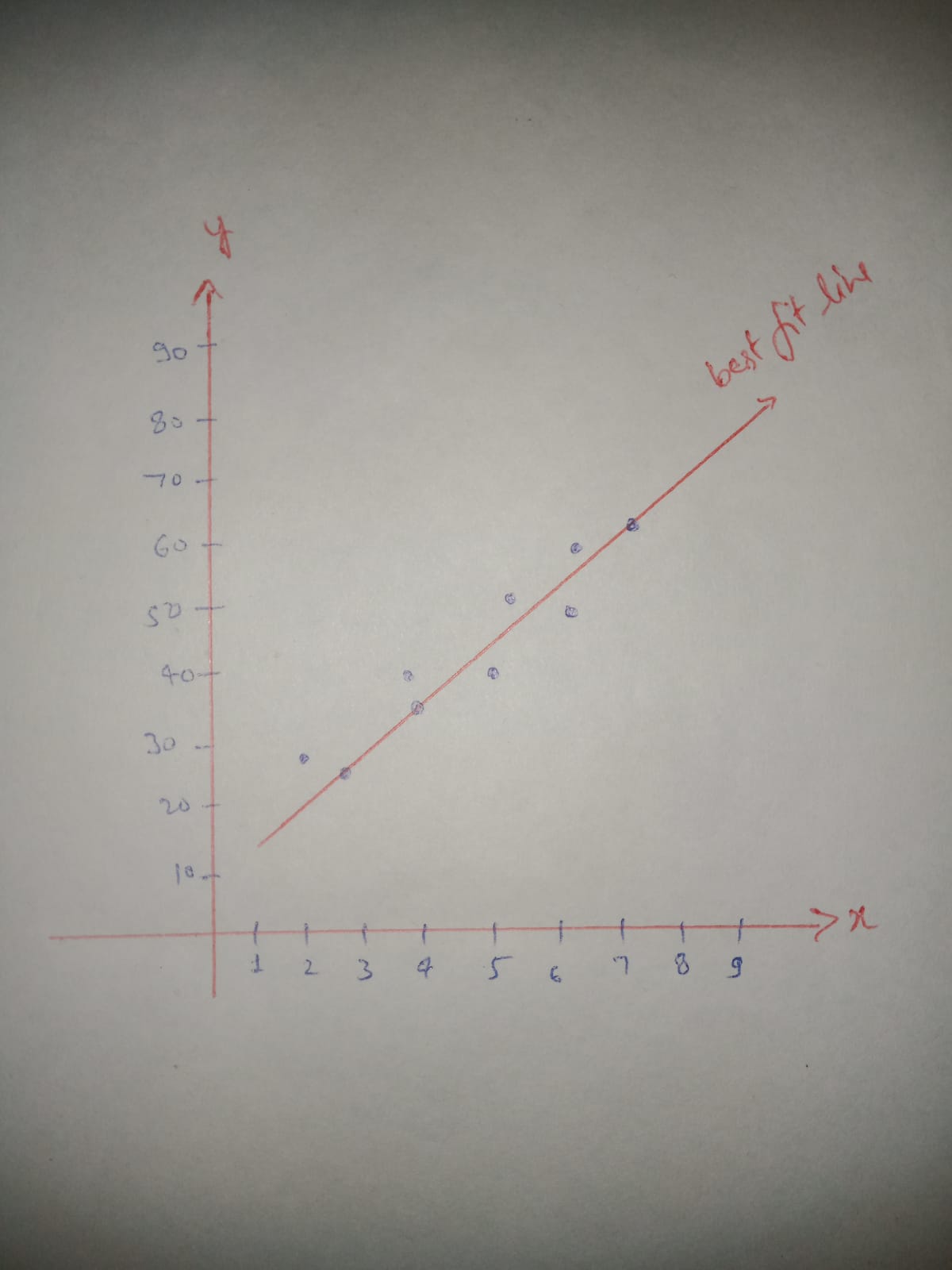
**Linear Regression-Supervised Learning**

**What is Linear Regression?**

Linear regression is a statistical method used for modeling the relationship between a dependent variable (also known as the outcome or response variable) and one or more independent variables (also known as predictors or explanatory variables). The goal of linear regression is to find the best-fitting line through a set of data points, where the line is defined by an equation of the form y = mx + b, where y is the dependent variable, x is the independent variable, m is the slope of the line, and b is the y-intercept. Linear regression can be used for both simple linear regression (one independent variable) and multiple linear regression (more than one independent variable).

****

**Step-(1) Importing Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

# Step-(2) Loading train and test dataset into pandas data frame.

train\_df = pd.read\_csv("train.csv")

*#Drop null values*

train\_df = train\_df.dropna()

train\_df.head()

test\_df = pd.read\_csv("test.csv")

*# Drop null values*

test\_df = test\_df.dropna()

test\_df.head()

# Step-(3) Selection of independent and and dependent variable.

We selected the columns in your data frame that we want to use for the x and y axis. For example, if you have a column called 'x' that represents the independent variable and a column called 'y' that represents the dependent variable, you can select those columns like this:

train\_x = train\_df['x']

train\_y = train\_df['y']

test\_x = test\_df['x']

test\_y = test\_df['y']

# Step-(4) Visualizing the training data.

To draw a linear graph using your data frame, we use the popular data visualization library in Python called Matplotlib. We imported it above.

Now we use the plt.scatter( ) function to plot the data points, and the plt.plot( ) function to plot the line of best fit.

We also use the numpy.polyfit( ) function to fit a line to the data points and get the slope and y-intercept of the line of best fit.

coefficients = np.polyfit(train\_x, train\_y, 1)

m, b = coefficients

plt.scatter(train\_x, train\_y)

plt.plot(train\_x, m\*train\_x + b)

plt.xlabel('train\_x')

plt.ylabel('train\_y')

plt.show()

**Step-(5)Visualizing test data**

coefficients = np.polyfit(test\_x, test\_y, 1)

m, b = coefficients

plt.scatter(test\_x, test\_y)

plt.plot(test\_x, m\*test\_x + b)

plt.xlabel('test\_x')

plt.ylabel('test\_y')

plt.show()

**Step-(6)Model Creation, training, and testing.**

To create a linear regression model and train and test the data using your data frame, we can use the scikit-learn library in Python. The first step is to import the library and the specific model you want to use.

For example, we use the LinearRegression class from the sklearn.linear\_model module.

from sklearn.linear\_model import LinearRegression

Create an instance of the model.

model = LinearRegression()

Now, we use the fit() method to train the model on the training data.

train\_x = train\_x.values.reshape(-1, 1)

test\_x = test\_x.values.reshape(-1, 1)

model.fit(train\_x, train\_y)

**Step-(7)Check the coefficients of the model and the intercept using following command.**

print("Coefficients: ",model.coef\_)

print("Intercept: ",model.intercept\_)

**Our model is trained, now we can use the predict() method to make predictions on the test data.**

y\_pred = model.predict(test\_x)

**Step-(8)Evaluating model performance.**

We can evaluate the performance of the model by comparing the predicted values with the actual values. There are many evaluation metrics such as  mean\_absolute\_error, mean\_squared\_error or r2\_score.

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

print("Mean Absolute Error: ",mean\_absolute\_error(test\_y, y\_pred))

print("Mean Squared Error: ",mean\_squared\_error(test\_y, y\_pred))

print("R2 Score: ",r2\_score(test\_y, y\_pred))

# Step-(9) Visualizing model performance.

We can also visualize the results by plotting the test data points and the predicted line using the same approach as before.

plt.scatter(test\_x, test\_y)

plt.plot(test\_x, y\_pred, color='r')

plt.xlabel('x')

plt.ylabel('y')

plt.show()