BARI ANKIT (56)

Exp – 2 : simple linear regression and multiple linear regression

Code :

```
import numpy as np


1) simple linear regression

class LinearRegression:
  def __init__ (self):
    self.b_0 = 0
    self.b_1 = 0


  def fit (self, X, y):
    X_mean = np.mean (X)
    y_mean = np.mean (y)
    ssxy, ssx = 0, 0
    for _ in range (len (X)):
      ssxy += (X[_]-X_mean)*(y[_]-y_mean)
      ssx += (X[_]-X_mean)**2


    self.b_1 = ssxy / ssx
    self.b_0 = y_mean - (self.b_1*X_mean)
    return self.b_0, self.b_1


  def predict (self, X):
    y_hat = self.b_0 + (X * self.b_1)
    return y_hat
```

```
if __name__ == '__main__':
   X = np.array ([45, 25, 56, 35, 34], ndmin=2)
   X = X.reshape(5, 1)
   y = np.array ([10, 15, 20, 30, 25])
   model = LinearRegression ()
   model.fit (X, y)
   y_pred = model.predict ([50])
   print (y_pred)
```

Output :

[18.92348754]

2) multiple linear regression

Code:

```
import numpy as np

class LinearRegression:
   def __init__ (self):
      self.params = np.zeros(int(np.random.random()), float)[:,np.newaxis]

   def fit (self, X, y):
      bias = np.ones (len (X))
      X_bias = np.c_[bias, X]
```

```
        lse = (np.linalg.inv (np.transpose(X_bias) @ X_bias) @ np.transpose (X_bias)) @ y
        self.params = lse
        return self.params

    def predict (self, X):
        bias_testing = np.ones (len (X))
        X_test = np.c_[bias_testing, X]
        y_hat = X_test @ self.params
        return y_hat

if __name__ == '__main__':
    X = np.array ([
        [1, 4],
        [2, 5],
        [3, 8],
        [4, 2]
    ])

    y = np.array ([1, 6, 8, 12])

    model = LinearRegression ()
    parameters = model.fit (X, y)
    print (f'The parameters for the model are : {parameters}')

    y_pred = model.predict ([[5, 3]])
    print (f'The predicted outcome is : {y_pred}')
```

Output :

The parameters for the model are : [-1.69945355 3.48360656 -0.05464481]

The predicted outcome is : [15.55464481]