Manyam Avinash

187233

Lab-6

Assignment-5

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

Q1 Perform linear regression from scratch without any library on the dataset, and also calculate square sum of error, slope, and intercepts of the best fitting line, show the graphs. Show the graph error graph.

In [2]:

```
data=pd.read_csv('linear_regression.csv')
x=np.array(data[['square_feet']])
y=np.array(data[['house_price']])
n=len(data)
test_data=data.sample(n)
data
```

Out[2]:

	square_feet	house_price
0	100	2750
1	150	4300
2	200	5453
3	250	7000
4	300	8241
5	350	9450
6	400	10665
7	450	12490
8	500	13680
9	550	15140
10	110	2800
11	120	3020
12	130	3500
13	140	4500
14	160	4511
15	170	4320
16	180	4908
17	190	5279
18	210	5698
19	220	5903
20	230	6709
21	240	6608

	square_feet	house_price
21	240	6698
22	260	7238
23	270	7894
24	280	7902
25	290	8299
26	310	8730
27	320	8897
28	330	9094
29	340	9321
30	360	9865
31	370	9304
32	380	10037
33	390	10458
34	410	10980
35	420	10870
36	430	11008
37	44	12004
38	460	13540
39	470	12904
40	480	12806
41	490	13809
42	510	14236
43	515	14907
44	525	14603
45	530	15298
46	538	16344

In [3]:

```
def gradient_descent(x,y):
    theta0=-70;
    theta1=27.2
    alpha=0.000001
    for i in range(1000):
        y_estimated=theta0+(theta1*x)
        cost=(1/n)*sum((y-y_estimated)**2)
        theta0=theta0-(alpha*(-(2/n)*sum(y-y_estimated)))
        theta1=theta1-(alpha*(-(2/n)*sum(x*(y-y_estimated))))

    print(test_data)
    test_data['house_price']=theta0+(theta1*test_data['square_feet'])
    print(test_data)
```

In [5]:

```
gradient_descent(x,y)
```

```
    square_feet   house_price
8           500  13911.098870
43          515  14330.520030
24          280   7759.588532
0           100   2726.534619
44          525  14610.134136
27          320   8878.044957
42          510  14190.712976
18          210   5802.289788
25          290   8039.202638
```

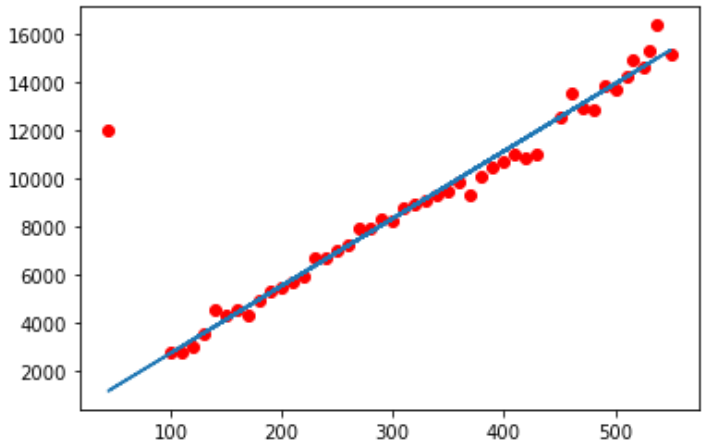
19	220	6081.903895
22	260	7200.360320
45	530	14749.941189
38	460	12792.642445
11	120	3285.762832
33	390	10835.343701
46	538	14973.632474
30	360	9996.501382
35	420	11674.186020
29	340	9437.273170
31	370	10276.115489
9	550	15309.169401
2	200	5522.675682
10	110	3006.148726
36	430	11953.800126
14	160	4404.219257
23	270	7479.974426
34	410	11394.571914
17	190	5243.061576
13	140	3844.991044
3	250	6920.746213
41	490	13631.484764
7	450	12513.028339
16	180	4963.447470
15	170	4683.833363
6	400	11114.957807
12	130	3565.376938
21	240	6641.132107
5	350	9716.887276
4	300	8318.816745
20	230	6361.518001
37	44	1160.695624
40	480	13351.870658
39	470	13072.256551
32	380	10555.729595
26	310	8598.430851
28	330	9157.659064
1	150	4124.605151

	square_feet	house_price
8	500	13911.098870
43	515	14330.520030
24	280	7759.588532
0	100	2726.534619
44	525	14610.134136
27	320	8878.044957
42	510	14190.712976
18	210	5802.289788
25	290	8039.202638
19	220	6081.903895
22	260	7200.360320
45	530	14749.941189
38	460	12792.642445
11	120	3285.762832
33	390	10835.343701
46	538	14973.632474
30	360	9996.501382
35	420	11674.186020
29	340	9437.273170
31	370	10276.115489
9	550	15309.169401
2	200	5522.675682
10	110	3006.148726
36	430	11953.800126
14	160	4404.219257
23	270	7479.974426
34	410	11394.571914
17	190	5243.061576
13	140	3844.991044
3	250	6920.746213
41	490	13631.484764
7	450	12513.028339
16	180	4963.447470

```
15          170   4683.833363
6           400  11114.957807
12          130   3565.376938
21          240   6641.132107
5           350   9716.887276
4           300   8318.816745
20          230   6361.518001
37           44   1160.695624
40          480  13351.870658
39          470  13072.256551
32          380  10555.729595
26          310   8598.430851
28          330   9157.659064
1           150   4124.605151
```

In [7]:

```
plt.scatter(x,y,c="red")
plt.plot(test_data['square_feet'],test_data['house_price'])
plt.show()
```

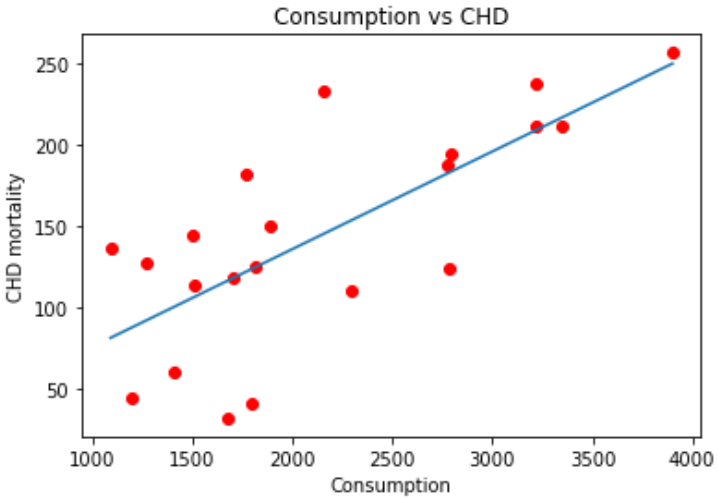


Q2

In [8]:

```
data = pd.DataFrame([
 [3900, 256.9],
 [3350, 211.6],
 [3220, 238.1],
 [3220, 211.8],
 [2790, 194.1],
 [2780, 124.5],
 [2770, 187.3],
 [2290, 110.5],
 [2160, 233.1],
 [1890, 150.3],
 [1810, 124.7],
 [1800, 41.2],
 [1770, 182.1],
 [1700, 118.1],
 [1680, 31.9],
 [1510, 114.3],
 [1500, 144.9],
 [1410, 59.7],
 [1270, 126.9],
 [1200, 43.9],
 [1090, 136.3]
], columns = ['consumption', 'chd'])
data.head()
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
lr = LinearRegression()
lr.fit(X, y)
y_pred = lr.predict(X)
```

```
plt.scatter(X, y, color = 'red')
plt.plot(X, y_pred)
plt.xlabel('Consumption')
plt.ylabel('CHD mortality')
plt.title('Consumption vs CHD')
plt.show()
print("theta_0: {}\ntheta_1: {}".format(lr.intercept_, lr.coef_))
```




```
theta_0: 15.77114669915062
theta_1: [0.06009767]
```

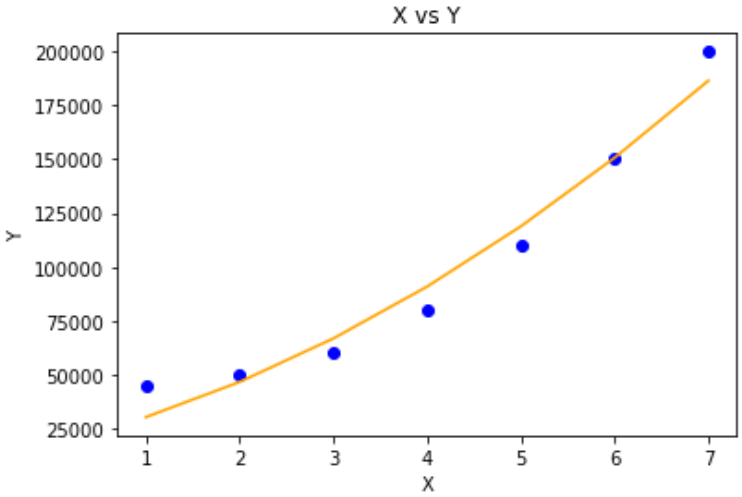
Q3 Perform polynomial regression from scratch without using library on the dataset. Choose dataset from internet. Take the polynomial degree as 3. Show the all graphs, and values.

In [10]:

```
class PolynomailRegression():
    def __init__(self, degree, learning_rate, iterations):
        self.degree = degree
        self.learning_rate = learning_rate
        self.iterations = iterations
    def transform(self, X):
        X_transform = np.ones((self.m, 1))
        j = 0
        for j in range(self.degree + 1):
            if j != 0:
                x_pow = np.power(X, j)
                X_transform = np.append(X_transform, x_pow.reshape(-1, 1), axis=1)
        return X_transform
    def normalize(self, X):
        X[:, 1:] = (X[:, 1:] - np.mean(X[:, 1:], axis=0)) / np.std(X[:, 1:], axis=0)
        return X
    def fit(self, X, Y):
        self.X = X
        self.Y = Y
        self.m, self.n = self.X.shape
        self.W = np.zeros(self.degree + 1)
        X_transform = self.transform(self.X)
        X_normalize = self.normalize(X_transform)
        for i in range(self.iterations):
            h = self.predict(self.X)
            error = h - self.Y
            self.W = self.W - self.learning_rate * (1 / self.m) * np.dot(X_normalize.T,
error)
        return self
    def predict(self, X):
        X_transform = self.transform(X)
        X_normalize = self.normalize(X_transform)
        return np.dot(X_transform, self.W)


def main():
    X = np.array([[1], [2], [3], [4], [5], [6], [7]])
```

```
    Y = np.array([45000, 50000, 60000, 80000, 110000, 150000, 200000])
    model = PolynomailRegression(degree=2, learning_rate=0.01, iterations=500)
    model.fit(X, Y)
    Y_pred = model.predict(X)
    plt.scatter(X, Y, color='blue')
    plt.plot(X, Y_pred, color='orange')
    plt.title('X vs Y')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
if __name__ == "__main__":
    main()
```



In [ ]: