In [1]:

```
# Import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.preprocessing import LabelEncoder ,OneHotEncoder
from sklearn.compose import ColumnTransformer
import joblib
```

Task # 1

In dataset "Fish.csv", Take Species, Length, Width as Input and predict its height using Linear Regression. You can validate your model through Descent Gradient approach to check results.

In [2]:

```
fish_df = pd.read_csv("DataSets/Fish.csv")
fish_df.head()
```

Out[2]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

cLEAN AND PROCESS THE DATA

In [3]:
```
fish_df.isna().sum()    # so no need to clean
```

Out[3]:
```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

In [4]:
```
fish_df.drop_duplicates().shape
```

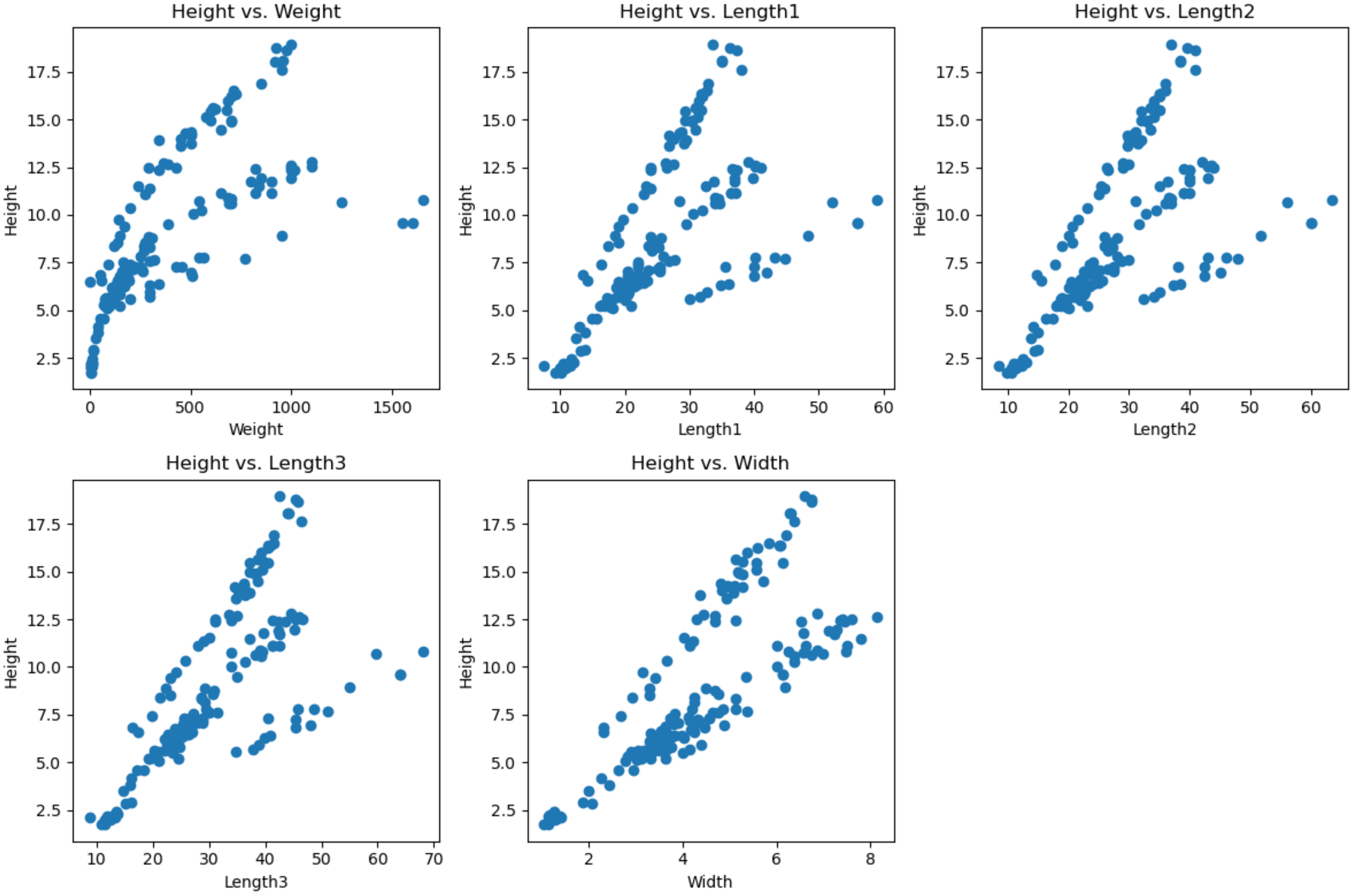
Out[4]:
```
(159, 7)
```

In [ ]:
```
```

In [5]:

```
import matplotlib.pyplot as plt
import pandas as pd

fish_df = pd.read_csv("DataSets/Fish.csv")
height = fish_df['Height']
weight = fish_df['Weight']
length1 = fish_df['Length1']
length2 = fish_df['Length2']
length3 = fish_df['Length3']
width = fish_df['Width']

# Set the figure size
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

# Plot Height vs. Weight
axes[0, 0].scatter(weight, height)
axes[0, 0].set_xlabel('Weight')
axes[0, 0].set_ylabel('Height')
axes[0, 0].set_title('Height vs. Weight')

# Plot Height vs. Length1
axes[0, 1].scatter(length1, height)
axes[0, 1].set_xlabel('Length1')
axes[0, 1].set_ylabel('Height')
axes[0, 1].set_title('Height vs. Length1')

# Plot Height vs. Length2
axes[0, 2].scatter(length2, height)
axes[0, 2].set_xlabel('Length2')
axes[0, 2].set_ylabel('Height')
axes[0, 2].set_title('Height vs. Length2')

# Plot Height vs. Length3
axes[1, 0].scatter(length3, height)
axes[1, 0].set_xlabel('Length3')
axes[1, 0].set_ylabel('Height')
axes[1, 0].set_title('Height vs. Length3')

# Plot Height vs. Width
axes[1, 1].scatter(width, height)
axes[1, 1].set_xlabel('Width')
axes[1, 1].set_ylabel('Height')
axes[1, 1].set_title('Height vs. Width')
```

```
# Remove empty subplot
axes[1, 2].remove()

plt.tight_layout()
plt.show()
```



Height vs. Weight

Height vs. Length1

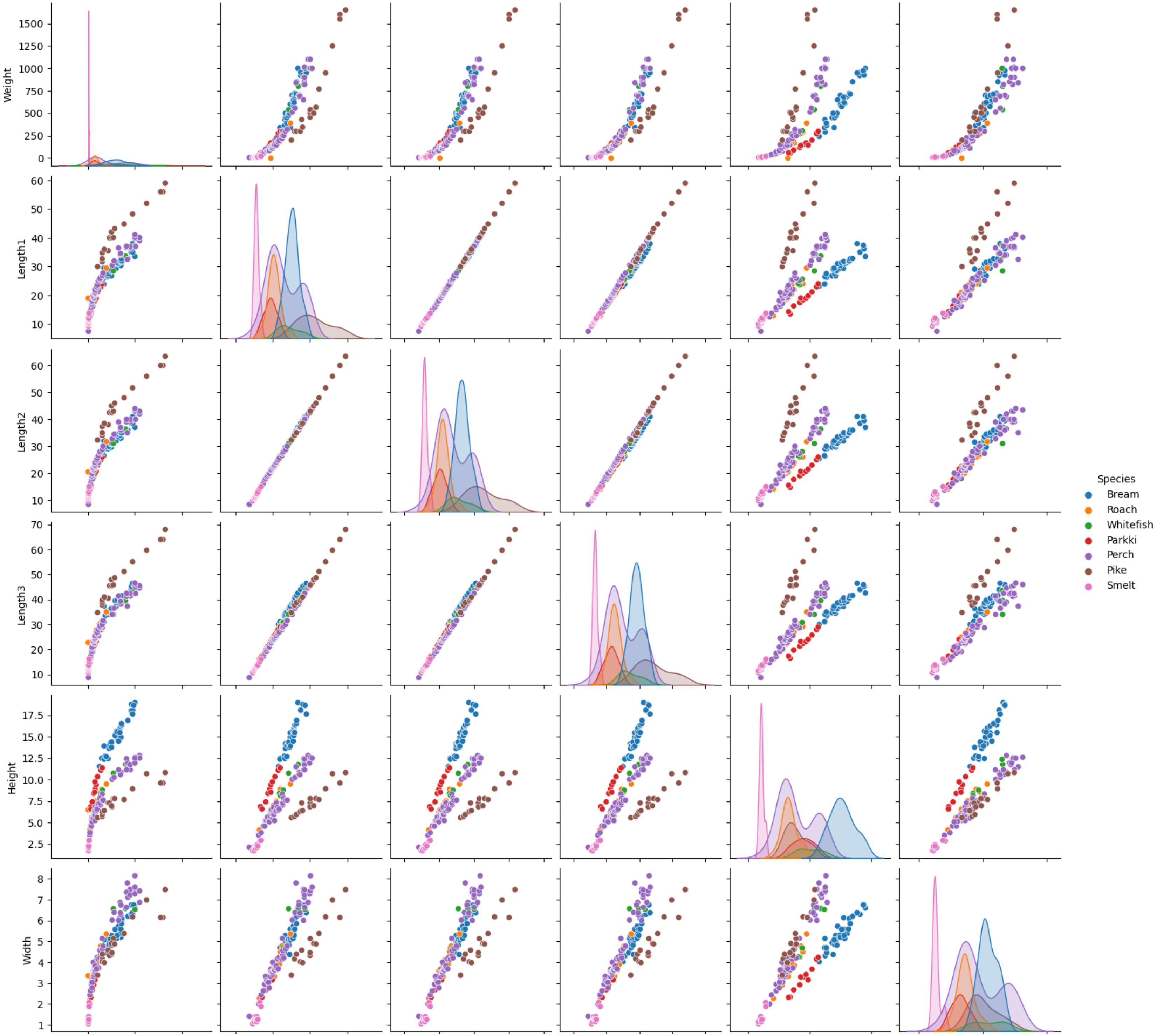
Height vs. Length2

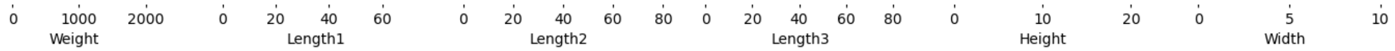
Height vs. Length3

Height vs. Width

In [6]:

```
import seaborn as sns
sns.pairplot(data=fish_df,hue="Species");
```


Weight
1500
1250
1000
750
500
250
0
Length1
60
50
40
30
20
10
Length2
60
50
40
30
20
10
Length3
70
60
50
40
30
20
10
Height
17.5
15.0
12.5
10.0
7.5
5.0
2.5
Width
8
7
6
5
4
3
2
1
Species
Bream
Roach
Whitefish
Parkki
Perch
Pike
Smelt



In [ ]:

```
```

First Apply basic model to evaluate the Dataset

In [7]:

```
x = fish_df.drop("Height",axis=1)
y = fish_df["Height"]
x["Species"].value_counts()
```

Out[7]:

```
Perch        56
Bream        35
Roach        20
Pike         17
Smelt        14
Parkki       11
Whitefish     6
Name: Species, dtype: int64
```

In [8]:

```
category = ["Species"]
encode = OneHotEncoder()

transf = ColumnTransformer([("encode",encode,category)],remainder="passthrough")
transformed_x = transf.fit_transform(x)
transformed_x
```

Out[8]:

```
array([[ 1.    ,  0.    ,  0.    , ..., 25.4   , 30.    ,  4.02  ],
       [ 1.    ,  0.    ,  0.    , ..., 26.3   , 31.2   ,  4.3056],
       [ 1.    ,  0.    ,  0.    , ..., 26.5   , 31.1   ,  4.6961],
       ...,
       [ 0.    ,  0.    ,  0.    , ..., 13.    , 13.8   ,  1.2558],
       [ 0.    ,  0.    ,  0.    , ..., 14.3   , 15.2   ,  2.0672],
       [ 0.    ,  0.    ,  0.    , ..., 15.    , 16.2   ,  1.8792]])
```

In [9]:

```
x_train ,x_test ,y_train ,y_test = train_test_split(transformed_x,y,test_size=0.2,random_state=42)
```

In [10]:

```
model = LinearRegression()
model.fit(x_train,y_train)
```

Out[10]:

```
▾ LinearRegression

LinearRegression()
```

In [11]:

```
model.score(x_test,y_test)
```

Out[11]:

```
0.980397551332076
```

In [12]:

```
y_pred = model.predict(x_test)

pd.DataFrame({"y_test":y_test ,"y_pred":y_pred})
```

Out[12]:

	y_test	y_pred
78	5.1992	4.999679
155	2.4300	2.130787
128	5.5680	4.321885
55	8.3804	7.996653
94	5.2185	6.162787
29	18.9570	16.658972
147	1.7284	1.769991
51	7.0866	7.286182
98	6.7334	6.927047
141	10.6863	10.812777
19	14.4738	15.310182
60	12.3540	11.922111
15	15.4380	15.008784
65	8.8928	9.002424
24	16.2405	15.410072
30	18.0369	16.536581
126	12.6040	13.274712
101	7.1680	7.052905
96	7.2930	6.351509
16	14.8604	14.789092
151	2.2139	2.109468
18	15.6330	14.673377
12	13.7592	13.614805
9	14.2266	14.270440
31	18.0840	16.488222
125	12.5125	12.380040

	y_test	y_pred
95	6.2750	6.317563
56	8.1454	8.073877
145	1.7388	1.572797
152	2.2139	1.983054
135	6.8250	6.625377
76	4.5880	4.732026

Gradient Decsent Method

In [13]:

```
import numpy as np
cost = float('inf')
def gradient_Descent(X,Y):
    m_curr = c_curr =0
    iteration = 10000
    LearningRate = 0.00001
    n = len(X)
    cost = float('inf')
    i=0
    for i in range(iteration):

        cost_prev_val = cost
        y_predicted = (m_curr * X) + c_curr
        cost = 1/n * np.sum([val**2 for val in (Y - y_predicted)])
        mp =-(2/n) * np.sum(X*(Y-y_predicted))

        cp = -(2/n) * np.sum(Y-y_predicted)
        m_curr = m_curr - LearningRate * mp
        c_curr = c_curr - LearningRate * cp

        print("m_curr =" , m_curr)
        print("c_curr =" , c_curr)
        print("cost =" , cost)
        print("\nitteration =" , i)
```

In [19]:

```
X =np.array([4 ,8,9,10,12,14,15])
Y =np.array([12 ,15,17,20,25,30,35])
```

In [20]:
```
gradient_Descent(X,Y)
```

```

itteration = 9977
m_curr = 2.122044512011252
c_curr = 0.18312784064554202
cost = 4.609652226846168

itteration = 9978
m_curr = 2.1220445292356866
c_curr = 0.1831276432176868
cost = 4.609652222918709

itteration = 9979
m_curr = 2.1220445464600854
c_curr = 0.18312744579023682
cost = 4.609652218991269

itteration = 9980
m_curr = 2.1220445636844487
c_curr = 0.18312724836319208
cost = 4.609652215063842
```

In [ ]:

Task 2

In dataset "Salary.csv", Take no. of years as Input and Predict Employee Salary. Since its 2D dataset, check and validate model using Linear Regression by Gradient descent approach

LinearRegression model

In [126]:

```


# Import libraries
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score,mean_absolute_error ,mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.preprocessing import LabelEncoder ,OneHotEncoder
from sklearn.compose import ColumnTransformer
import joblib
```

In [127]:

```
Salary_df = pd.read_csv("DataSets/Salary_dataset.csv")
Salary_df.head()
```

Out[127]:

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

In [128]:

```
x = np.array(Salary_df["YearsExperience"]).reshape(-1, 1)
y = np.array(Salary_df["Salary"])
```

In [129]:

```
x_train ,x_test ,y_train ,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

In [130]:

```
model = LinearRegression()
model.fit(x_train,y_train)
```

Out[130]:

```
▾ LinearRegression

LinearRegression()
```

In [131]:
```
model.score(x_test,y_test)
```

Out[131]:
```
0.9414466227178215
```

In [132]:
```
y_pred = model.predict(x_test)

pd.DataFrame({"y_test":y_test ,"y_pred_using_model":y_pred})
```

Out[132]:

	y_test	y_pred_using_model
0	112636.0	115574.622884
1	67939.0	71680.938782
2	113813.0	102499.908470
3	83089.0	75416.571471
4	64446.0	55804.499851
5	57190.0	60474.040713
6	122392.0	122111.980090
7	109432.0	107169.449332
8	56958.0	63275.765230

In [133]:
```
mse = mean_squared_error(x_train,y_train)
print(f"Mean Squared Error: {mse}")


mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
```

```
Mean Squared Error: 5728211930.733333
Mean Absolute Error: 5161.328710400178
```

Gradient Descent

In [136]:

```
import numpy as np
cost = float('inf')
def gradient_Descent(X,Y):
    m_curr = c_curr =0
    iteration = 1000
    LearningRate = 0.01
    n = len(X)

    i=0
    for i in range(iteration):


        y_predicted = (m_curr * X) + c_curr
        cost = 1/n * np.sum([val**2 for val in (Y - y_predicted)])
        mp =-(2/n) * np.sum(X*(Y-y_predicted))

        cp = -(2/n) * np.sum(Y-y_predicted)
        m_curr = m_curr - LearningRate * mp
        c_curr = c_curr - LearningRate * cp

        print("m_curr =" , m_curr)
        print("c_curr =" , c_curr)
        print("cost =" , cost)
        print("\nitteration =" , i)
```

In [137]:

```
gradient_Descent(x_train.reshape(1,-1)[0],y_train.reshape(1,-1)[0])
```

```
itteration = 990
m_curr = 9376.779222394893
c_curr = 24744.284819751472
cost = 28936001.116909843

itteration = 991
m_curr = 9376.606487117215
c_curr = 24745.390240080145
cost = 28935875.072082177

itteration = 992
m_curr = 9376.434543337044
c_curr = 24746.49059521631
cost = 28935750.17971855

itteration = 993
m_curr = 9376.263387427627
c_curr = 24747.585908369394
cost = 28935626.429281704
```

In [138]:

```
m = 9375.586570542466
c =  24752.98764930146
```

In [139]:

```
 y = m*x  + c
```

In [140]:

```
x_values = x_test.reshape(1,-1)[0]

y_values = []

for x in x_values:
    y = m * x + c
    y_values.append(y)

print("Resulting y values:", y_values)
```

```
Resulting y values: [115696.17738356336, 71630.92050201379, 102570.35618480391, 75381.15513023078, 55692.423
332091596, 60380.21661736283, 122259.0879829431, 107258.14947007515, 63192.89258852557]
```

In [141]:

```
pd.DataFrame({"y_test":y_test ,"y_pred_Gradient":y_values})
```

Out[141]:

	y_test	y_pred_Gradient
0	112636.0	115696.177384
1	67939.0	71630.920502
2	113813.0	102570.356185
3	83089.0	75381.155130
4	64446.0	55692.423332
5	57190.0	60380.216617
6	122392.0	122259.087983
7	109432.0	107258.149470
8	56958.0	63192.892589

In [ ]:

In [ ]: