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

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

```
x=df['Head Size(cm^3)'].values
y=df['Brain Weight(grams)'].values
```

```
plt.figure(figsize=(10,6))
plt.scatter(x,y)
plt.show()
```

```
mean_x=np.mean(x)
mean_y=np.mean(y)
```

```
numer=0
denom=0
n=len(x)
for i in range(n):
    numer+=(x[i]-mean_x)*(y[i]-mean_y)
    denom+=(x[i]-mean_x)**2

slope=numer/denom
intercept=mean_y -(slope*mean_x)
print(slope)
print(intercept)
```

```
0.26342933948939945
325.57342104944223
```

```
prediction = []
for i in range(n):
  y_pred = slope* x[i] + intercept
  prediction.append(int(y_pred))
```

```
print(prediction[:5])
```

```
[1514, 1310, 1448, 1320, 1425]
```

```
print(y[:5])
```

```
[1530 1297 1335 1282 1590]
```

```
plt.figure(figsize=(10,6))
plt.scatter(x,y)
plt.plot(prediction,x)
plt.show()
```

```
#erfg
```

```
#Mse
error=0
for i in range(n):
    error+=(prediction[i]-y[i])**2

mse=error/n
print(mse)
```

```
5202.9029535864975
```

```
np.sqrt(mse)
```

```
np.float64(72.13115106239812)
```

Gradien Descent based slope and itercept

```
#Normalisation
from sklearn.preprocessing import MinMaxScaler
```

```
x.shape
```

```
(237,)
```

```
x=np.reshape(x,(-1,1))
y=np.reshape(y,(-1,1))
```

```
x.shape
```

```
(237, 1)
```

```
minmax=MinMaxScaler()
scale_x=minmax.fit_transform(x)
scale_y=minmax.fit_transform(y)
```

```
print(scale_x[:5])
```

```
[[0.88406512]
 [0.50222003]
 [0.7602368 ]
 [0.52146029]
 [0.71879625]]
```

```
def gradientDescent(epochs, alpha):
  gd_slope = 0  # Initialize slope for gradient descent
  gd_intercept = 0  # Initialize intercept for gradient descent

  for i in range(epochs):
    # y_pred using current slope and intercept from gradient descent
    y_pred = scale_x * gd_slope + gd_intercept

    # Calculate the loss (error)
    loss = y_pred - scale_y

    # Calculate gradients for slope and intercept
    # gradSlope: (2/n) * sum((y_pred - y) * x)
    grad_gd_slope = (2/n) * np.sum(loss.T * scale_x)

    # gradIntercept: (2/n) * sum(y_pred - y)
    grad_gd_intercept = (2/n) * np.sum(loss)

    # Update slope and intercept
    gd_slope = gd_slope - alpha * grad_gd_slope
    gd_intercept = gd_intercept - alpha * grad_gd_intercept

  return gd_slope, gd_intercept

# Define epochs and learning rate (alpha)
epochs = 1000
alpha = 0.01

# Call the gradientDescent function and get the final slope and intercept
final_gd_slope, final_gd_intercept = gradientDescent(epochs, alpha)

print(f"Gradient Descent Slope: {final_gd_slope}")
print(f"Gradient Descent Intercept: {final_gd_intercept}")
```

```
Gradient Descent Slope: 1.0475826980038068
Gradient Descent Intercept: 0.009802831410670107
```

```
epochs =1
alpha =0.01
slope,intercept=gradientDescent(epochs,alpha)
```

```
print(slope)
print(intercept)
```

```
1.0305381504565505
0.009643335815338794
```

```
prediction=[]
for i in range(n):
  y_pred=slope*x[i]+intercept
  prediction.append(int(y_pred[0]))
```

```
plt.figure(figsize=(10,6))
```

```
plt.figure(figsize=(10,6))
plt.scatter(x,y, label='Original Data') # Plot original data
plt.plot(x, prediction, color='red', label='Gradient Descent Regression Line') # Plot the regression line on original scale
plt.xlabel('Head Size (cm^3)')
plt.ylabel('Brain Weight (grams)')
plt.title('Head Size vs Brain Weight with Gradient Descent Regression Line')
plt.legend()
plt.show()
```

```
Start coding or generate with AI.
```