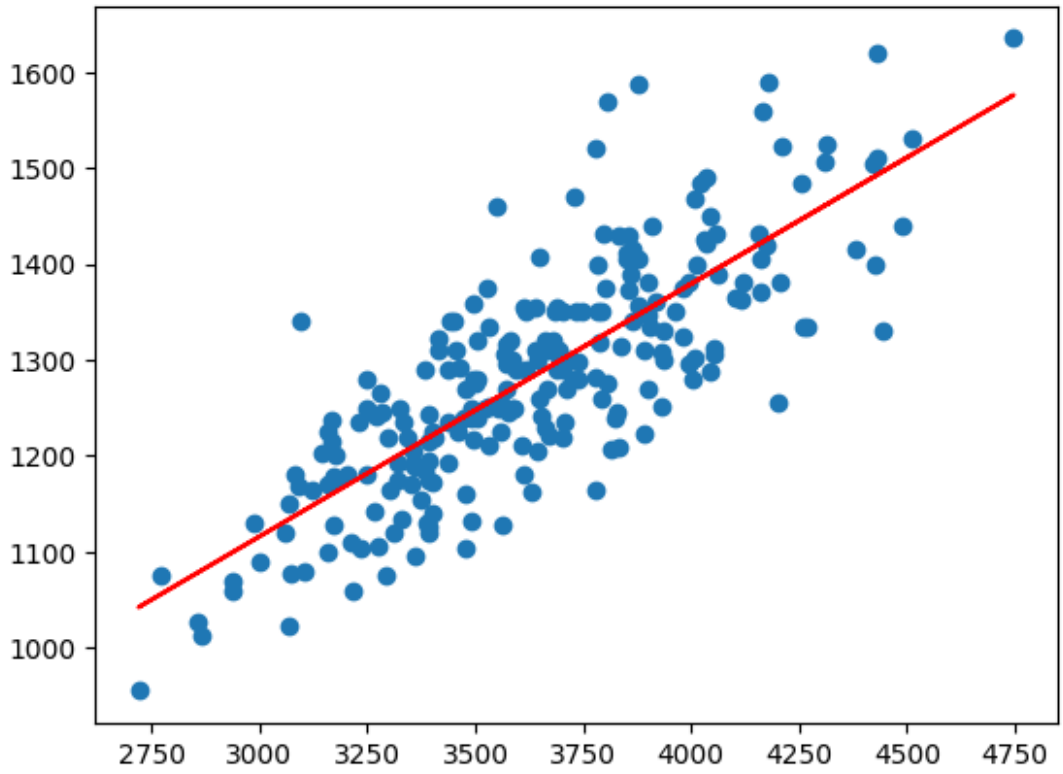
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data=pd.read_csv('headbrain.csv')

x=data['Head Size(cm^3)']
y=data['Brain Weight(grams)']

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

num=0
dem=0

for i in range(len(x)):
    num+=(x[i]-mean_x)*(y[i]-mean_y)
    dem+=(x[i]-mean_x)**2

m=num/dem
c=mean_y-(m*mean_x)

print(c+(m*10))
regression_line = c + m * x

plt.scatter(x,y)
plt.plot(x,regression_line,color="red")
plt.show()
```

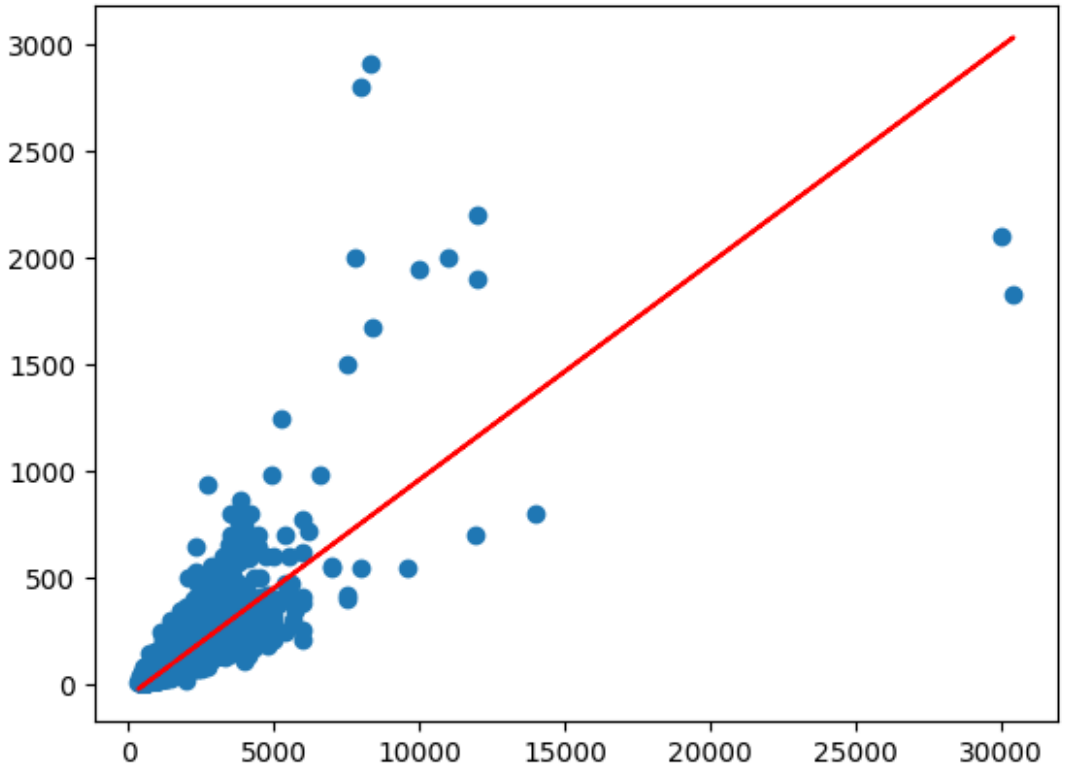
```
328.2077144443362
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data=pd.read_csv('hprice.csv')

x=data['total_sqft_int']
y=data['price']

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

num=0
dem=0

for i in range(len(x)):
    num+=(x[i]-mean_x)*(y[i]-mean_y)
    dem+=(x[i]-mean_x)**2

m=num/dem
c=mean_y-(m*mean_x)

print(c+(m*10))
y_pred = c + m * x

rmse = np.sqrt(np.mean((y-y_pred)**2))
print(f"Root Mean Square Error (RMSE): {rmse}")
```

```
stotal=np.sum((y-mean_y)**2)
s1=np.sum((y-y_pred)**2)
r=1-(s1/stotal)
print(r)

print(c+(m*4500))

plt.scatter(x,y)
plt.plot(x,y_pred,color="red")
plt.show()
```

```
-52.66112846876946
Root Mean Square Error (RMSE): 70.02176909392128
0.6367323019884751
402.88612451570884
```




```
import pandas as pd
import numpy as np

data = pd.DataFrame([
    [43, 30, 63, 33],
    [63, 45, 47, 52],
    [71, 68, 67, 62],
```

```
    [61, 45, 83, 42],
    [81, 66, 84, 42]
], columns=['y', 'x1', 'x2', 'x3'])

data['x1']=data['x1'].fillna(data['x1'].median())
x=data.drop('y',axis=1)
x.insert(0,'B0',1)
y=data['y'].values

bhat=np.linalg.inv(x.T @ x)@(x.T @ y)

print(bhat[0]+bhat[1]*50+bhat[2]*70+bhat[3]*80)
```

```
49.796206730836516
```

```
import pandas as pd
import numpy as np

data = pd.DataFrame([
    [2600, 3, 20, 550000],
    [3000, 4, 15, 565000],
    [3200, np.nan , 18, 610000],
    [3600, 3, 30, 595000],
    [4000, 5, 8, 760000]
], columns=['a', 'b', 'age', 'price'])
data['b']=data['b'].fillna(data['b'].median())
x=data.drop('price',axis=1)
x.insert(0,'B0',1)
y=data['price'].values

bhat=np.linalg.inv(x.T @ x)@(x.T @ y)
print(bhat)
print(bhat[0]+bhat[1]*3000+bhat[2]*3+bhat[3]*40)
```

```
[ 6.56046539e+05  1.98471590e+02 -1.16583739e+05 -1.42677759e+04]
330999.058232676
```

```
import pandas as pd
import numpy as np

data = pd.DataFrame([
    [2600, 3, 20, 550000],
    [3000, 4, 15, 565000],
    [3200, np.nan , 18, 610000],
    [3600, 3, 30, 595000],
```

```
    [4000, 5, 8, 760000]
], columns=['a', 'b', 'age', 'price'])
data['b']=data['b'].fillna(data['b'].median())
x=data.drop('price',axis=1)
x.insert(0,'B0',1)
y=data['price'].values

bhat=np.linalg.inv(x.T @ x)@(x.T @ y)
print(bhat)
print(bhat[0]+bhat[1]*3000+bhat[2]*3+bhat[3]*40)
```

```
[ 6.56046539e+05  1.98471590e+02 -1.16583739e+05 -1.42677759e+04]
330999.058232676
```

```
import pandas as pd
import numpy as np

# Dataset
data = pd.DataFrame([
    [2600, 3, 20, 550000],
    [3000, 4, 15, 565000],
    [3200, np.nan, 18, 610000],
    [3600, 3, 30, 595000],
    [4000, 5, 8, 760000]
], columns=['a', 'b', 'age', 'price'])

# Handle missing values
data['b'] = data['b'].fillna(data['b'].median())

# Create a binary target variable
data['expensive'] = (data['price'] > 600000).astype(int)  # 1 for
Expensive, 0 for Affordable

# Prepare features and target
x = data.drop(['price', 'expensive'], axis=1)
x.insert(0, 'B0', 1)  # Add intercept term
y = data['expensive'].values  # Binary target

# Logistic regression: coefficients calculation
bhat = np.linalg.inv(x.T @ x) @ (x.T @ y)
print("Coefficients (bhat):", bhat)

# Compute the logistic function for a new house
log = bhat[0] + bhat[1] * 5300 + bhat[2] * 3 + bhat[3] * 40

# Define the logistic function
def logs(x):
    return 1 / (1 + np.exp(-x))  # Corrected logistic function
```

```
# Predict probability of being "Expensive"
print(1 if logs(log)>0.5 else 0)
```

```
Coefficients (bhat): [ 5.87780568e+00  1.90511694e-03 -2.14016638e+00
-2.09229320e-01]
1
```

```
import numpy as np

x = np.array([[5, 2], [6, 3], [7, 2], [8, 3], [4, 1]])
y = np.array(["flower", "flower", "shrub", "shrub", "flower"])

def knn(x,y,test,k=3):
    distance=np.linalg.norm(x-test,axis=1)
    sort=distance.argsort()[:k]
    nearest=y[sort]
    return max(set(nearest),key=list(nearest).count)

test=np.array([5,2])
print(knn(x,y,test))
```

```
flower
```

```
import pandas as pd
import numpy as np


def kmean(x,y,k,iterw):
    centroids=[(x[i],y[i]) for i in range(k)]
    for i in range(iterw):
        clusters=[[] for i in range(k)]
    for i in range(len(x)):
            distance=[((x[i]-cx)**2) +((y[i]-cy)**2) for cx,cy in
centroids]
            ci=distance.index(max(distance))
            clusters[ci].append((x[i],y[i]))
            new_cen=[]
    for i,cluster in enumerate(clusters):
        if cluster:
            avgx=(np.sum(p[0] for p in cluster)/len(cluster))
            avgy=(np.sum(p[1] for p in cluster)/len(cluster))
            new_cen.append([avgx,avgy])
        else:
            new_cen.append(centroids[i])
        centroids=new_cen
    return centroids,clusters


x=[185,170,168,179,182,188,180,180,183,180,180,177]
y=[72,56,60,68,72,77,71,70,84,88,67,76]
```

```
k=2
iteration=10
centroid,cluster=kmean(x,y,k,iteration)
print(centroid)
print(cluster)
```

```
[[169.0, 58.0], [181.4, 74.5]]
[[(170, 56), (168, 60)], [(185, 72), (179, 68), (182, 72), (188, 77),
(180, 71), (180, 70), (183, 84), (180, 88), (180, 67), (177, 76)]]
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_15196\3727055364.py:16:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in
the future will give a different result. Use
np.sum(np.fromiter(generator)) or the python sum builtin instead.
  avgx=(np.sum(p[0] for p in cluster)/len(cluster))
C:\Users\User\AppData\Local\Temp\ipykernel_15196\3727055364.py:17:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in
the future will give a different result. Use
np.sum(np.fromiter(generator)) or the python sum builtin instead.
  avgy=(np.sum(p[1] for p in cluster)/len(cluster))
```

```
import pandas as pd
import numpy as np

data = pd.DataFrame({
    'Outlook': ['Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Sunny',
'Overcast', 'Rain', 'Rain', 'Sunny', 'Rain', 'Overcast', 'Overcast',
'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
'Strong'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})

def entro(col):
    prob=col.value_counts(normalize=True)
    return -np.sum(prob*np.log2(prob))

def calcu_info(data,feature,target):
    totale=entro(data[target])
    subr=sum(len(group)/len(data)*entro(group[target]) for _,group in
data.groupby(feature))
    return totale-subr
```

```
def buildtree(data,features,target):
    tree={}
    while(len(features)>0):
        gains={feature:calcu_info(data,feature,target) for feature in
features}
        bestf=max(gains,key=gains.get)

        tree[bestf]={}
        for value,subset in data.groupby(bestf):
            if len(subset[target].unique())==1:
                tree[bestf][value]=subset[target].iloc[0]
            else:
                tree[bestf][value]=subset[target].mode()[0]
        features.remove(bestf)
    return tree



def cal(test,tree):
    if not isinstance(tree,dict):
        return tree
    feature=next(iter(tree))
    brach=tree[feature]
    return cal(test,brach[test[feature]])

features = ['Outlook', 'Temperature', 'Humidity', 'Wind']
target = 'Play'
decision_tree=buildtree(data,features,target)

example_case = {'Outlook': 'Rain', 'Temperature': 'Mild', 'Humidity':
'High', 'Wind': 'Weak'}
print(cal(example_case,decision_tree))
```

```
No
```

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

x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
y = np.array([3, 4, 2, 5, 6, 7, 8, 6, 10, 9, 11, 14, 13, 16, 15])

m=0
b=0
lr=0.01
for i in range(1000):
    y_pred=m*x+b
    error=y-y_pred
    mg=-(2/len(x))*(np.sum(x*error))
    bg=-(2/len(x))*(np.sum(error))
```

```
    m-=lr*mg
    b-=lr*bg

print(f"Final slope (m): {m}, intercept (b): {b}")
plt.scatter(x,y)
plt.plot(x,m*x+b)
plt.show()
```

```
Final slope (m): 0.958065070712213, intercept (b): 0.9333533906954091
```