In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [2]:
```
df=pd.read_csv('height-weight.csv')
```

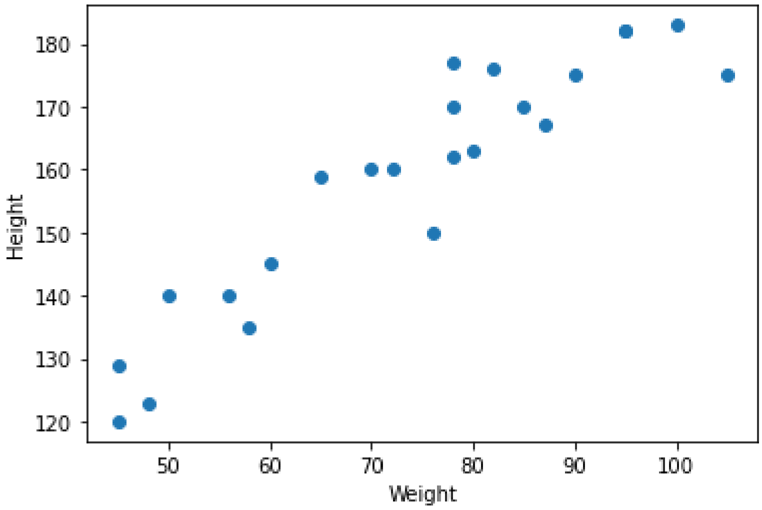
In [3]:
```
df.head()
```

Out[3]:

	Weight	Height
0	45	120
1	58	135
2	48	123
3	60	145
4	70	160

In [7]:
```
## Scatter plot
plt.scatter(df['Weight'],df['Height'])
plt.xlabel("Weight")
plt.ylabel("Height")
```

Out[7]: `Text(0, 0.5, 'Height')`



We used scatter plot because we wanted to find the relation between the height and weight.

From the above graph we can say that the relation in linear

In [9]:
```
## Correlation
df .corr()
```

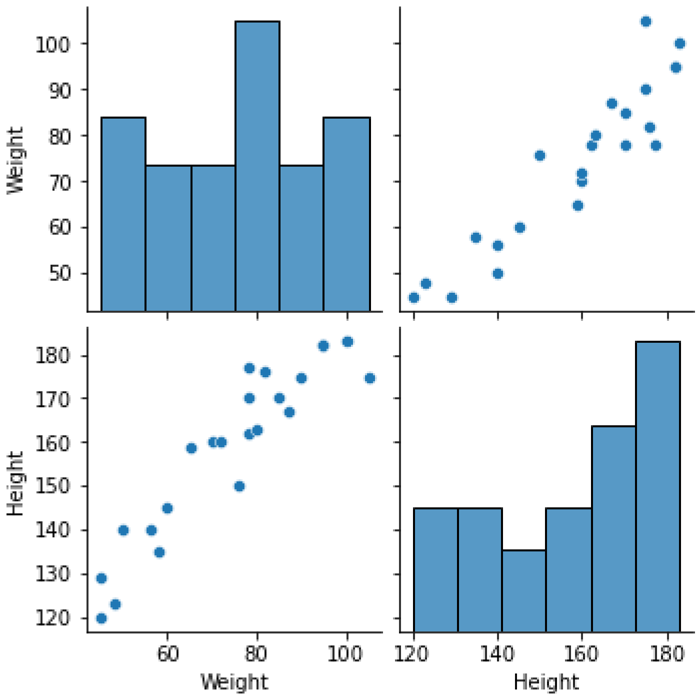
Out[9]:

	Weight	Height
Weight	1.000000	0.931142
Height	0.931142	1.000000

we can note that the correlation between weight and height is 0.93 , which means they are positively correlated.

In [13]:
```
## Seaborn for visualization
import seaborn as sns
sns.pairplot(df)
```

Out[13]: <seaborn.axisgrid.PairGrid at 0x2678f9ec610>



In [18]:
```
## Independent and Dependent Feature
X=df[['Weight']] ## We have to always make sure that our independent featur
y=df['Height'] ## Dependent feature can be in series form or 1-D array
```

In [31]:
```
X_series=df['Weight']
```

In [19]:
```
## Train Test Split
from sklearn.model_selection import train_test_split
```

In [20]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25, random_s
```

In [24]:
```
## Standardizzation
from sklearn.preprocessing import StandardScaler
```

Standardization is very important step.Let's understand it with example. Ley there be two features X(independent in kg) and Y(dependent in cm), so we know that in linear regression we want to reach global minima in the gradient descent, but as the the value of X in big value it will take more time and more optimizations to reach to global minima. This problem can be solved by the Z-score. The formlula is zscore= (Xi- Mean/ SD). What this will do is that after applying this it will make the mean will be zero and SD will be eqal to 1

In [26]:
```
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
```

In [27]:
```
X_test= scaler.transform(X_test)
```

In [28]:
```
X_test
```

Out[28]:
```
array([[ 0.33497168],
       [ 0.33497168],
       [-1.6641678 ],
       [ 1.36483141],
       [-0.45256812],
       [ 1.97063125]])
```

We are using transform() function and not fit.transform() in the testing part, this is because we use the same mean and SD which we got in the training part in the testing. If we use fit.transform() in the testing then there will be new SD and Mean , which we don't want.

In [29]:
```
## Apply Simple Linear Regression
from sklearn.linear_model import LinearRegression
```

In [33]:
```
regression=LinearRegression(n_jobs=-1)
```

In [34]:
```
regression.fit(X_train,y_train)
```

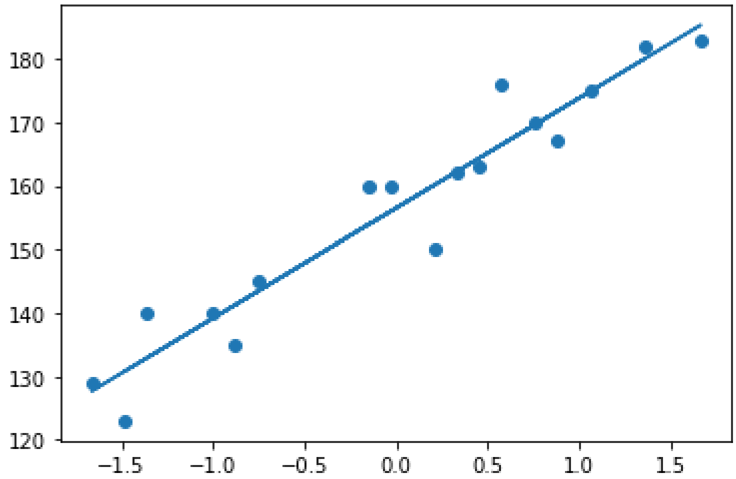
Out[34]:
```
LinearRegression(n_jobs=-1)
```

In [37]:
```
print("Coefficient or slope:",regression.coef_)
print("Coefficient or intercept:",regression.intercept_)
```

```
Coefficient or slope: [17.2982057]
Coefficient or intercept: 156.47058823529412
```

In [39]:
```
## Plot Training data plot best fit line
plt.scatter(X_train, y_train)
plt.plot(X_train,regression.predict(X_train))
```

Out[39]: `[<matplotlib.lines.Line2D at 0x26790353f70>]`



In [40]:
```
## Prediction for test Data
y_pred=regression.predict(X_test)
```

prediction of test data

1.predicted height output= intercept +coef_(Weights) 2.y_pred_test =156.470 + 17.29(X_test)

In [41]:
```
## Performance Metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [42]:
```
mse=mean_squared_error(y_test,y_pred)
mae=mean_absolute_error(y_test,y_pred)
rmse=np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
```

```
114.84069295228699
9.665125886795005
10.716374991212605
```

R square

Formula

R^2 = 1 - SSR/SST

R^2 = coefficient of determination SSR = sum of squares of residuals SST = total sum of square

In [43]:
```
from sklearn.metrics import r2_score
```

In [44]:
```
score=r2_score(y_test,y_pred)
print(score)
```

```
0.7360826717981276
```

Adjusted R2 = 1 – [(1-R2)*(n-1)/(n-k-1)]

where:

R2: The R2 of the model n: The number of observations k: The number of predictor variables

In [45]:
```
#display adjusted R-squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[45]:
```
0.6701033397476595
```

In [46]:
```
## OLS Linear Regression
import statsmodels.api as sm
```

In [47]:
```
model=sm.OLS(y_train,X_train).fit()
```

In [48]:
```
prediction=model.predict(X_test)
print(prediction)
```

```
[  5.79440897   5.79440897 -28.78711691  23.60913442  -7.82861638
  34.08838469]
```

In [49]:
```
print(model.summary())
```

```
                                 OLS Regression Results
=================================================================================
=============
Dep. Variable:                 Height   R-squared (uncentered):
0.012
Model:                            OLS   Adj. R-squared (uncentered):
-0.050
Method:                 Least Squares   F-statistic:
0.1953
Date:                Sat, 28 Sep 2024   Prob (F-statistic):
0.664
Time:                        17:37:57   Log-Likelihood:
-110.03
No. Observations:                  17   AIC:
222.1
Df Residuals:                      16   BIC:
222.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
x1            17.2982     39.138      0.442      0.664     -65.671      10
0.267
==============================================================================
====
Omnibus:                        0.135   Durbin-Watson:
0.002
Prob(Omnibus):                  0.935   Jarque-Bera (JB):
0.203
Skew:                          -0.166   Prob(JB):
0.904
Kurtosis:                       2.581   Cond. No.
1.00
==============================================================================
====

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

C:\Users\Dell\anaconda3\lib\site-packages\scipy\stats\stats.py:1541: UserW
arning: kurtosistest only valid for n>=20 ... continuing anyway, n=17
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [50]:
```
## Prediction For new data
regression.predict(scaler.transform([[72]]))
```

Out[50]:
```
array([155.97744705])
```

In [ ]: