In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

In [3]:
```
data=pd.read_csv("weight-height.csv")
#The pd.read_csv() function is used to read the
#"weight-height.csv" file into a pandas DataFrame named data.
df=data
```

In [4]:
```
data.describe()
#displays various descriptive statistics for the numerical columns in the DataFrame
#such as the count, mean, standard deviation, minimum,
#25th percentile, 50th percentile (median), 75th percentile, and maximum values.
```

Out[4]:

	Height	Weight
count	10000.000000	10000.000000
mean	66.367560	161.440357
std	3.847528	32.108439
min	54.263133	64.700127
25%	63.505620	135.818051
50%	66.318070	161.212928
75%	69.174262	187.169525
max	78.998742	269.989699

In [5]:
```
data.head()
#method used to display the first few rows of a DataFrame
```

Out[5]:

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

In [6]:
```
data.tail()
#is amethod used to display the last few rows of a DataFrame.
```

Out[6]:

	Gender	Height	Weight
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

In [7]:
```
type(data)
 # to determine the data type of the data object.
```

Out[7]: pandas.core.frame.DataFrame

In [8]:
```
data.shape # to determine the dimensions of a DataFrame.
```

Out[8]: (10000, 3)

In [9]:
```
data.info()
# used to display a concise summary of a DataFrame.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Gender  10000 non-null  object 
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

In [10]:
```
column=data
#This creates a new DataFrame with the same structure but
# containing only the rows with zero values.
count=column[column==0].count()
#This effectively gives you the count of occurrences of
#  the value 0 in each column.


print(count) #print the result
```

```
Gender    0
Height    0
Weight    0
dtype: int64
```

In [12]:
```
count=(data["Height"]==22).sum()
#filters the "Height" column of the DataFrame data to only
# include rows where the values are equal to 22
#sums the Boolean values in the filtered Series.
#Since True is treated as 1 and False as 0,
```

```
print(count)
```

```
0
```

In [13]:
```
data.isnull().head()
#for missing values (null values) in the first few rows of a DataFrame.
```

Out[13]:

	Gender	Height	Weight
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False

In [14]:
```
data.info()
# concise summary of a DataFrame's structure and characteristics.
```

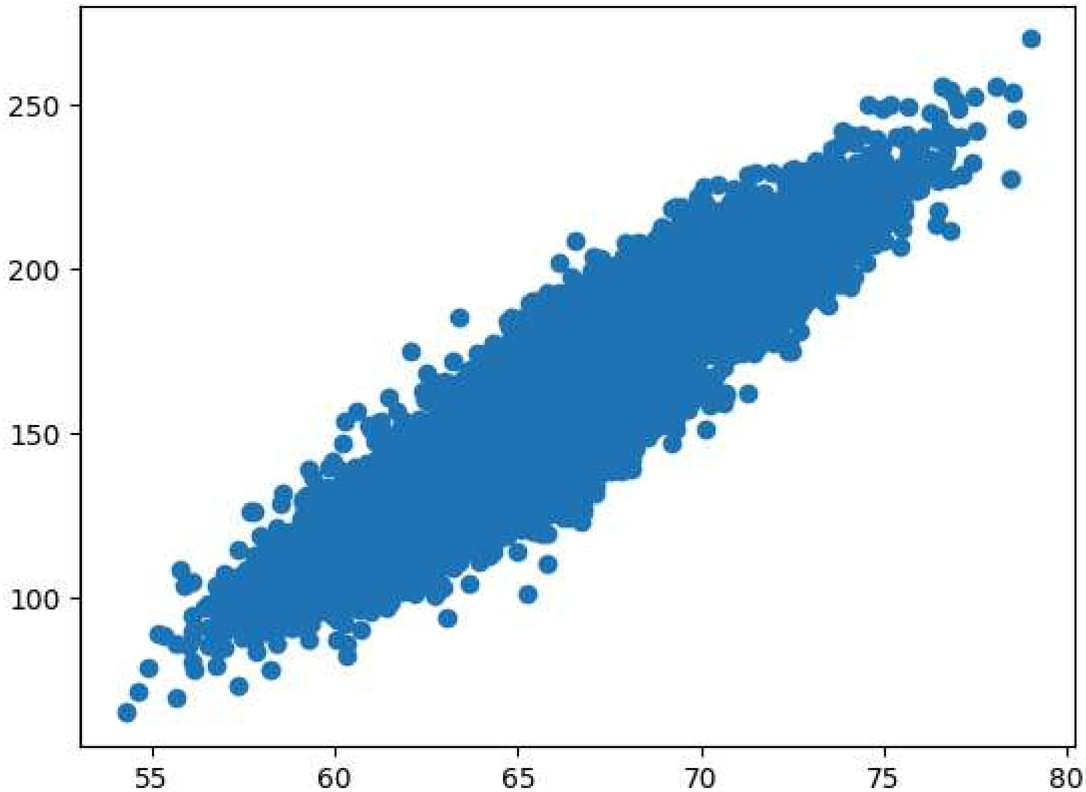
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Gender  10000 non-null  object 
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

In [15]:
```
x=data.iloc[:,1:6]
#data.iloc[:, 1:6] selects columns 1 to 5 (inclusive) from the DataFrame data.
#The iloc indexing method is used to access data by integer position.
#The : notation indicates selecting all rows
# , while 1:6 specifies the desired column range.
y=data.iloc[:,-1:]
#data.iloc[:, -1:] selects the last column from the DataFrame data.
#The -1 index represents the last element in the column range.
#The : notation again indicates selecting all rows.
```

In [16]:
```
x=data["Height"]
y=data["Weight"]
```

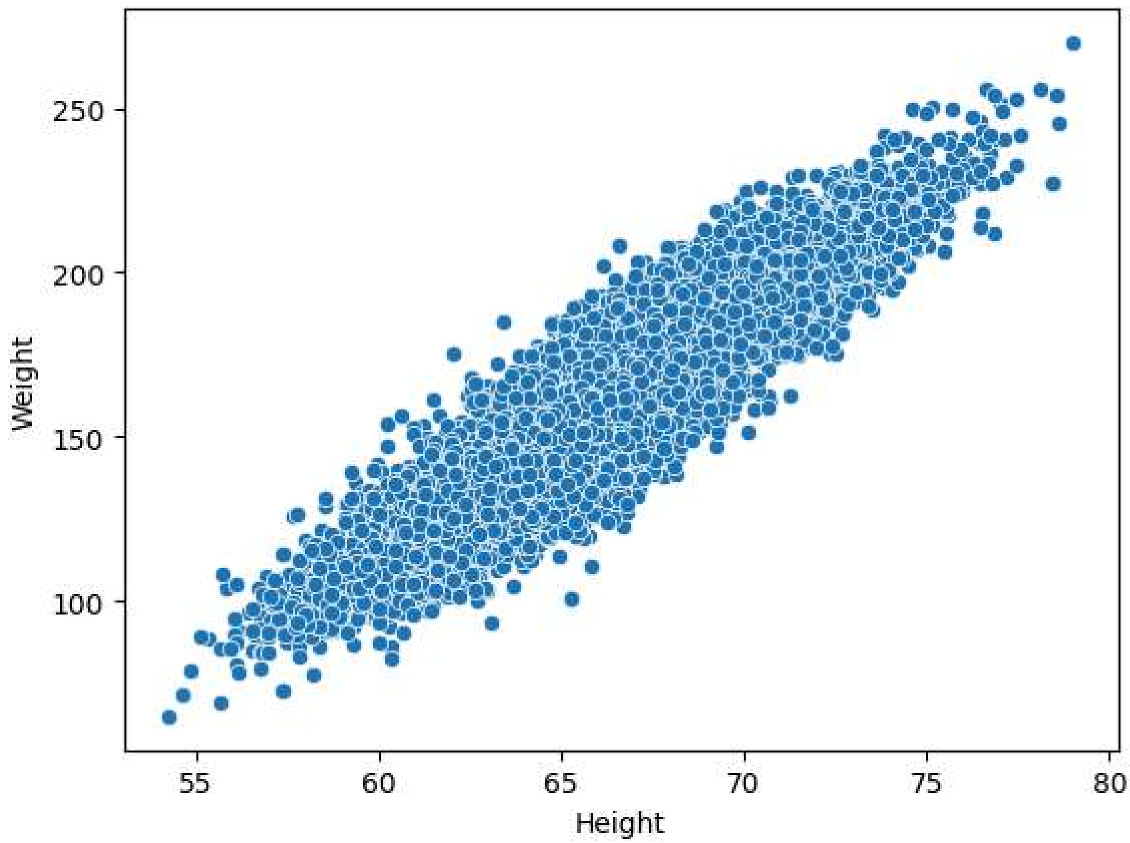
In [17]:
```
plt.plot(x,y,'o')
 #line imports the Matplotlib library and assigns it the alias plt,
# which is commonly used for plotting.
```

Out[17]: `[<matplotlib.lines.Line2D at 0x20f5fa9b580>]`



In [18]:
```
sns.scatterplot(x=x,y=y,data=df)
#using the Seaborn library,
# creates a scatter plot to visualize the relationship
# between the variables x and y from the DataFrame df.
```

Out[18]: <Axes: xlabel='Height', ylabel='Weight'>



In [19]:
```
type(x)
```
Out[19]: pandas.core.series.Series

In [20]:
```
x.shape
```
Out[20]: (10000,)

In [21]:
```
x=x.values
```
In [22]:
```
x=x.reshape(10000 ,1)
```
In [23]:
```
x.shape
```
Out[23]: (10000, 1)

In [24]:
```
type(x)
```
Out[24]: numpy.ndarray

In [26]:
```
x_train, x_test, y_train,y_test=train_test_split(x,y,test_size=0.25)
#x and y are the input features and target variable, respectively.
# test_size=0.25 specifies that 25% of the data
# should be allocated to the testing set,
# while the remaining 75% will be used for training.
print(f"x training dataset:{x_train.shape}")
#prints the shape of the x_train training set,
#showing the number of samples and features.
```

```
print(f"y training dataset: {y_train.shape}")
#prints the shape of the y_train training set,
#showing the number of samples.
print(f"x test dataset : {x_test.shape}")
#prints the shape of the x_test testing set,
# showing the number of samples and features.
print(f"y test dataset : {y_test.shape}")
# prints the shape of the y_test testing set,
#showing the number of samples.
```

```
x training dataset:(7500, 1)
y training dataset: (7500,)
x test dataset : (2500, 1)
y test dataset : (2500,)
```

In [27]:

```
model=LinearRegression()
# This class is used for implementing linear regression models,
#a statistical method used to model the relationship
# between a dependent variable and one or more independent variables.
```

In [28]:

```
model.fit(x_train, y_train)
#The model.fit(x_train, y_train) line calls the
# fit method on the model object.
#This method is responsible for training the
# linear regression model based on the provided data.
#x_train: This represents the training set of input features,
# typically a 2D array where each row represents a
# sample and each column represents a feature.
#y_train: This represents the training set of target variables,
# typically a 1D array where each element corresponds
#to the target value for the corresponding sample in x_train.
```

Out[28]:

```
▾ LinearRegression
LinearRegression()
```

In [29]:

```
model.coef_
#when used with a linear regression model,
#returns a NumPy array containing
# the coefficients (weights) learned by the model during training.
```

Out[29]: array([7.73982037])

In [30]:

```
model.intercept_
#attribute in Python, when used with a linear regression model,
# returns the intercept term of the regression equation.
```

Out[30]: -352.30603025252424

In [32]:

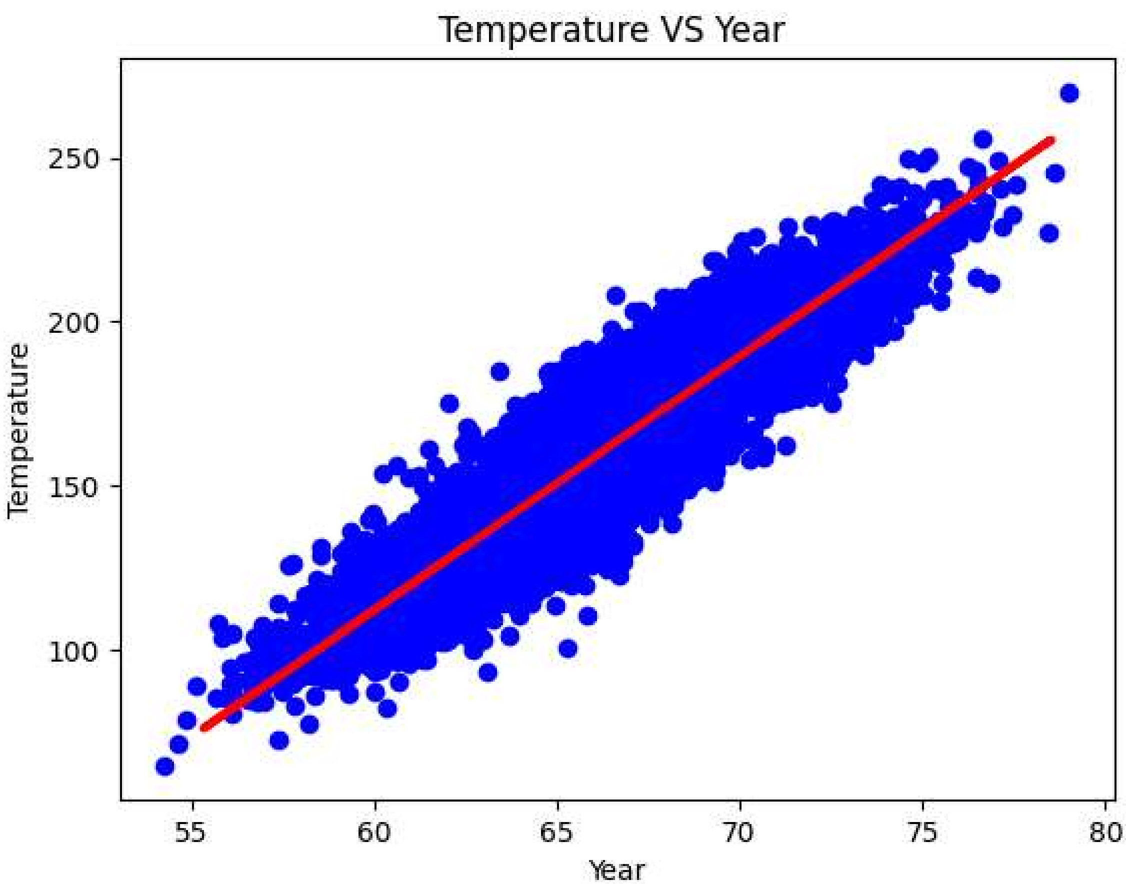
```
y_pred=model.predict(x_test)
#when used with a trained linear regression model
#, predicts the target variable values for a given
# set of input features in the x_test dataset.
```

In [34]:
```
y_pred.shape

#The y_pred.shape attribute in Python returns a tuple
# representing the dimensions of the y_pred array,
# which contains the predicted target variable
# values from a linear regression model.
```

Out[34]: (2500,)

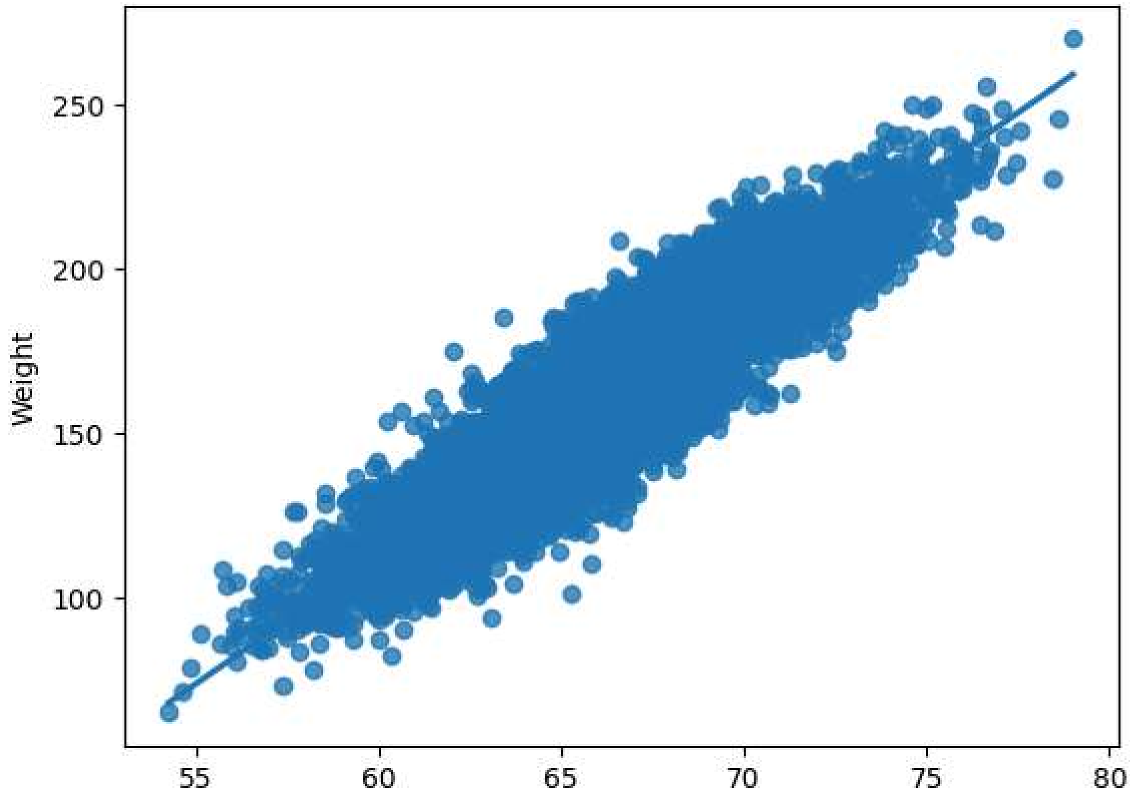
In [35]:
```
plt.scatter(x_train, y_train, color='blue')
# creates a scatter plot where the points represent the
# actual target values (y_train) for each sample in the training set (x_train).
plt.plot(x_test,y_pred, color='red', linewidth=3)
#creates a line plot that shows the predicted target
# values (y_pred) for each sample in the testing set (x_test).
plt.title("Temperature VS Year ")
#sets the title of the plot to "Temperature VS Year ".
# Make sure the title is accurate based on your data.
plt.xlabel("Year")
plt.ylabel("Temperature")
#plt.xlabel("Year") and plt.ylabel("Temperature") set the
# labels for the x-axis and y-axis, respectively.
plt.show()
# displays the generated plot.
```



In [36]:
```
sns.regplot(data=df ,x=x_train, y=y_train)
#using the Seaborn library, creates a regression plot to visualize the relationship
```

```
# between the variables x_train and y_train from the DataFrame df.
```

Out[36]: <Axes: ylabel='Weight'>



In [39]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error,r2_score
# imports the necessary functions from the sklearn.metrics
# module for calculating these metrics.
print(f"MSE: {mean_squared_error(y_test, y_pred)}")
# calculates the MSE between the actual target values
# (y_test) and the predicted values (y_pred) and prints the result.
#MSE measures the average squared
# difference between the predicted and actual values.
print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
#calculates the MAE between the actual target value
# s and the predicted values and prints the result.
#MAE measures the average absolute difference
# between the predicted and actual values.
print(f"R-square: {r2_score(y_test, y_pred)}")
#calculates the R-squared score between the actual target values
# and the predicted values and prints the result.
#R-squared represents the proportion of variance in the
# target variable that is explained by the model.
```

```
MSE: 145.8333794275773
MAE: 9.6272194749545
R-square: 0.8541178086703406
```

In [ ]: