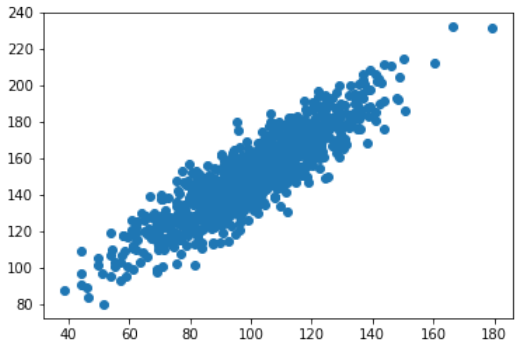
1. Import libraries for data manipulation and visualization
2. Set the seed to 1 (Parameter to ensure the reproducibility , if we rerun the code many times the same data will be generated)
3. Generate two sets of random data (data1 and data2) using numpy randn function. data1 is generated with a mean of 100 and standard deviation of 20, while data2 is generated by adding a random normal noise with mean 50 and standard deviation 10 to data1.
4. Calculate the covariance matrix between data1 and data2.
5. Calculate the pearson's correlation coefficient between data1 and data2.
6. Plotting
7. Print the statistics values

In [6]:
```
# calculate the Pearson's correlation between two variables
from numpy import mean
from numpy import std
from numpy import cov
from numpy.random import randn
from numpy.random import seed
from matplotlib import pyplot as plt
import seaborn as sns

from scipy.stats import pearsonr
# seed random number generator
seed(1)

# prepare data
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)

# calculate covariance matrix
covariance = cov(data1, data2)

# calculate Pearson's correlation
corr, _ = pearsonr(data1, data2)

# plot
plt.scatter(data1, data2)
plt.show()

# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
print('Covariance: %.3f' % covariance[0][1])
print('Pearsons correlation: %.3f' % corr)
```




```
data1: mean=100.776 stdv=19.620
data2: mean=151.050 stdv=22.358
Covariance: 389.755
Pearsons correlation: 0.888
```

In a numpy randn function, when you generate random numbers using for example randn(1000), it automatically creates an array of 1000 random numbers sampled from a standard normal distribution. standard normal distribution = (standard deviation = 1) and (mean = 0)

Some conclusions :

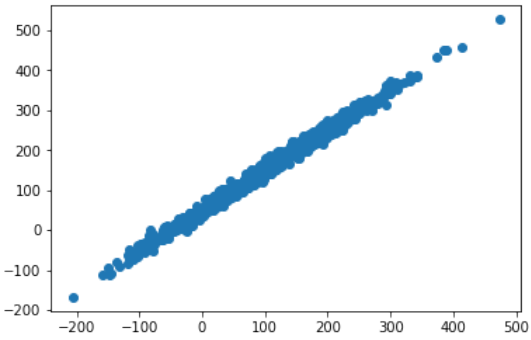
- Covaraince is positive ==> There is a link between data 1 and data 2.
- Pearson correlation = 0.888 ==> there is a strongly positive strenght for the linear relationship .

To observe how changes in data points impact correlation, we can modify the variables data1 and data2. Here are some changes that we can make:

- Increase/decrease the spread of data: Change the standard deviation of data1 and data2 by multiplying the randn function by a different value.

Increase the spread (we will multiply by 100 instead of 10)

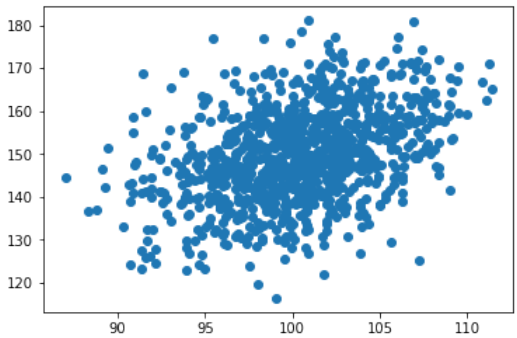
In [7]:
```
# prepare data
data1 = 100 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)

# calculate covariance matrix
covariance = cov(data1, data2)

# calculate Pearson's correlation
corr, _ = pearsonr(data1, data2)

# plot
plt.scatter(data1, data2)
plt.show()

# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
print('Covariance: %.3f' % covariance[0][1])
print('Pearsons correlation: %.3f' % corr)
```




```
data1: mean=97.783 stdv=97.162
data2: mean=147.819 stdv=98.001
Covariance: 9480.929
Pearsons correlation: 0.995
```

Decrease the spread (we will multiply by 4 instead of 10)

In [3]:
```
# prepare data
data1 = 4 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)

# calculate covariance matrix
covariance = cov(data1, data2)

# calculate Pearson's correlation
corr, _ = pearsonr(data1, data2)

# plot
plt.scatter(data1, data2)
plt.show()

# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
print('Covariance: %.3f' % covariance[0][1])
print('Pearsons correlation: %.3f' % corr)
```

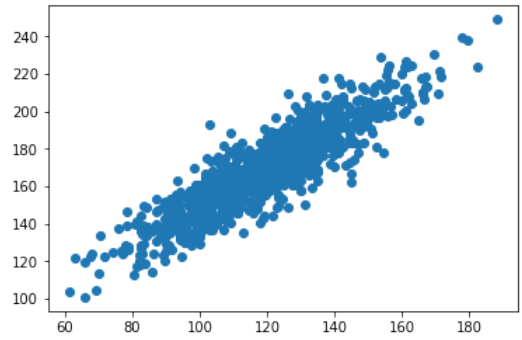



```
data1: mean=100.296 stdv=4.070
data2: mean=149.791 stdv=10.746
Covariance: 16.895
Pearsons correlation: 0.386
```

- We noticed that when we increase the spread , the pearson correlation also get higher (from 0.888 to 0.995). At the same time, when we decrease the spread by multiplying rand function by 4, the correlation also decrease (0.358).
- We can also notice that when decreasing the std deviation, the values are more centred around the average , and while decreasing this value , it is more spreaded and far from the mean value (100)

- The second strategy is to increase the mean to see how this will affect the results

In [4]:
```
# prepare data
data1 = 20 * randn(1000) + 120
data2 = data1 + (10 * randn(1000) + 50)

# calculate covariance matrix
covariance = cov(data1, data2)

# calculate Pearson's correlation
corr, _ = pearsonr(data1, data2)

# plot
plt.scatter(data1, data2)
plt.show()

# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
print('Covariance: %.3f' % covariance[0][1])
print('Pearsons correlation: %.3f' % corr)
```

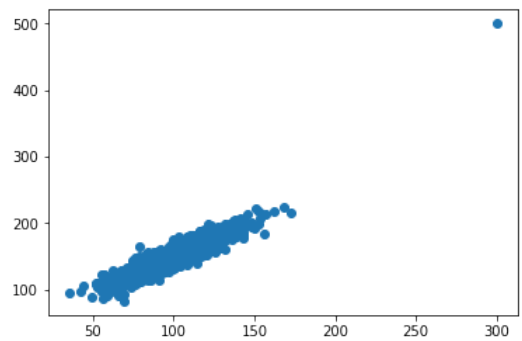



```
data1: mean=119.676 stdv=19.690
data2: mean=169.609 stdv=22.314
Covariance: 393.162
Pearsons correlation: 0.894
```

We noticed that the covariance increased a little bit, but in terms of correlation we approximately don't have changes.

- We will try also to introduce some outliers and observe

In [5]:
```
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)
data1[0] = 300  # Introducing an outlier in data1
data2[0] = 500  # Introducing a corresponding outlier in data2
# calculate covariance matrix
covariance = cov(data1, data2)

# calculate Pearson's correlation
corr, _ = pearsonr(data1, data2)

# plot
plt.scatter(data1, data2)
plt.show()

# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
print('Covariance: %.3f' % covariance[0][1])
print('Pearsons correlation: %.3f' % corr)
```




```
data1: mean=100.715 stdv=21.737
data2: mean=151.120 stdv=25.796
Covariance: 513.081
Pearsons correlation: 0.914
```

We noticed in the scatter plot a cloud of points with most of the data clustered around the range [50,150], but with one point far away from the main cluster due to the defined outliers.

In [ ]: