```
fit(training, test)
```

where **training** is a multi-variate training set, and test is a multi-variate test set. In **sklearn**, the data sets are expected to have multiple columns, to represent the different components of the independent variable. These components are called features; we will discuss this in the next chapter. When y is a function of a single variable x then we have to do some massaging of the original data to get it into a format that **sklearn** is happy with. We do that with the **numpy** array **reshape** command with arguments (-1,1). Assume that we already have **xtrain** and **ytrain** as defined previously. Then our **sklearn**-compatible training sets **XT** and **YT** are derived as:

```
XT=np.array(xtrain).reshape(-1,1)
YT=np.array(ytrain).reshape(-1,1)
```

The linear regression is calculated using the **fit** method. The Pearson r value, the intercept (a) and slope (y) are found using the **score**, **intercept_** and **coef_** methods. The latter two methods are returned as arrays because in general **fit** works with a multidimensional data set.

```
r=LinearRegression().fit(XT,YT)
print(r.intercept_[0], r.coef_[0,0])
```

```
-0.21681307672599814 0.9246615087756955)
```

The resulting linear fit is the same as that obtained with **linregress**, except for numerical errors in the higher precision digits.

Most classes in **sklearn** also have a **predict** method. After doing a fit, we can then make a prediction, using a training set, based on the fit. For a one-dimensional data set, we need to massage the test set as well before we use the **predict** method:

```
TX=np.array(xtest).reshape(-1,1)
TY=np.array(ytest).reshape(-1,1)
P=r.predict(TX)
```

The array **P** will contain values of y as determined by the fit $y = a + bx$ at each of the x values in **xtest**. We can calculate the MSS (eq. 1.66) using either the built-in function

```
mean_squared_error(TY,P)
```

```
1233.7745814823718
```