In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from prettytable import PrettyTable
```

In [2]:

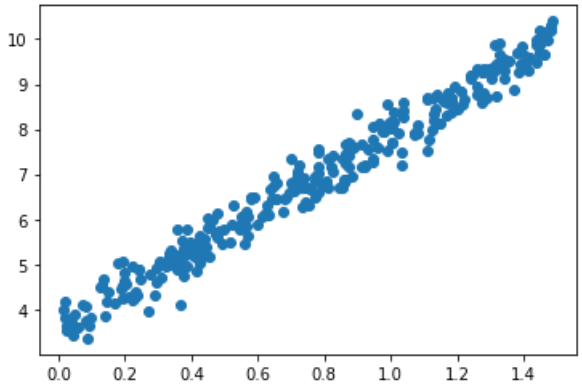
```
data_train = pd.read_csv('Dataset_2_train.csv')  # load data set
X_train = data_train.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y_train = data_train.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of r
ows, but have 1 column
```

In [3]:

```
plt.scatter(X_train, Y_train)
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0xbeb16d0>
```



In [4]:

```
data_train.shape
data_train.dtypes
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   1.1343615213    299 non-null    float64
 1   8.75521779949   299 non-null    float64
 2   Unnamed: 2      0 non-null      float64
dtypes: float64(3)
memory usage: 7.1 KB
```

In [5]:

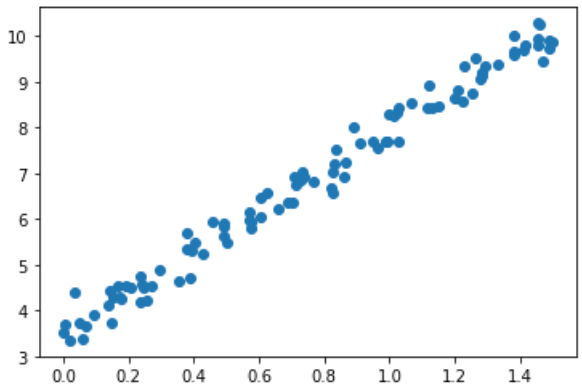
```
data_valid = pd.read_csv('Dataset_2_valid.csv')  # load data set
X_valid = data_valid.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y_valid = data_valid.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of r
ows, but have 1 column
```

In [6]:

```
plt.scatter(X_valid, Y_valid)
```

Out[6]:

```
<matplotlib.collections.PathCollection at 0xcf149a0>
```



In [7]:

```
data_valid.shape
data_valid.dtypes
data_valid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   0.285946632312  99 non-null     float64
 1   5.06873885044   99 non-null     float64
 2   Unnamed: 2      0 non-null      float64
dtypes: float64(3)
memory usage: 2.4 KB
```

Cost function:

In [8]:

```
def lr(x, y, m_new, c_new, learning_rate, epoch):
    N = float(len(y))
    for i in range(epoch):
        y_pred = (m_new * x) + c_new
        cost = sum([t ** 2 for t in (y - y_pred)]) / N     # MSE
        c_grad = - (2 / N) * sum (y - y_pred)              # Intercept
        m_grad = - (2 / N) * sum (x * (y - y_pred))        # Slope
        m_new = m_new - (learning_rate * m_grad)
        c_new = c_new - (learning_rate * c_grad)
        line = m_new * x + c_new
        plt.scatter(x, y, c = 'r')
        plt.plot(x, line)
#         plt.pause(3)
    print("MSE : ", cost)
    return c_new, m_new, cost,
```

Calling cost function for train data:

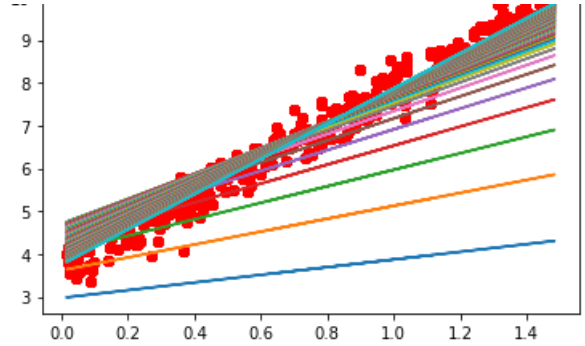
In [9]:

```
lr(x = X_train, y = Y_train, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 100)
```

```
MSE :  [0.10353464]
```

Out[9]:

```
(array([3.75213038]), array([4.11343891]), array([0.10353464]))
```



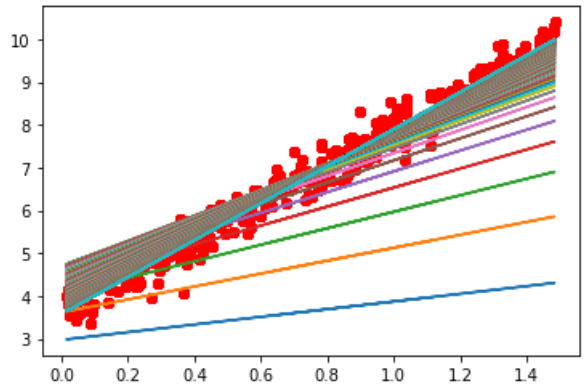
In [10]:

```
lr(x = X_train, y = Y_train, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 300)
```

```
MSE :  [0.09556913]
```

Out[10]:

```
(array([3.57959671]), array([4.31537957]), array([0.09556913]))
```



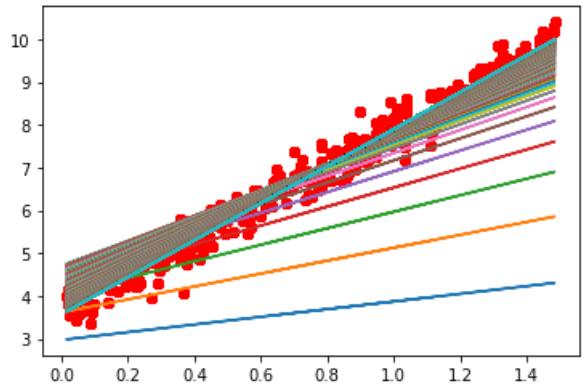
In [11]:

```
lr(x = X_train, y = Y_train, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 600)
```

```
MSE :  [0.09556752]
```

Out[11]:

```
(array([3.57710859]), array([4.31829178]), array([0.09556752]))
```



Calling cost function for train data with different learning rates and epoches:

In [12]:

```
# lr(x = X_valid, y = Y_valid, m_new = 0.001, c_new = 2, learning_rate = 1e-6, epoch = 1000000)
```

Observation:

- As we can see from the plot above, the smaller learning rates require higher training epoch, because of the small changes made to the weights of each epoch.

- Therefore in order to set the learning rate to (1e-6) the training epoch must be >= 10^6.

- Because of the runtime the call for (1e-6) learning rate is deactivated.

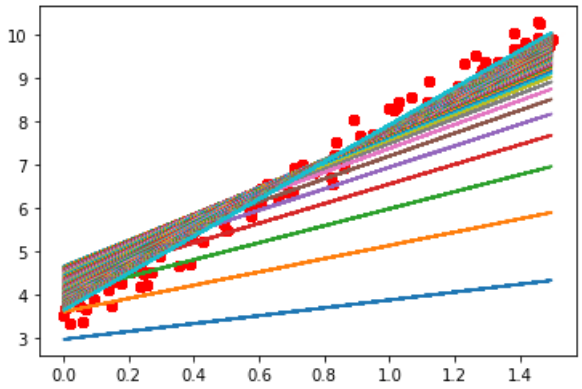
In [13]:

```
lr(x = X_valid, y = Y_valid, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 100)
```

```
MSE :  [0.07659054]
```

Out[13]:

```
(array([3.63102907]), array([4.27694458]), array([0.07659054]))
```



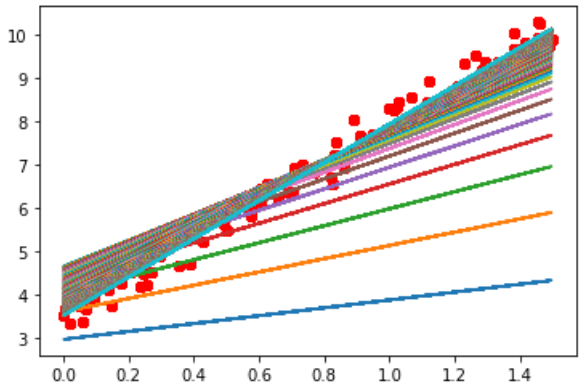
In [14]:

```
lr(x = X_valid, y = Y_valid, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 300)
```

```
MSE :  [0.07233881]
```

Out[14]:

```
(array([3.51583249]), array([4.41114908]), array([0.07233881]))
```



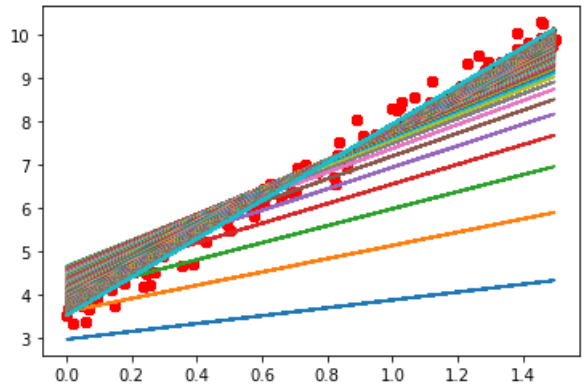
In [15]:

```
lr(x = X_valid, y = Y_valid, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 600)
```

```
MSE :  [0.07233867]
```

Out[15]:

```
(array([3.5151747]), array([4.41191542]), array([0.07233867]))
```



Calling cost function for test data with sutable variables:

In [16]:

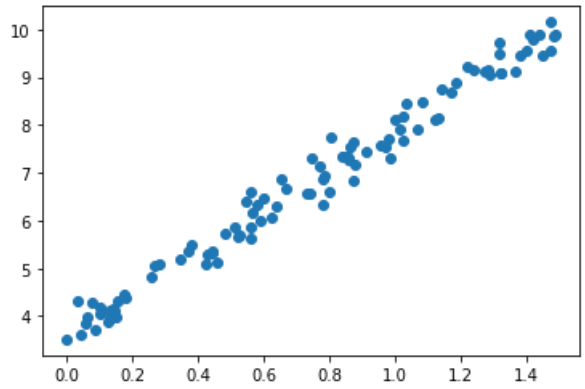
```
data_test = pd.read_csv('Dataset_2_test.csv')  # load data set
X_test = data_test.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y_test = data_test.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of row
s, but have 1 column
```

In [17]:

```
plt.scatter(X_test, Y_test)
```

Out[17]:

```
<matplotlib.collections.PathCollection at 0xd5c6928>
```



In [18]:

```
data_test.shape
data_test.dtypes
data_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   0.239309719927  99 non-null     float64
 1   4.48901037639   99 non-null     float64
 2   Unnamed: 2      0 non-null      float64
dtypes: float64(3)
memory usage: 2.4 KB
```

In [19]:

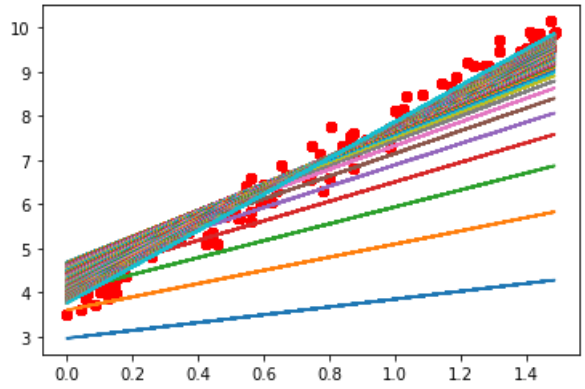
In [19]:

```
lr(x = X_test, y = Y_test, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 100)
```

```
MSE :  [0.07418261]
```

Out[19]:

```
(array([3.75530167]), array([4.11498689]), array([0.07418261]))
```



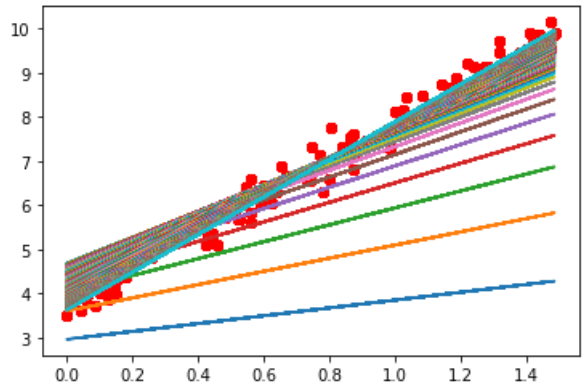
In [20]:

```
lr(x = X_test, y = Y_test, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 300)
```

```
MSE :  [0.06916835]
```

Out[20]:

```
(array([3.62617645]), array([4.2669677]), array([0.06916835]))
```



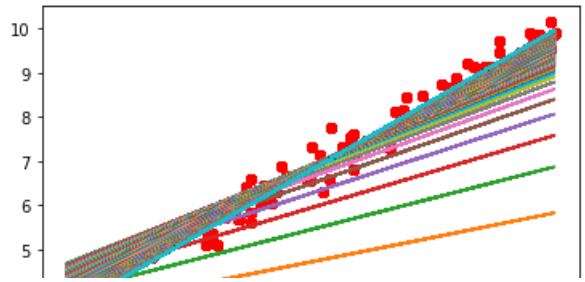
In [21]:

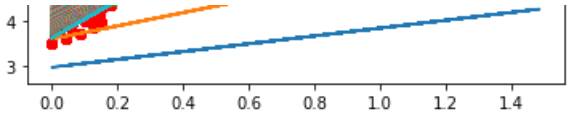
```
lr(x = X_test, y = Y_test, m_new = 0.001, c_new = 2, learning_rate = 1e-1, epoch = 600)
```

```
MSE :  [0.069168]
```

Out[21]:

```
(array([3.62508897]), array([4.26824767]), array([0.069168]))
```





Summery of the most sutable values:

Observation:

- There were improvements in outputs after increasing the epoch values from (100) to (300).

But:

- There were no huge different in outputs after increasing the epoch values from (300) to (600).

In [22]:

```
x = PrettyTable()

x.field_names = ["Variable", "Train Dataset", "Valid Dataset", "Test Dataset"]

x.add_row(["Intercept", 3.57959755, 3.51583271, 3.62617683])
x.add_row(["Slope", 4.31537859, 4.41114884, 4.26696726])
x.add_row(["Learning rate", 1e-1, 1e-1, 1e-1])
x.add_row(["Epoch", 300, 300, 300])
x.add_row(["Mean squared erroe", 0.09556913, 0.07233881, 0.06916835])

print(x)
```

```
+--------------------+---------------+---------------+--------------+
|      Variable      | Train Dataset | Valid Dataset | Test Dataset |
+--------------------+---------------+---------------+--------------+
|     Intercept      |   3.57959755  |   3.51583271  |  3.62617683  |
|       Slope        |   4.31537859  |   4.41114884  |  4.26696726  |
|   Learning rate    |      0.1      |      0.1      |     0.1      |
|       Epoch        |      300      |      300      |     300      |
| Mean squared erroe |   0.09556913  |   0.07233881  |  0.06916835  |
+--------------------+---------------+---------------+--------------+
```