In [1]:

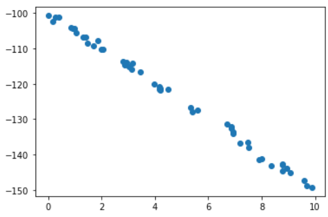
```python
from sklearn import linear_model
from sklearn import datasets

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

First example

In [2]:

```python
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = - 5 * x - 100 + rng.randn(50)
plt.scatter(x, y)
plt.show()
```

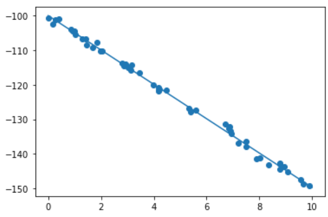


In [3]:

```python
model = linear_model.LinearRegression(fit_intercept=True)
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])
plt.scatter(x, y)
plt.plot(xfit, yfit)
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x1ff5496be08>]
```



In [4]:

```python
print('Accuracy: :', model.score(x[:, np.newaxis], y))
```

```
Accuracy: : 0.996461661920057
```

We created a linear samples that are noised not so much that can be decrease the accuracy, so we have very good model to predict linear feature Мы создали линейную выборку которая слабо зашумлена чтобы повлиять на снижение точности, потому мы получили можель с высокой точностью (выше 99)

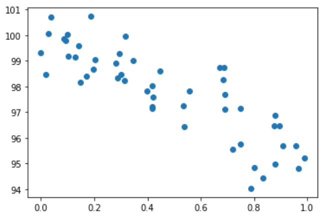
Second example

In [5]:

```python
rng = np.random.RandomState(1)
x = 1 * rng.rand(50)
y = - 5 * x + 100 + rng.randn(50)
plt.scatter(x, y)
```

Out[5]:

```
<matplotlib.collections.PathCollection at 0x1ff54a215c8>
```

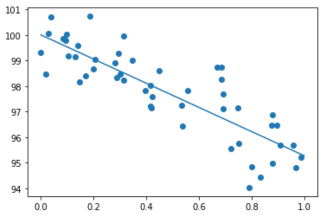


In [6]:

```python
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 1, 1000)
yfit = model.predict(xfit[:, np.newaxis])
plt.scatter(x, y)
plt.plot(xfit, yfit)
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x1ff548c9e48>]
```



In [7]:

```python
print('Accuracy: :', model.score(x[:, np.newaxis], y))
```

```
Accuracy: : 0.7179646757110463
```

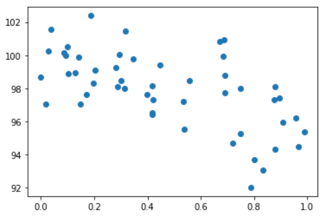
Now we have pretty noised sample then model lose the accuracy to 71% Теперь у нас достаточно зашумленные данные и модель потеряла в точности до 71%

In [8]:

```python
rng = np.random.RandomState(1)
x = 1 * rng.rand(50)
y = - 5 * x + 100 + 2 * rng.randn(50)
plt.scatter(x, y)
```

Out[8]:

```
<matplotlib.collections.PathCollection at 0x1ff54a971c8>
```

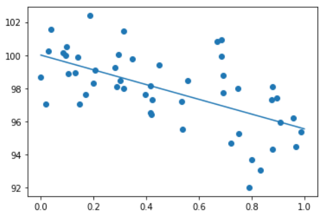


In [9]:

```python
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 1, 1000)
yfit = model.predict(xfit[:, np.newaxis])
plt.scatter(x, y)
plt.plot(xfit, yfit)
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x1ff54acd588>]
```



In [10]:

```python
print('Accuracy: :', model.score(x[:, np.newaxis], y))
```

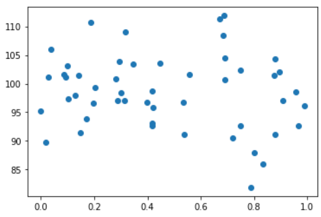
```
Accuracy: : 0.3611330670356685
```

In [11]:

```python
rng = np.random.RandomState(1)
x = 1 * rng.rand(50)
y = - 5 * x + 100 + 7 * rng.randn(50)
plt.scatter(x, y)
```

Out[11]:

```
<matplotlib.collections.PathCollection at 0x1ff54b67a48>
```

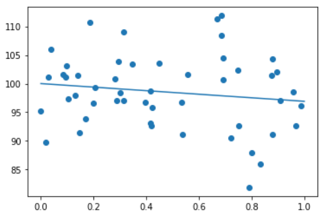


In [12]:

```python
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 1, 1000)
yfit = model.predict(xfit[:, np.newaxis])
plt.scatter(x, y)
plt.plot(xfit, yfit)
```

Out[12]:

```
[<matplotlib.lines.Line2D at 0x1ff54ba0e08>]
```



In [13]:

```python
print('Accuracy: :', model.score(x[:, np.newaxis], y))
```

```
Accuracy: : 0.021783545551722794
```

When noise raises we cant estimate this data Мы не сможем нормально оценивать данные при росте зашумленности

Real Example

In [14]:

```python
data = datasets.load_boston()

df = pd.DataFrame(data.data, columns=data.feature_names)
df
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98

1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [15]:
```
target = pd.DataFrame(data.target, columns=["MEDV"])
target
```

Out[15]:

	MEDV
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
...	...
501	22.4
502	20.6
503	23.9
504	22.0
505	11.9

506 rows × 1 columns

In [16]:
```
X = df
y = target["MEDV"]
```

In [17]:
```
lm = linear_model.LinearRegression()
model = lm.fit(X, y)

predictions = lm.predict(X)
predictions[::10] # each 10th
```

Out[17]:
```
array([30.00384338, 18.99949651, 12.52385753, 11.45511759, 34.21510225,
       21.28152535, 17.8725804 , 25.20148859, 28.40576968, 27.1190194 ,
       24.58022019, 20.64965864, 21.91632071, 20.03527399, 13.57141932,
       20.83858153, 32.70827666, 22.5551466 , 34.72440464, 30.76109485,
       30.64393906, 22.39251096, 33.18419746, 24.48449227, 27.29375938,
       24.20633547, 34.81211508, 22.30295533, 38.79756966, 33.43563311,
       30.77159449, 18.54266897, 24.88682244, 21.59260894, 21.42939305,
       20.45944095, 22.66616105, 34.60684042, 14.36994825, 17.21225183,
       11.88145871, 15.22308298, 19.6000267 , 18.15955693, 12.70609699,
       16.52271631, 19.04860533, 20.16719441, 23.46878866,  3.66399672,
       20.46870847])
```

In [18]:
```
print('Accuracy: :', lm.score(X, y))
```

```
Accuracy: : 0.7406426641094094
```

Better estimator

we take real dataset and try to estimate? bun in real life data cannot be estimated as linear

In [23]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
degree = 2
polyreg = make_pipeline(PolynomialFeatures(degree), LinearRegression())
model = polyreg.fit(X, y)

predictions = polyreg.predict(X)
predictions[::10] # each 10th
```

Out[23]:
```
array([24.3257612 , 19.91724885, 14.36456096, 14.74991608, 35.22468674,
       21.20877671, 17.67108107, 24.10908532, 28.60627639, 25.66422725,
       24.73222625, 24.33368027, 21.37493861, 20.71394777, 12.97355878,
       22.70421612, 33.10782409, 19.76273012, 39.28320932, 30.71164918,
       34.63082731, 18.20830035, 27.41357017, 22.69528449, 26.15843749,
       24.95066726, 34.9408021 , 20.69364631, 43.53638494, 31.99840903,
       26.95734119, 13.93474865, 25.24075377, 19.4572084 , 21.14143312,
       21.68253481, 23.88177872, 51.00177801, 13.00667083, 16.78720033,
        5.42285466, 17.41147089, 17.11824799, 18.86050451,  6.49524653,
       14.39224672, 17.66275156, 19.07828701, 23.4737525 ,  7.28745854,
       18.82971895])
```

In [25]:
```
print('Accuracy: :', polyreg.score(X, y))
```

```
Accuracy: : 0.9239966562443899
```

In [26]:
```
degree = 3
polyreg = make_pipeline(PolynomialFeatures(degree), LinearRegression())
model = polyreg.fit(X, y)

predictions = polyreg.predict(X)
predictions[::10] # each 10th
print('Accuracy: :', polyreg.score(X, y))
```

```
Accuracy: : 0.9979693979159469
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

model to solve real-life problem