Create a simulated function $f(x)$ and try to recover it through ML.

$$f(x) = 1.05^x$$

which emulates the compound interest growth.

1. Heuristic programming — we know (or think we know) the function f. We program it determinstically in the code.
2. Use in production: pass x, infer y, i.e. $y = f(x)$.
3. Machine learning uses *data* to learn $f(x)$. We don't know $f(x)$ upfront. All we have is data. We use an ML algorithm to approximate $f(x)$. We call such $f(x)$ *a model*. Once, we build the model the use in production is the same as in #2.

The code below illustrates how we can learn an underlying function $f(x)$ by just looking at the data.

In [1]:
```
%matplotlib inline
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
def f(x):
    return np.power(1.10, x)
```

Use an Object to hold all relevant data to avoid creating many global variables. Global variables are visible from anywhere in the notebook, can be used by accident, and lead to confusing results.

In [3]:
```
class Object(object): pass

var = Object()

var.x = np.arange(0, 50)
var.y = f(var.x)

# We'll use values below to *test* our model.
var.x_test = np.arange(0.5, 50.5)
var.y_test = f(var.x_test)
```

In [4]:
```
from sklearn.ensemble import RandomForestRegressor
var.m = RandomForestRegressor(n_estimators=100, oob_score=True)
_ = var.m.fit(var.x.reshape(-1, 1), var.y)
var.m.oob_score_
```

Out[4]:
```
0.9923736498833117
```

In [5]:
```
1 var.m.score(var.x_test.reshape(-1, 1), var.y_test)
```

Out[5]:
```
0.996623154019475
```

This is a very high R^2 score, meaning our random forest model approximates this function extremely well.

In [6]:
```
1 print(f(1.25), var.m.predict([[1.25]]))
2 print(f(7.77), var.m.predict([[7.77]]))
```

```
1.1265250579928898 [1.1379522]
2.0971097706731032 [2.09585428]
```

In [7]:
```
1 f(1.24)
```

Out[7]:
```
1.1254518764414492
```

In [8]:
```
1 var.m.predict([[1.24]])
```

Out[8]:
```
array([1.1379522])
```

In [9]:
```
1 print(f(42.45), var.m.predict([[42.45]]))
```

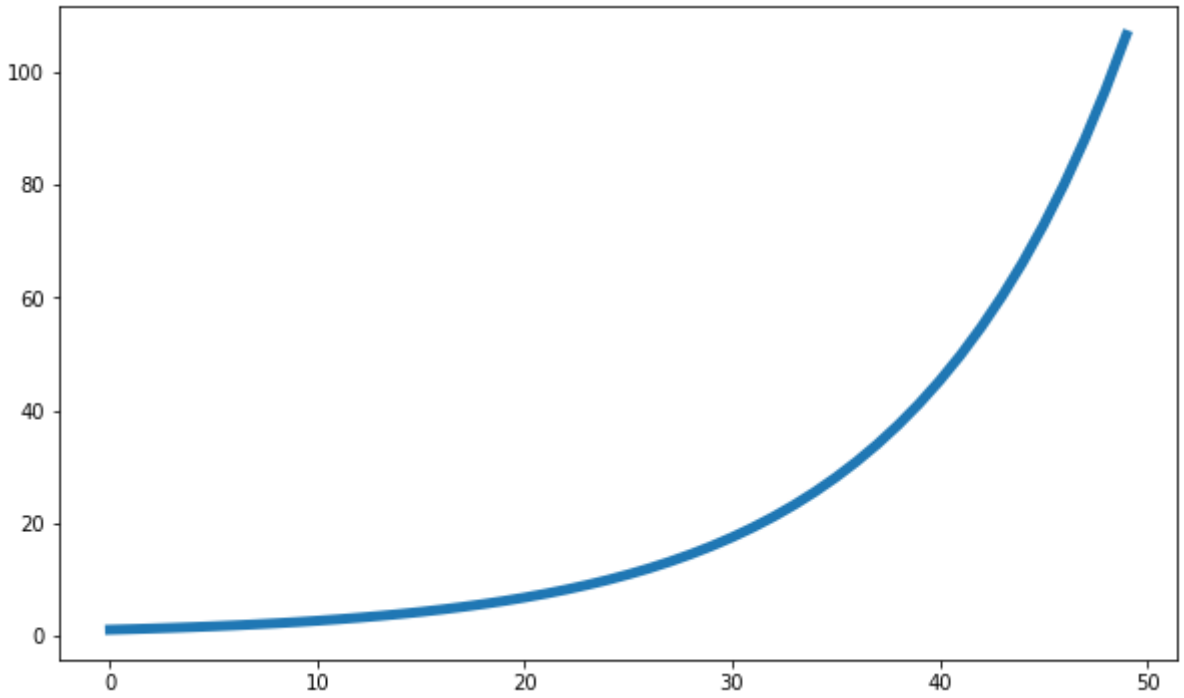
```
57.16358859640517 [55.16202595]
```

In [10]:
```
1 print(f(49.84), var.m.predict([[49.84]]))
```

```
115.61426652245606 [102.30747931]
```

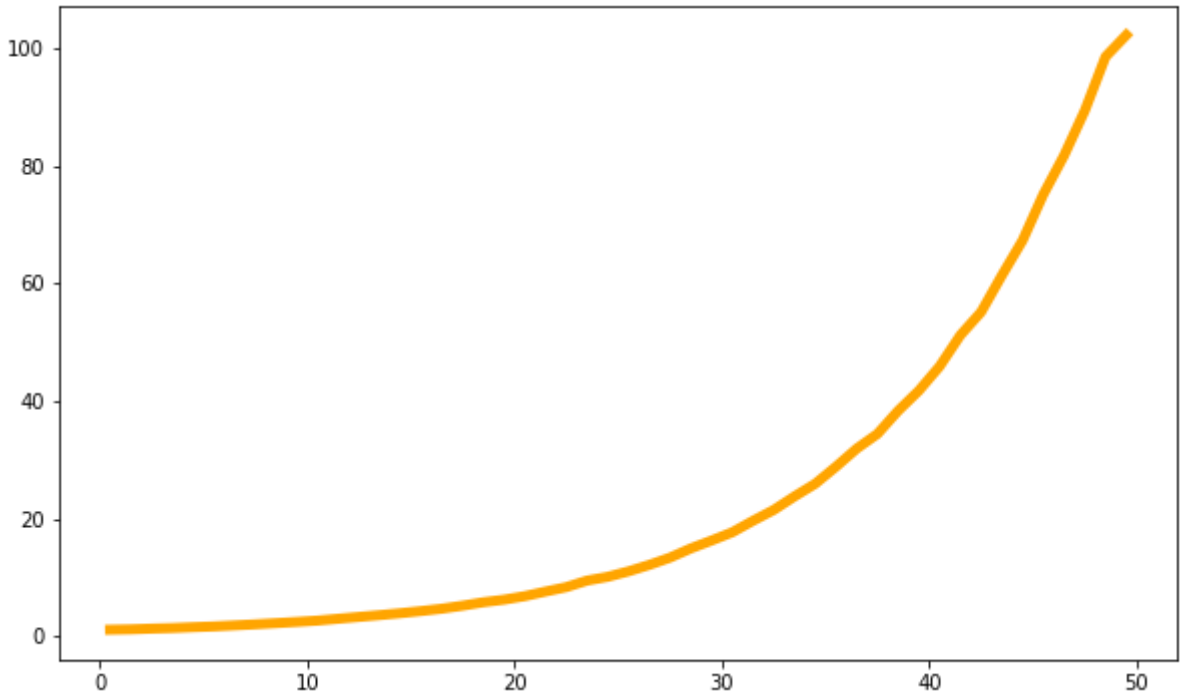
In [11]:
```
1 fig = plt.figure(figsize=(10, 6))
2 plt.plot(var.x, var.y, linewidth=5)
```

Out[11]: [<matplotlib.lines.Line2D at 0x2732eebd68>]



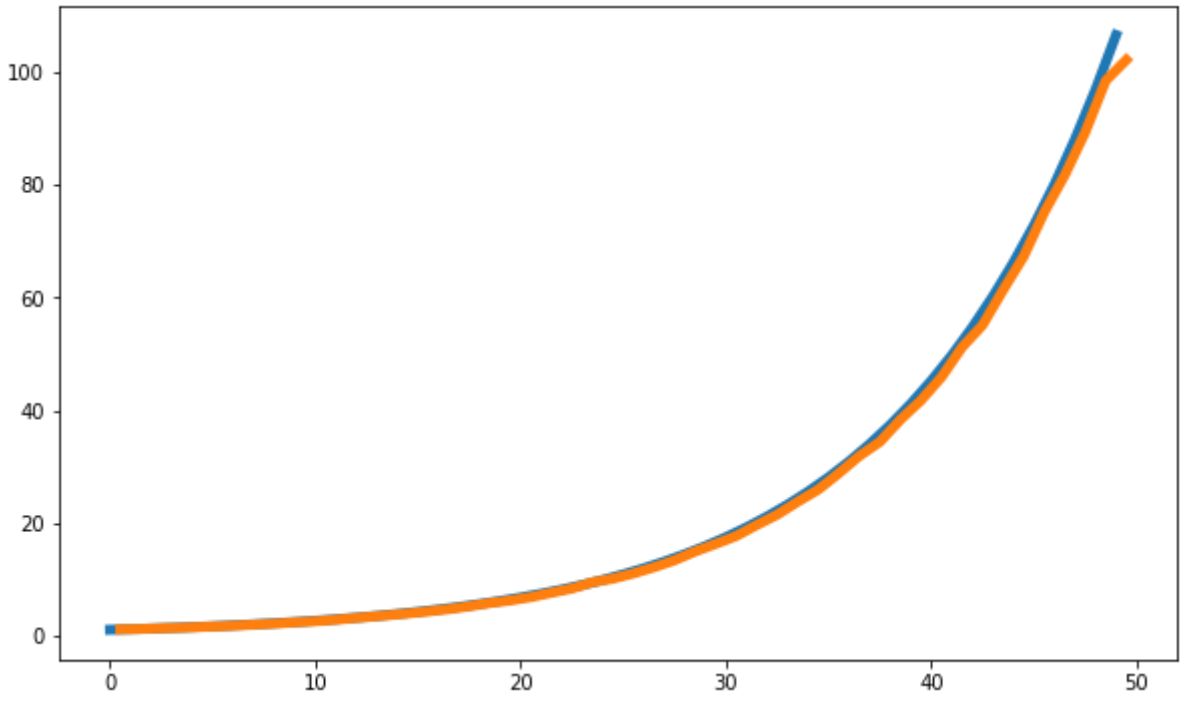
In [15]:
```
1 fig = plt.figure(figsize=(10, 6))
2 plt.plot(var.x_test, var.m.predict(var.x_test.reshape(-1, 1)), lin
```

Out[15]: [<matplotlib.lines.Line2D at 0x27339223c8>]



In [14]:
```
1 fig = plt.figure(figsize=(10, 6))
2 plt.plot(var.x, var.y, linewidth=5)
3 plt.plot(var.x_test, var.m.predict(var.x_test.reshape(-1, 1)), lin
```

Out[14]: [<matplotlib.lines.Line2D at 0x2733722438>]



Different $f(x)$, same algorithm (Random Forest)

$f(x) = 1.7x - 3.2$

In [16]:
```
1 _2 = Object()
2
3 def f_2(x):
4     """
5         f(x) = 1.7 * x - 3.2
6     """
7     return np.multiply(x, 1.7) - 3.2
```

In [17]:
```
1 _2.x = np.arange(0, 50)
2 _2.x_test = np.arange(0.5, 50.5)
3 _2.y = f_2(_2.x)
```

In [18]:
```
1 _2.df = pd.DataFrame(data={'x' : _2.x, 'y' : _2.y})
```

The entirety of our machine learning code is this:

In [19]:
```
1 _2.m = RandomForestRegressor(n_estimators=100, oob_score=True)
2 _ = _2.m.fit(_2.x.reshape(-1, 1), _2.y)
```

In [20]:
```
1 f_2(11.77)
```

Out[20]: 16.809

In [21]:
```
1 _2.m.predict([[11.77]])
```

Out[21]: array([16.605])

In [22]:
```
1 fig = plt.figure(figsize=(10, 6))
2 plt.plot(_2.x, _2.y, linewidth=5)
3 plt.plot(_2.x_test, _2.m.predict(_2.x_test.reshape(-1, 1)), linewi
```

Out[22]: [<matplotlib.lines.Line2D at 0x273398ea20>]