Define model

You can define your model by declaring a new function. The function should take the data as input and return the predicted values.

Python

```python
def model(Y, X):
    # Define the model here - the parameter name is taken from the variable name
    a = m.dist.normal(0, 1)
    b = m.dist.normal(1, 1)
    s = m.dist.exponential(1)

    # Likelihood - pass the observed data through obs
    m.dist.normal(a + b * X, s, obs = Y)
```

R

```r
model <- function(Y, X){
  # Define the model here - the parameter name is taken from the variable name
    a = m$dist$normal(0, 1)
    b = m$dist$normal(1, 1)
    s = m$dist$exponential(1)
    m$dist$normal(a + b * X, s, obs = Y)
}
```

Note the additional `obs` argument when declaring the likelihood function. This argument is used to specify the observed data.

The `name` argument is optional

Every prior needs a unique name so it can be tracked in the posterior. You **no longer have to pass it explicitly**: when `name` is omitted, BayesForge infers it from the variable the distribution is assigned to. In the model above, `a = m.dist.normal(0, 1)` registers a parameter named `"a"`, `b = ...` registers `"b"`, and so on.

You can still pass `name=` to override the inferred name — this is useful when the posterior name must differ from the local variable (e.g. to match an external reference such as a Stan parameter):

```
def model(Y, X):
    alpha = m.dist.normal(0, 1, name = 'a')   # variable 'alpha', but posterior name is 'a'
    beta  = m.dist.normal(1, 1, name = 'b')
    s     = m.dist.exponential(1)             # inferred name: 's'
    m.dist.normal(alpha + beta * X, s, obs = Y)
```

> **Note**
>
> Name inference only works for a **direct assignment** (`param = m.dist.<...>(...)`). If the result is used without being bound to a single variable — for example built inside a list/comprehension, passed straight into another call, or unpacked from a tuple — the name cannot be inferred and you must pass `name=` explicitly.