

Build, update, and query causal models with CausalQueries: Cheat Sheet

1. Make your model

CausalQueries works with models that assume a causal structure over **binary nodes**.

a. Make your model

Provide a causal structure connecting nodes using **dagitty** syntax.

```
model_1 <- make_model("X -> Y; Z -> Y")      # simple
model_2 <- make_model("Z -> X -> Y; X <-> Y") # with confounding
```

b. Optionally: refine your model

Set restrictions (to rule out some nodal types)

```
# naming types
make_model("X -> Y") |>
  set_restrictions(labels = list(X = c("0", "1"), Y = "00"))

# using logical statements
make_model("X -> Y") |>
  set_restrictions(statement = "Y[X=1] > Y[X=0]")

# natural language helpers
make_model("X -> Y") |> set_restrictions(decreasing("X", "Y"))
```

Set priors (to make use of prior knowledge)

```
make_model("X -> Y") |> set_priors(distribution = "jeffreys")

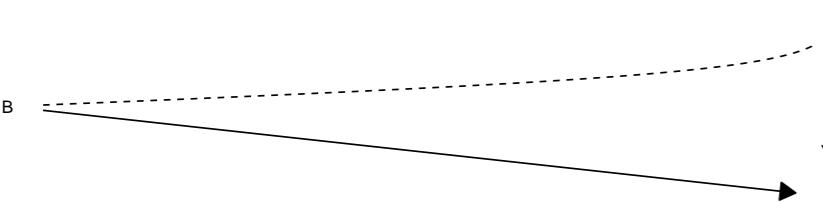
make_model("X -> Y") |>
  set_priors(statement = "Y[X=1] > Y[X=0]", alphas = .25)
```

Set parameters (to imagine a particular world)

```
make_model("X -> Y") |>
  set_parameters(statement = "Y[X=1] > Y[X=0]", parameters = .5)
```

c. Graph your model

```
make_model("A -> Y <- B; A <-> B") |> plot()
```



d. Inspect your model

```
make_model("X -> Y") |> inspect("nodal_types")
make_model("X -> Y") |> inspect("causal_types")
```

2. Update your model

To update, pass the model and data to **update_model()**.

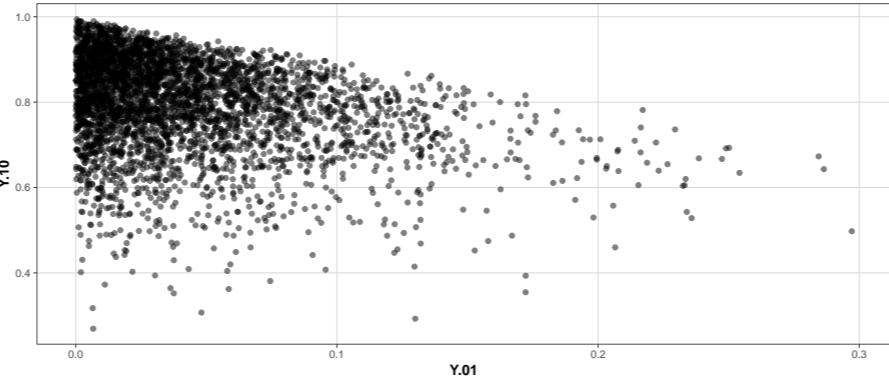
i Data can be complete or missing at random.

```
model <- make_model("X -> M -> Y")
data  <- data.frame(X = 0:1, M = 1:0, Y = 0:1) |> uncount(10)

updated <- update_model(model, data)
```

Posterior distributions on parameters can be accessed directly or when you query the model.

```
grab(updated, "posterior_distribution") |>
  ggplot(aes(Y.01, Y.10)) + geom_point(alpha = .5)
```



💡 Use additional Stan arguments to control updating

Other Stan arguments can be passed to **update_model()**:

- `iter` sets the number of iterations (and ultimately posterior draws)
- `chains` sets the number of chains (parallel chains can speed things up)
- many other options via `?rstan::stan`

💡 Causal syntax

CausalQueries uses a causal syntax that lets you describe arbitrary observational or controlled values of a node.

Square brackets indicate controlled values (implied application of the “do” operator).

An observational query: types that produce $Y == 1$.

```
make_model("X -> M -> Y <- X") |>
  get_query_types("Y==1")
```

A simple counterfactual query: types that produce $Y == 1$ when X is set to 1.

```
make_model("X -> Y") |>
  get_query_types("Y[X=1]==1")
```

A complex query: types for which $Y == 1$ when $X=1$ and M is held constant at the value it would take if X were 0.

```
make_model("X -> M -> Y") |>
  get_query_types("Y[M=M[X=0], X=1]==1")
```

We provide helpers for common causal statements (increasing, decreasing, interacts, etc.), which are useful for querying models or setting restrictions.

3. Query your model

Formulate a causal query using causal syntax and put the query to the model.

i Provide:

- causal queries (queries)
- conditions (given)
- which parameters to use (using)

```
model <- make_model("X -> M -> Y")
```

```
results <- model |>
  query_model(
    queries = list(
      ATE = "Y[X=1] - Y[X=0]",
      POS = "Y[X=1] > Y[X=0]"
    ),
    given = c(
      "All",
      "X==1 & Y==1",
      "X==1 & Y==1 & M==0"
    ),
    using = c("parameters", "priors")
  )
```

💡 Putting it all together

```
results <-
  make_model("X->M->Y") |>
  update_model(data) |>
  query_model(
    query = "Y[X=1]>Y[X=0]",
    given = c("All", "X==1 & Y==1", "X==1 & Y==1 & M==1"),
    using = c("priors", "posteriors")
  )
```

and view as table or plot:

```
plot(results) + theme(legend.position = "bottom")
```

