scratch pad:

functionality

primitives:

```
viz.bar(['a','b','c'], [1,2,3])
viz.hist(repeat(1e4, function() { return uniformDraw(['a','b','c']) }))
viz.scatter([1,2,3],[4,5,6])
viz.line([1,3,2],[4,5,6])
viz.density(repeat(1e2, function() { return gaussian(0,1) }))
```

line:

```
viz.line([1,2,3], [4,5,6], {xLabel: 'foo', yLabel: 'bar'})
```

hist:

```
viz.hist(repeat(1e4, function() { return categorical([1/2,1/3,1/6], ['a','b','c']) }));
viz.hist(MH(function() { return {a: randomInteger(30) } }, 60), {numBins: 9})
viz.hist(MH( function() { return {p: uniform(0, 1)} }, 1000), {numBins: 10})
viz.hist([1,1]) // make sure single value works
```

density:

```
viz.density(MH(function() { return uniform(0,1) }, 1000))
viz.density(MH(function() { return {foo: uniform(0,1)} }, 1000))
viz.density(repeat(1e2, function() { return gaussian(0,1) }))
viz.density(repeat(1e2, function() { return uniformDraw(['a','b']) })) // TODO: add error message
```

table:

```
var dist = ParticleFilter(function() {
  return {fruit: categorical([0.1, 0.2, 0.3, 0.4], ["apple", "banana", "orange", "grape"]),
          bread: uniformDraw(["sourdough", "wheat"])
         }
}, 500);

viz.table(dist, {destructure: false, log: false})
viz.table(dist, {destructure: false, log: true})
viz.table(dist, {destructure: true, log: false})
viz.table(dist, {destructure: true, log: true})
viz.table(dist, {destructure: true, log: false, top: 3})

// stringify cell contents if they're objects
viz.table(Enumerate(function() {
  return [
    {x: uniformDraw(['a','b']), y: uniformDraw(['c','d'])},
    {x: uniformDraw(['e','f']), y: uniformDraw(['g','h'])}
  ]
}))

// make sure table handles non-object support values
viz.table(Enumerate(function() { uniformDraw(['a','b','c']) }))
```

heat map:

```
viz.heatMap(repeat(2e2, function() { return [gaussian(0,1), beta(0.5,0.5)] }))
```

auto samples:

```
var samp = function() { categorical([0.1, 0.2, 0.3, 0.4], ["apple", "banana", "orange", "grape"])}

viz.auto(repeat(100, function() { return {fruit: samp()} }));
viz.auto(repeat(100, function() { return samp() }));
```

c:

```
viz.auto(ParticleFilter(function() {
  return {fruit: categorical([0.1, 0.2, 0.3, 0.4], ["apple", "banana", "orange", "grape"])}
}, 500))
```

(c):

```
viz.auto(ParticleFilter(function() {
  return categorical([0.1, 0.2, 0.3, 0.4], ["apple", "banana", "orange", "grape"])
}, 500))
```

r:

```
viz.auto(ParticleFilter(function() {
  return {x: beta(3,2)};
},500))
```

marginals:

```

viz.marginals(MH(function() {
  return {x1: uniform(0,1),
          x2: uniformDraw(['fruit', 'vegetable'])
         }
}, 100))

```

cc:

```
viz.auto(Enumerate(function() {
  return {
    fruit: categorical([0.1, 0.2, 0.3, 0.4], ["apple", "banana", "orange", "grape"]),
    boolean: flip(0.7)
  }}))

viz.auto(Enumerate(function() {
  return {
    x: binomial({n: 6, p: 0.5}) + "",
    y: binomial({n: 6, p: 0.3}) + ""
  }}))
```

cr:

```
viz.auto(MH(function() {
  var brand = flip(0.7) ? 'coke' : 'pepsi';
  return {
    brand: brand,
    price: gaussian(brand == 'coke' ? 4 : 2, 1)
  }
}, 2000));
```

rr:

```
viz.auto(MH(function() {
  var x = uniform(0,1);
  var y = uniform(0,1);
  factor(Math.log(Math.abs(x-y)))
  return {x: x, y: y};
},5000))
```

ccc:

```
viz.auto(Enumerate(function() {
  return {
    who: categorical([1, 1, 1, 1, 1, 1], ["Plum", "Peacock", "White", "Scarlet", "Mustard", "Green"]),
    where: categorical([1, 3, 5, 2, 4], ["candlestick", "knife", "revolver", "rope", "poison"]),
    with_what: categorical([1, 1, 1, 1, 1], ["dining", "kitchen", "hall", "conservatory", "library"])
  }
}))
```

ccr:

```
viz.auto(MH(function() {
  var brand = flip(0.7) ? 'coke' : 'pepsi';
  return {
    country: uniformDraw(['usa','mexico','canada']),
    brand: brand,
    price: gaussian(brand == 'coke' ? 4 : 2, 1)
  }
}, 2000));
```

crr:

```
viz.auto(MH(function() {
  var brand = flip(0.7) ? 'coke' : 'pepsi'
  var x = uniform(0,1);
  var y = uniform(0,1);
  factor(Math.log(Math.abs(x-y)))
  return {brand: brand, x: x, y: y};
},5000))
```

rrr:

```
viz.auto(ParticleFilter(function() {
  var x = uniform(0,1);
  var y = uniform(0,1);
  var z = uniform(0,1);
  factor(Math.log(Math.abs(x-y) + Math.abs(x-z)))
  return {x: x, y: y, z: z};
},500))
```

marginals:

```
var model = function(){
  var my_mu = sample(Gaussian({mu: 0, sigma: 31}))
  var my_sigma = sample(Uniform({a:0.0, b:10.0}))
  observe(Gaussian({mu: my_mu, sigma: my_sigma}), 1.1)
  observe(Gaussian({mu: my_mu, sigma: my_sigma}), 1.9)
  observe(Gaussian({mu: my_mu, sigma: my_sigma}), 2.3)
  observe(Gaussian({mu: my_mu, sigma: my_sigma}), 1.8)
  return {mu: my_mu, sigma: my_sigma}
}
var posterior = Infer({method: 'rejection', samples: 10, maxScore: -2.32}, model)
viz.marginals(posterior)
```

base distributions:

```

viz(Gaussian({mu: 0, sigma: 1}))
viz(Beta({a: 1, b: 1}))
viz(Exponential({a: 5}))
viz(Gamma({shape: 2, scale: 2}))
viz(Cauchy({location: -50, scale: 10}))
viz(MultivariateGaussian({mu: Vector([2,-3]),
                          cov: Matrix([[1,0.5],
                                       [0.5,1]])
                         }))

viz(Binomial({p: .5, n: 5}))
viz(Rejection(function() {
  return {x: uniformDraw(_.range(20))}
}, {samples: 5}))


viz(Dirichlet({alpha: Vector([2,1]) }))
viz(Dirichlet({alpha: Vector([1,1,1]) }), {samples: 200})
viz(Poisson({mu: 5}))
```

static analysis

dependency structure:

```
var f = function() {
  var x = flip(0.5); // [x],
  var y = repeat(2, function() { x ? gaussian(0,1) : beta(2,2) }); // [x,y], // TODO: second arg should be a function, not a value
  var z = map(function (x) { var z = x + 1; return z }, y); // [x,y,x],
  return z
};

viz.model(f)
```

depAuto: visualize independent cliques

```
var model = function() {
  var x = flip(0.5);
  var y = beta(4,4);
  var z = gaussian(0,1);
  var w = uniform(-z, z);

  return {x: x, y: y, z: z, w: w}}

viz.depAuto(Infer({method: 'rejection', samples: 1000}, model))
```

(thought -- if there are independent cliques, does this mean we can decompose inference into subproblems over independent submodels?)

user friendliness

tick label rotation:

```

var dist = Enumerate(function() {uniformDraw(['alphabet','city','soup']) })
viz.auto(dist)

```

smart axis number formatting:

```

viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.00000001}; },500))
///fold:
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.0000001}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.000001}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.00001}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.0001}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.001}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.01}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 0.1}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 1}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 10}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 100}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 1000}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 10000}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 100000}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 1000000}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 10000000}; },500))
viz.auto(ParticleFilter(function() { return {x: beta(3,2) * 100000000}; },500))
///
```

regression testing

silverman's rule can fail:

```
var model = function() { gaussian(flip() ? 30 : -30, 1); }
var dist = Rejection(model, {samples: 200});
viz.hist(dist)
viz.density(dist)
```

make sure x axis doesn't force lower bound of zero:

```

      var model = function() { return gaussian(1000,20) }
var dist = MH(model, 500)
viz.density(dist)
viz.auto(dist)
```