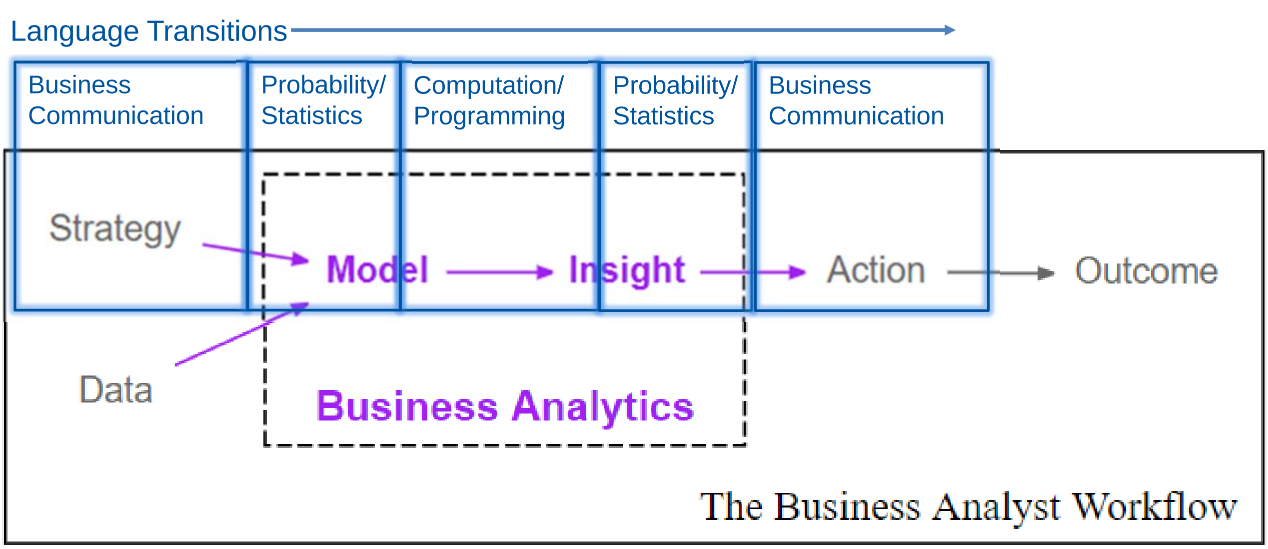
causact's Generative DAGs As Solution To Three Language Problem (business/math/code)

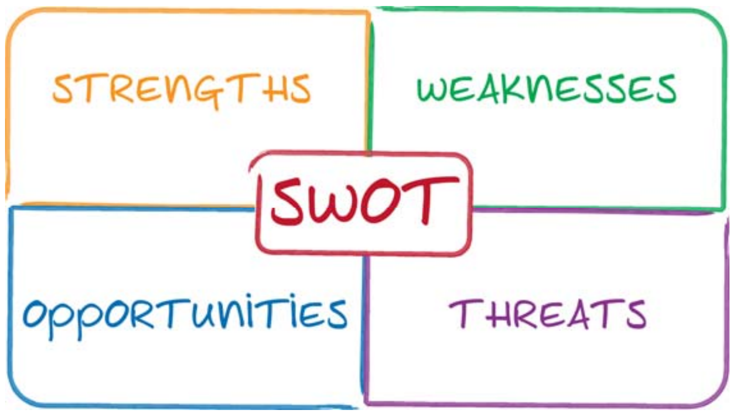
Speaking Data Analytics

UNIVERSITY OF DELAWARE



UNIVERSITY OF DELAWARE

1



Language of Business

Pictures

$$
\begin{aligned}
K_i &\sim \mathrm{Binomial}(n_i, p_i) \\
p_i &= \mathrm{ilogit}(\alpha_j + \beta_j x_i) \\
\alpha_j &\sim \mathrm{Normal}(\alpha, \sigma_\alpha) \\
\beta_j &\sim \mathrm{Normal}(\beta, \sigma_\beta) \\
\alpha &\sim \mathrm{Normal}(0, 10) \\
\sigma_\alpha &\sim \mathrm{Uniform}(0, 5) \\
\beta &\sim \mathrm{Normal}(0, 2) \\
\sigma_\beta &\sim \mathrm{Uniform}(0, 5)
\end{aligned}
$$

Language of Statistics

Formulas

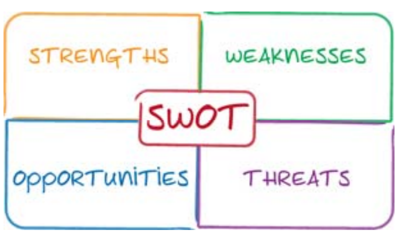
```python
class BnLayer(nn.Module):
    def __init__(self, ni, nf, stride=2):
        super().__init__()
        self.conv = nn.Conv2d(ni, nf, kernel_size=3, stride=stride, bias=False, padding=1)
        self.a = nn.Parameter(torch.zeros(nf,1,1))
        self.m = nn.Parameter(torch.ones(nf,1,1))

    def forward(self, x):
        x = F.relu(self.conv(x))
        x_chan = x.transpose(0,1).contiguous().view(x.size(1), -1)
        if self.training:
            self.means = x_chan.mean(1)[:,None,None]
            self.stds  = x_chan.std (1)[:,None,None]
        x = x - self.means
        x = x / self.stds
        return x*self.m+self.a
```

```python
class ResnetLayer(BnLayer):
    def forward(self, x): return x + super().forward(x)
```

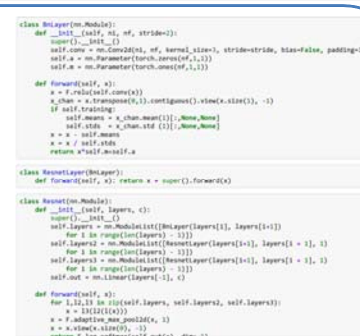
```python
class Resnet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.layers3 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
            x = l3(l2(l(x)))
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

Language of Computers

Code



$$K_i \sim \text{Binomial}(n_i, p_i)$$
$$p_i = \text{ilogit}(\alpha_j + \beta_j x_i)$$
$$\alpha_j \sim \text{Normal}(\alpha, \sigma_\alpha)$$
$$\beta_j \sim \text{Normal}(\beta, \sigma_\beta)$$
$$\alpha \sim \text{Normal}(0, 10)$$
$$\sigma_\alpha \sim \text{Uniform}(0, 5)$$
$$\beta \sim \text{Normal}(0, 2)$$
$$\sigma_\beta \sim \text{Uniform}(0, 5)$$



Output We Want For Making Decisions Under Uncertainty

A Joint Distribution

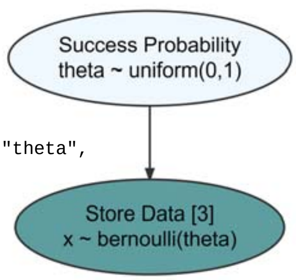
Joint Distribution Milestone

x	y	$P(x,y)$
0	Toyota Corolla	50%
0	Jeep Wrangler	5%
0	Subaru Outback	5%
0	Kia Forte	4%
1	Toyota Corolla	10%
1	Jeep Wrangler	15%
1	Subaru Outback	10%
1	Kia Forte	1%

Wanted:
The Rosetta Stone Of Data Science

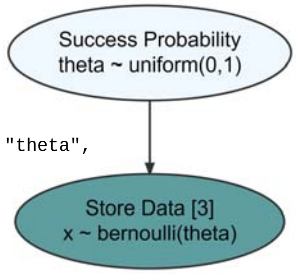
Generative DAGs are the Rosetta Stone

```
graph = dag_create() %>%
  dag_node(descr = "Store Data", label = "x",
           rhs = bernoulli(theta),
           data = c(1,1,0)) %>%
  dag_node(descr = "Success Probability", label = "theta",
           rhs = uniform(0,1)) %>%
  dag_edge(from = "theta",
           to = "x")
```

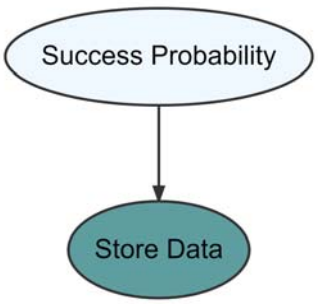


Generative DAGs are the Rosetta Stone

```
graph = dag_create() %>%
  dag_node(descr = "Store Data", label = "x",
           rhs = bernoulli(theta),
           data = c(1,1,0)) %>%
  dag_node(descr = "Success Probability", label = "theta",
           rhs = uniform(0,1)) %>%
  dag_edge(from = "theta",
           to = "x")
```




```
graph %>% dag_render(shortLabel = TRUE)
```




```
graph %>% dag_greta()
```

```
> graph %>% dag_greta(mcmc=FALSE)
## The below greta code will return a posterior distribution
## for the given DAG. Either copy and paste this code to use greta
## directly, evaluate the output object using 'eval', or
## or (preferably) use dag_greta(mcmc=TRUE) to return a data frame
 of
## the posterior distribution:
x <- as_data(c(1, 1, 0))   #DATA
theta  <- uniform(min = 0, max = 1)   #PRIOR
distribution(x) <- bernoulli(prob = theta)   #LIKELIHOOD
gretaModel  <- model(theta)   #MODEL
meaningfulLabels(graph)
draws        <- mcmc(gretaModel)                #POSTERIOR
drawsDF      <- replaceLabels(draws) %>% as.matrix() %>%
                dplyr::as_tibble()              #POSTERIOR
tidyDrawsDF <- drawsDF %>% addPriorGroups()   #POSTERIOR
```

UNIVERSITY OF DELAWARE