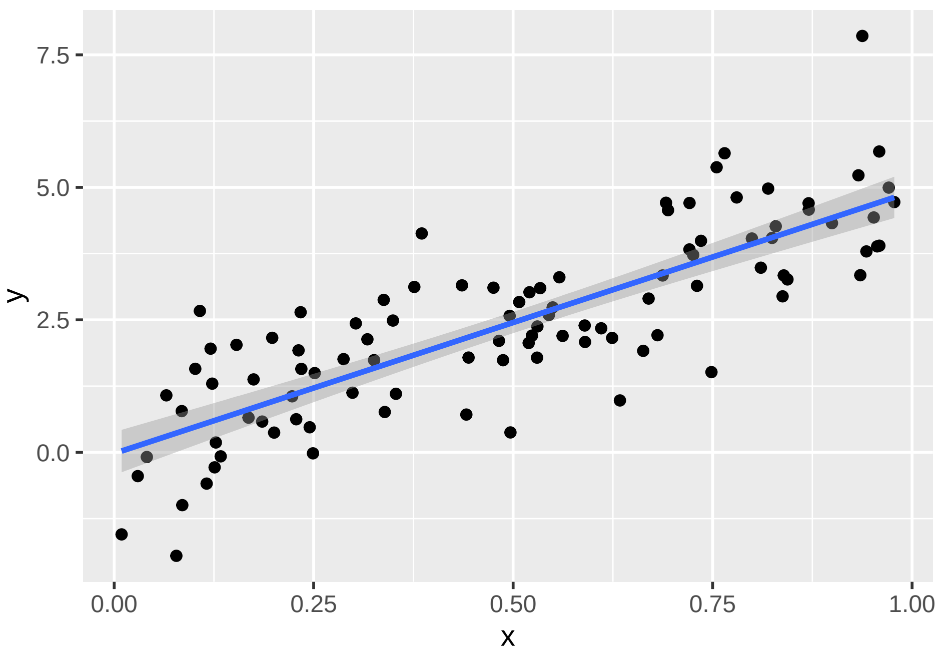
Stan demo

Part 1. Linear Regression First consider a simple linear regression model with a single continuous variable.

```
n <- 100
beta <- 5
sigma <- 1
x <- runif(n)
y <- rnorm(n, x * beta, sigma)

lm_dat <- tibble(x = x, y = y)
lm_dat %>% ggplot(aes(y = y, x = x)) + geom_point() + geom_smooth(formula = 'y~x', method = 'lm')
```



We have used both `lm` and `stan_glm` to fit these models.

```
lm_dat %>% lm(y ~ x, data = .) %>% display()
```

```
## lm(formula = y ~ x, data = .)
##             coef.est coef.se
## (Intercept) -0.02     0.20
## x            4.94     0.35
## ---
## n = 100, k = 2
## residual sd = 1.01, R-Squared = 0.67
```

```
lm_dat %>% stan_glm(y ~ x, data = ., refresh = 0) %>% print(digits = 2)
```

```
## stan_glm
##  family:       gaussian [identity]
```

```
##  formula:      y ~ x
##  observations: 100
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) -0.02   0.21
## x            4.94   0.36
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 1.02   0.07
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Unsurprisingly, `stan_lm` uses stan. The code can be extracted, but is not particularly easy to follow. However, we can fairly easily write code Stan code for this model, or obtain it see https://mc-stan.org/docs/2_29/stan-users-guide/linear-regression.html.

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  y ~ normal(alpha + beta * x, sigma);
}
```

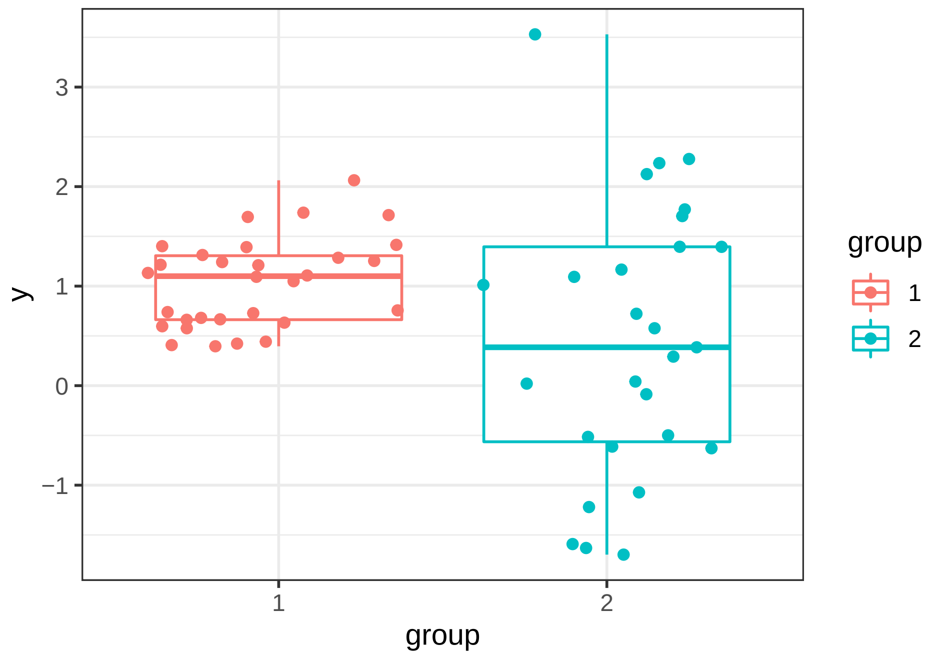
```r
Reg_params <- stan("lm.stan",
                  data=list(N = n,
                            y = y,
                            x = x),
                  iter = 2000)
```

```r
print(Reg_params, pars = c('alpha', 'beta','sigma'))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## alpha -0.03    0.00 0.20 -0.45 -0.16 -0.03 0.11  0.35  1709    1
## beta   4.95    0.01 0.35  4.31  4.72  4.95 5.18  5.64  1791    1
## sigma  1.02    0.00 0.07  0.89  0.97  1.02 1.07  1.18  2134    1
##
## Samples were drawn using NUTS(diag_e) at Fri Apr 22 09:40:38 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Part 2. Bayesian "t-test" The standard 2-sample t-test, typically has an assumption of constant variance between the groups. However, consider simulated data where the variance terms are different for each group.

```
n1 <- 30
mu1 <- 1
sigma1 <- .5
y1 <- rnorm(n1, mu1, sigma1)

n2 <- 27
mu2 <- .5
sigma2 <- 1.5
y2 <- rnorm(n2, mu2, sigma2)


tibble(y = c(y1, y2), group = factor(c(rep(1, n1), rep(2, n2)))) %>%
  ggplot(aes(y = y, x = group, color = group)) +
  geom_boxplot() + theme_bw() +
  geom_jitter()
```



The default settings for the `t.test()` function do not account for different variances. There is an option for non-equal variances, but isn't necessarily clear what the procedure does and it doesn't directly return estimated variances.

```
t.test(y1, y2)
```

```
##
##  Welch Two Sample t-test
##
## data:  y1 and y2
## t = 2.1202, df = 31.125, p-value = 0.04206
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.02226184 1.14335386
## sample estimates:
## mean of x mean of y
```

```
## 1.034236 0.451428
```

Thus, we can easily construct this procedure in Stan. Consider the following code for a two-sample t-test.

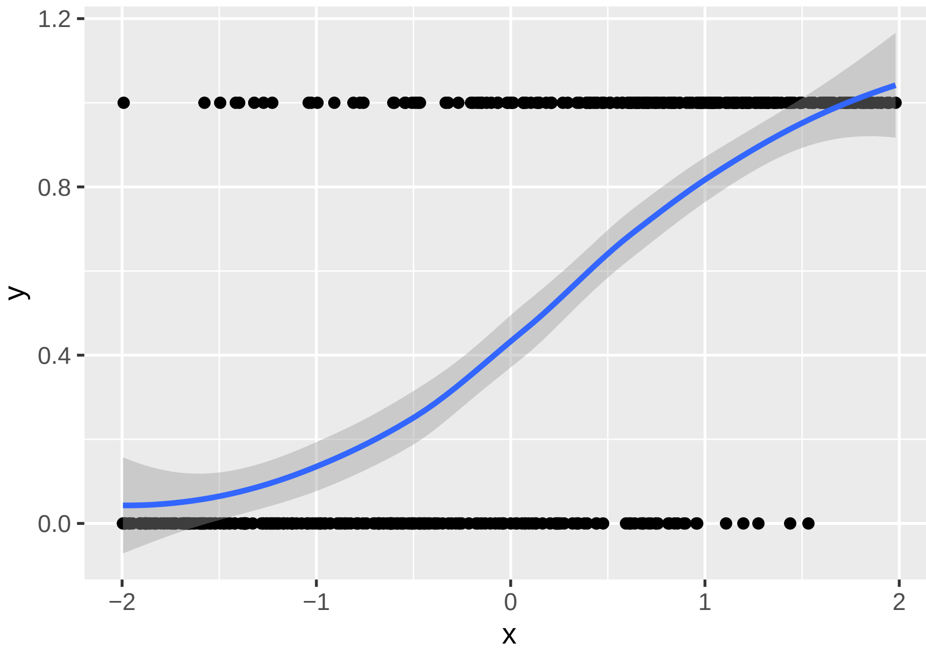
```
data {
  int<lower=1> n1; // number of observations in group 1
  vector[n1] y1; // observations from group 1
  int<lower=1> n2; // number of observations in group 1
  vector[n2] y2; // observations from group 1

}
parameters {
  real<lower=0> sigma; // variance parameter
  real<lower=0> mu1; // group 1 mean
  real<lower=0> mu2; // group 2 mean
}
transformed parameters{
  real diff;
  diff = mu1 - mu2;
}
model {
  y1 ~ normal(mu1, sigma);
  y2 ~ normal(mu2, sigma);
}
```

```
ttest_params <- stan("t_test.stan",
                  data=list(n1 = n1,
                            y1 = y1,
                            n2 = n2,
                            y2 = y2),
                  iter = 2000)

print(ttest_params)
```

Q1: Update the code to allow for different variances between the two groups.

Part 3. Logistic Regression Similarly for a logistic regression model with a single continuous variable.

```
n <- 500
beta <- 2
x <- runif(n, -2, 2)
p <- invlogit(x * beta)
y <- rbinom(n, 1, p)

logistic_dat <- tibble(x = x, y = y)
logistic_dat %>%
  ggplot(aes(y = y, x = x)) +
  geom_point() +
  geom_smooth(formula = 'y~x', method = 'loess')
```



We have used both `glm` and `stan_glm` to fit these models.

```
logistic_dat %>% glm(y ~ x, data = ., family = binomial) %>% display()
```

```
## glm(formula = y ~ x, family = binomial, data = .)
##             coef.est coef.se
## (Intercept) -0.16     0.13
## x            1.82     0.15
## ---
##   n = 500, k = 2
##   residual deviance = 407.0, null deviance = 692.9 (difference = 285.9)
```

```
logistic_dat %>% stan_glm(y ~ x, data = ., family = binomial, refresh = 0) %>% print(digits = 2)
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      y ~ x
##  observations: 500
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) -0.16   0.12
## x            1.83   0.15
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Similarly Stan code can be used to fit a logistic regression model.

```
data {
  int <lower = 0> N;
  int <lower = 0, upper = 1> y [N];
  vector [N] x;
}
```

```
parameters {
  real alpha;
  real beta;
}

model {
  y ~ bernoulli_logit(alpha + beta * x);

  // alpha ~ normal(0, 1);
  // beta ~ normal(1, 1);
}
```

```
log_params <- stan("logistic.stan",
                   data=list(N = n,
                             y = y,
                             x = x),
                   iter = 2000)
```

```
print(log_params, pars = c('alpha', 'beta'))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## alpha -0.16       0 0.13 -0.41 -0.25 -0.16 -0.08  0.08  3182    1
## beta   1.83       0 0.15  1.56  1.73  1.83  1.93  2.15  2867    1
##
## Samples were drawn using NUTS(diag_e) at Fri Apr 22 09:43:22 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Q2: Update the STAN code to place priors on alpha and beta.

Part 4. Hierachical Logistic Regression The stan reference book contains code for a hierarchical logistic regression model https://mc-stan.org/docs/2_29/stan-users-guide/hierarchical-logistic-regression.html.

```
data {
  int<lower = 0> K;
  int<lower = 0> N;
  int<lower = 1, upper = K> kk[N];
  vector[N] x;
  int<lower = 0, upper = 1> y[N];
}
parameters {
  matrix[K,2] beta;
  vector[2] mu;
  vector<lower=0>[2] sigma;
}
model {
  mu ~ normal(0, 2);
  sigma ~ normal(0, 2);
  for (i in 1:2)
    beta[ , i] ~ normal(mu[i], sigma[i]);
```

```
  y ~ bernoulli_logit(beta[kk, 1] + beta[kk, 2] .* x);
}
```

Q3: Simulate hierarchical logistic regression data