HW1

HW1

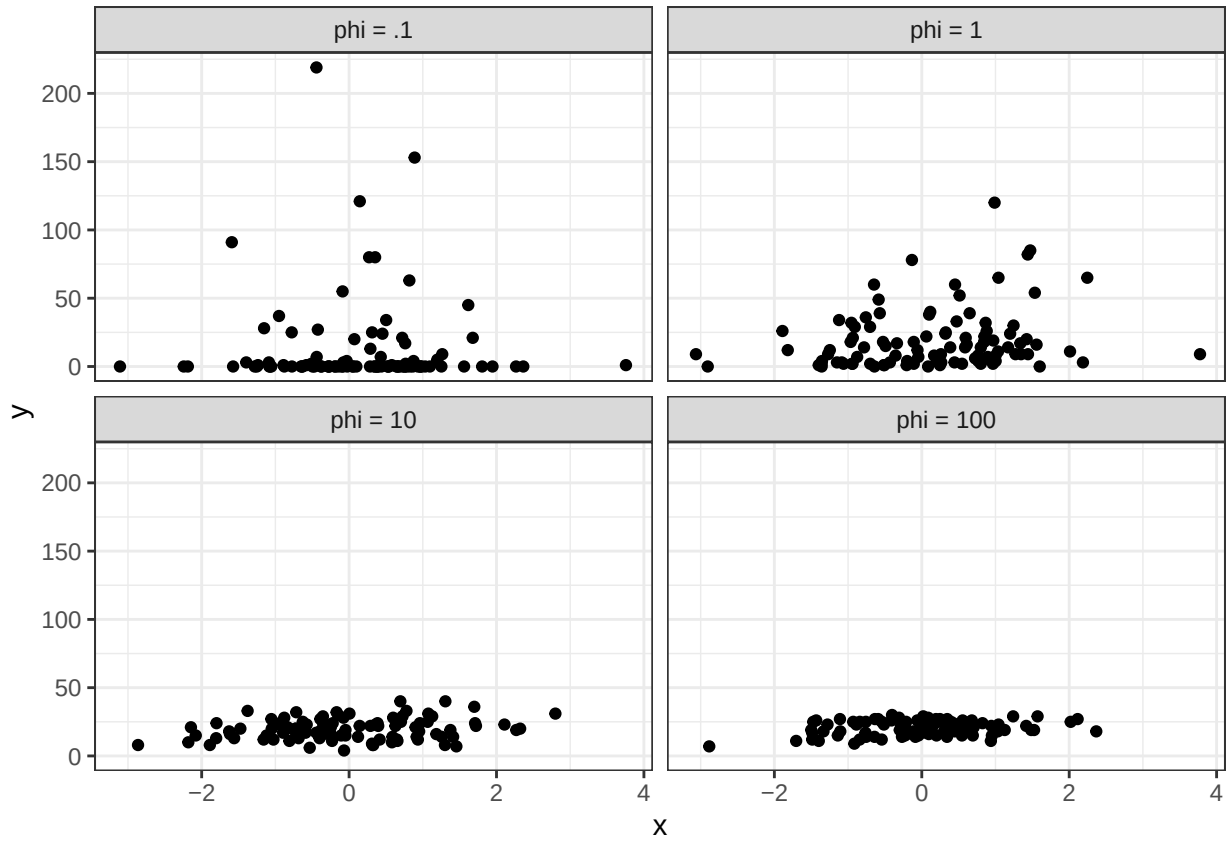
Q1.

a. (4 points) Write an R function or chunk of code to simulate data from a negative binomial regression model using a single continuous covariate and a user specified value of ϕ.

```
sim_nb <- function(n, phi, beta0 = 3, beta1 = .1){
  x <- rnorm(n)
  mu_vec <- exp(beta0 + x * beta1)
  return(tibble(x = x, y =  rnbinom(n, mu = mu_vec, size = phi)))
}
```

b. (4 points) Use the following four values of $\phi : \{.01, .1, 1, 10\}$ and simulate a dataset. Create a paneled figure that shows y and x for each scenario.

```
n <- 100
phi_.1 <- sim_nb(n, phi = .1)
phi_1 <- sim_nb(n, phi = 1)
phi_10 <- sim_nb(n, phi = 10)
phi_100 <- sim_nb(n, phi = 100)
tibble(x = c(phi_.1$x, phi_1$x, phi_10$x, phi_100$x),
       y = c(phi_.1$y, phi_1$y, phi_10$y, phi_100$y),
       group = rep(c('phi = .1', 'phi = 1', "phi = 10", "phi = 100"), each = n)) %>%
  ggplot(aes(x = x, y=y)) +
  geom_point() + facet_wrap(.~group) +
  theme_bw()
```



c. (4 points) Use `stan_glm()` to fit Poisson models and Negative Binomial models for each of the four simulated datasets. Create a table or figure that contains the intercept and slope coefficient for each outcome. Then comment on the implications of your results.

```
nb_.1 <- stan_glm(y ~ x, data = phi_.1, family = neg_binomial_2(link = 'log'))
pois_.1 <- stan_glm(y ~ x, data = phi_.1, family = poisson(link = 'log'))

nb_1 <- stan_glm(y ~ x, data = phi_1, family = neg_binomial_2(link = 'log'))
pois_1 <- stan_glm(y ~ x, data = phi_1, family = poisson(link = 'log'))

nb_10 <- stan_glm(y ~ x, data = phi_10, family = neg_binomial_2(link = 'log'))
pois_10 <- stan_glm(y ~ x, data = phi_10, family = poisson(link = 'log'))

nb_100 <- stan_glm(y ~ x, data = phi_100, family = neg_binomial_2(link = 'log'))
pois_100 <- stan_glm(y ~ x, data = phi_100, family = poisson(link = 'log'))

tibble(beta0 = c(nb_.1$coefficients[1], pois_.1$coefficients[1],
                 nb_1$coefficients[1], pois_1$coefficients[1],
                 nb_10$coefficients[1], pois_10$coefficients[1],
                 nb_100$coefficients[1], pois_100$coefficients[1]),
       beta0_se = c(nb_.1$ses[1], pois_.1$ses[1],
                 nb_1$ses[1], pois_1$ses[1],
                 nb_10$ses[1], pois_10$ses[1],
                 nb_100$ses[1], pois_100$ses[1]),
       beta1 = c(nb_.1$coefficients[2], pois_.1$coefficients[2],
                 nb_1$coefficients[2], pois_1$coefficients[2],
```

```
              nb_10$coefficients[2], pois_10$coefficients[2],
              nb_100$coefficients[2], pois_100$coefficients[2]),
       beta1_se = c(nb_.1$ses[2], pois_.1$ses[2],
              nb_1$ses[2], pois_1$ses[2],
              nb_10$ses[2], pois_10$ses[2],
              nb_100$ses[2], pois_100$ses[2]),
       model = rep(c('NB',"Pois"), 4),
       phi = rep(c(.1,1,10,100),each = 2)) %>% kable(digits = 2)
```

beta0	beta0_se	beta1	beta1_se	model	phi
2.57	0.31	-0.08	0.37	NB	0.1
2.53	0.03	-0.05	0.03	Pois	0.1
2.88	0.11	0.26	0.11	NB	1.0
2.89	0.02	0.22	0.02	Pois	1.0
2.97	0.04	0.08	0.04	NB	10.0
2.97	0.02	0.08	0.02	Pois	10.0
3.01	0.04	0.07	0.04	NB	100.0
3.01	0.02	0.07	0.03	Pois	100.0

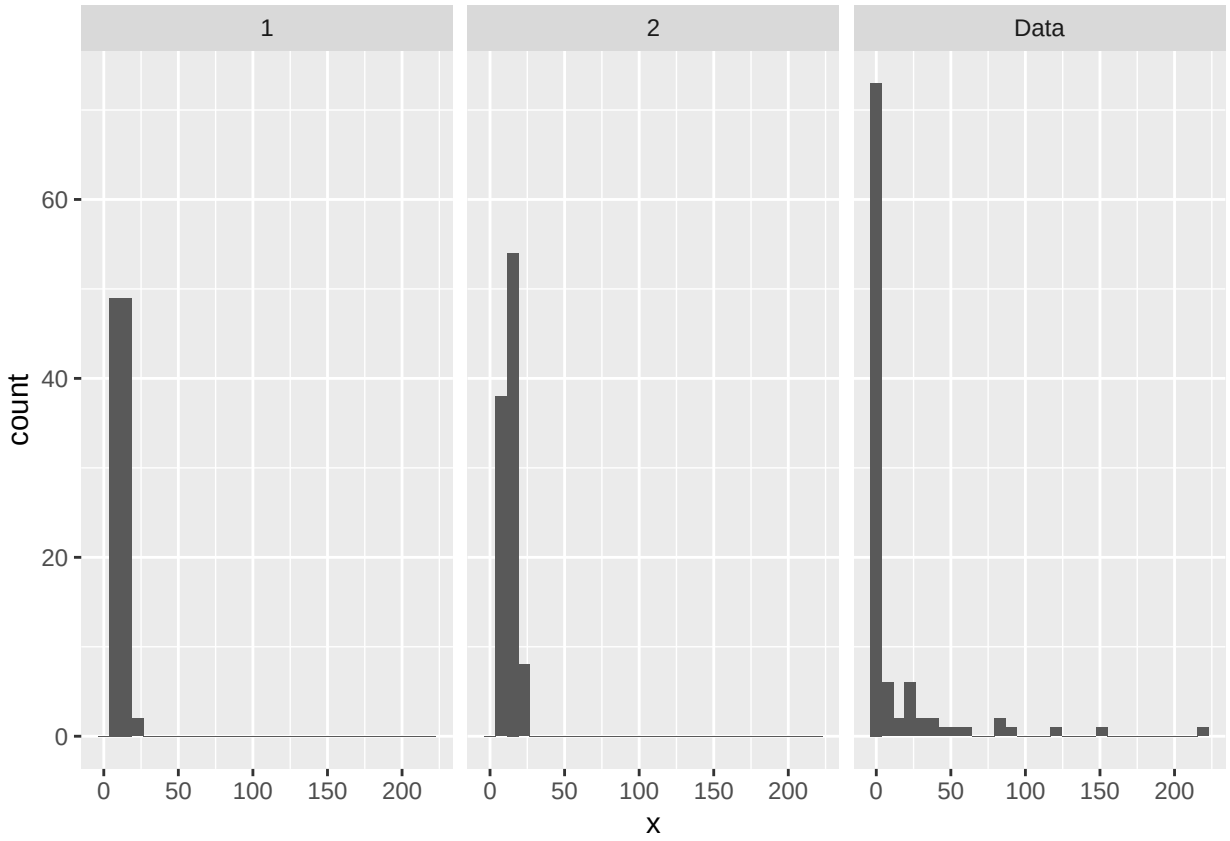
The point estimates are similar, but the standard errors are substantially larger for the negative-binomial model. These larger errors are more appropriate and result in valid coverage for these intervals.

d. (4 points) Use `posterior_predict()` and posterior predictive checks to further interrogate the model fit (using the Poisson models)

```
pois_pred_.1 <- posterior_predict(pois_.1)

tibble(x = c(phi_.1$y, as.numeric(pois_pred_.1[1:2,])),
       sim = c(rep('Data', n),rep(c('1',"2"),n))) %>%
  ggplot(aes(x = x)) + geom_histogram() +
  facet_wrap(.~sim)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The poisson model does a poor job capturing the variability in the data.

Q2.

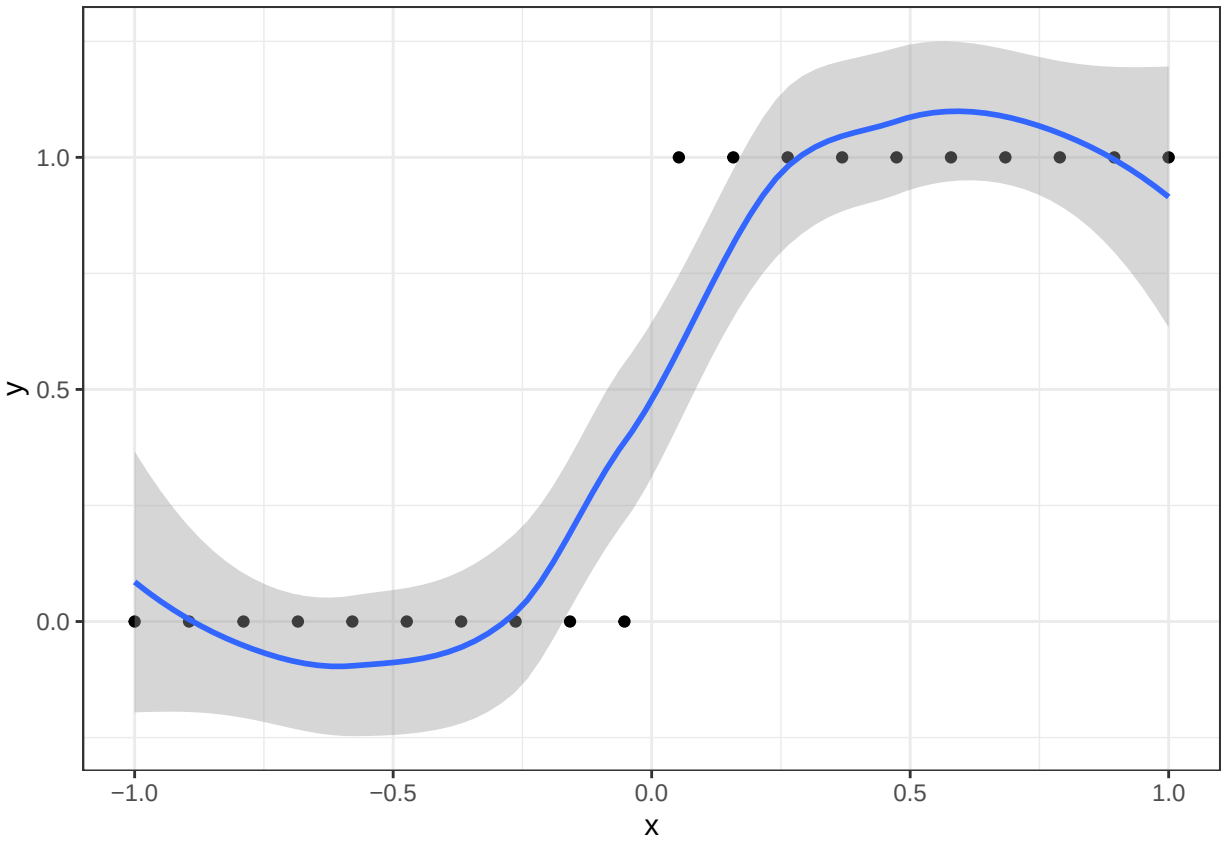
With binary regression, "separation" is a common problem. This occurs when a continuous predictor is perfectly separated with all zeros below a certain point and all zeros above a certain point. See the simulated data below for an example.

```
x <- seq(-1, 1, length.out = 20)
y <- rep(c(0,1), each = 10)

df_sep <- tibble(x=x, y=y)

df_sep %>% ggplot(aes(y=y, x=x)) + geom_point() + theme_bw() + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



a. (4 points) Using the figure above - and any other references - define separation and describe why it is problematic.

Separation results when there is complete separation in the data - meaning all responses below a threshold value are successes (or failures) where all responses above the threshold are the opposite value. The issue is that this that the model could be approximated with a step function that requires a vertical line (slope = infinity) which cannot (should not?) be achieved with our link functions.

b. (4 points) Use both `glm` and `stan_glm` to fit the data. Identify the differences in the model output and discuss why they might differ.

```
df_sep %>% stan_glm(y ~ x, family = binomial(link = "logit"), data = ., refresh = 0)
```

```
## stan_glm
##  family:       binomial [logit]
##  formula:      y ~ x
##  observations: 20
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 0.0    0.8
## x           7.1    2.4
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

```
df_sep %>% glm(y ~ x, family = binomial(link = "logit"), data = .)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:  glm(formula = y ~ x, family = binomial(link = "logit"), data = .)
##
## Coefficients:
## (Intercept)            x
##   4.618e-09    3.921e+02
##
## Degrees of Freedom: 19 Total (i.e. Null);  18 Residual
## Null Deviance:       27.73
## Residual Deviance: 4.357e-09     AIC: 4
```

The GLM approach has numerical issues, this is alleviated with the Bayesian approach where the prior regularizes the estimates.