505 Assignment 8

David Agyemfra Atakora

November 06th, 2022

Question 1 (from 2020 midterm)

Using a candy dataset (https://math.montana.edu/ahoegh/teaching/stat446/candy-data.csv), define and fit a regression model to understand the relationship between winpercent and pricepercent, chocolate, and caramel. More insight into the data is available at https://fivethirtyeight.com/videos/the-ultimate-halloween-candy-power-ranking/.

a. Write out the model and define all of the coefficients. (2 points)

Model:

$$winpercent = \beta_0 + \beta_1 x_{pricepercent} + \beta_2 x_{chocolate=1} + \beta_3 x_{caramel=1} + \epsilon$$

where;
$winpercent$ = The overall win percentage

β_0 = Represents the predicted overall win percentage for a zero-unit price percentile when the candy doesn't both contain caramel and chocolate.

β_1 = Comparing the overall win percentage of the candy with or without chocolate or caramel, but a difference of 1 point change in pricepercent, we would expect to see a difference of β_1 points in the overall win percentage.

β_2 = Comparing the overall win percentage of the candy with the same pricepercent, but who differs in whether there is chocolate or caramel, the model predicts an expected difference of β_2 in the overall win percentage.

β_3 = Comparing the overall win percentage of the candy with the same pricepercent, but who differs in whether there is chocolate or caramel, the model predicts an expected difference of β_3 in the overall win percentage.

$x_{pricepercent}$ = The unit price percentile.

$x_{chocolate=1}$ = Is an indicator function for whether the candy contains chocolate.

$x_{caramel=1}$ = Is an indicator function for whether the candy contains caramel.

$\epsilon \sim N(0,\, \sigma^2)$

b. Fit the model with software of your choice and print the results. (2 points)

```
set.seed(2022)
fit<- stan_glm(winpercent ~ pricepercent + chocolate + caramel, data=candy, refresh = 0)
print(fit)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      winpercent ~ pricepercent + chocolate + caramel
##  observations: 85
##  predictors:   4
## ------
##              Median MAD_SD
## (Intercept)  41.5    2.4
## pricepercent  1.4    5.2
## chocolate    18.0    2.8
## caramel       2.2    3.6
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 11.6    0.9
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

c. Summarize your results from part 2, in a way that Willy Wonka could understand. (2 points)

The first column shows estimates; 41.5, 1.4, 18.0 and 2.2 are the coefficients in the fitted line, $winpercent = 41.5 + 1.4x_{pricepercent} + 18.0x_{chocolate=1} + 2.2x_{caramel=1}$. The second column displays uncertainties in the estimates using the median absolute deviations. The last output line displays the estimation and uncertainty of σ, the size of the data variation that the regression model is unable to explain.(That is; the deviation of the regression line's upper and lowermost points). 41.5 is interpreted as the average overall win percentage when there is no pricepercent and the candy has doesn't contains chocolate and caramel. 1.4 represent the mean increase of the overall win percentage for every additional one unit in price percent with no chocolate and caramel in the candy. 18.0 explains the difference between the overall win percentage for a candy with chocolate and without caramel with a zero-unit pricepercent, and candy without both chocolate and caramel with a zero-unit pricepercent. Lastly, With a zero-unit pricepercent, the 2.2 also explains the difference between a candy with caramel and no chocolate and a candy with without chocolate and caramel.

d. Using your model from part b, create the candy with the highest win percentage. Then specify the levels of the predictors and create a predictive distribution for an individual type of candy with those features. (2 points)

Coefficients of the models

```
set.seed(505)
coefs_fit <- coef(fit)
coefs_fit
```

```
##  (Intercept) pricepercent    chocolate      caramel
##    41.533859     1.382804    17.953546     2.155693
```

Maximum values for the predictors:

```
price_percent <- max(candy$pricepercent)

chocolate <- 1
caramel <- 1

price_percent
```

```
## [1] 0.976
```

```
chocolate
```

```
## [1] 1
```

```
caramel
```

```
## [1] 1
```

Highest win percentage;

```
coefs_fit[1] + coefs_fit[2]*price_percent + coefs_fit[3]*1 + coefs_fit[4]*1
```

```
## (Intercept)
##    62.99272
```

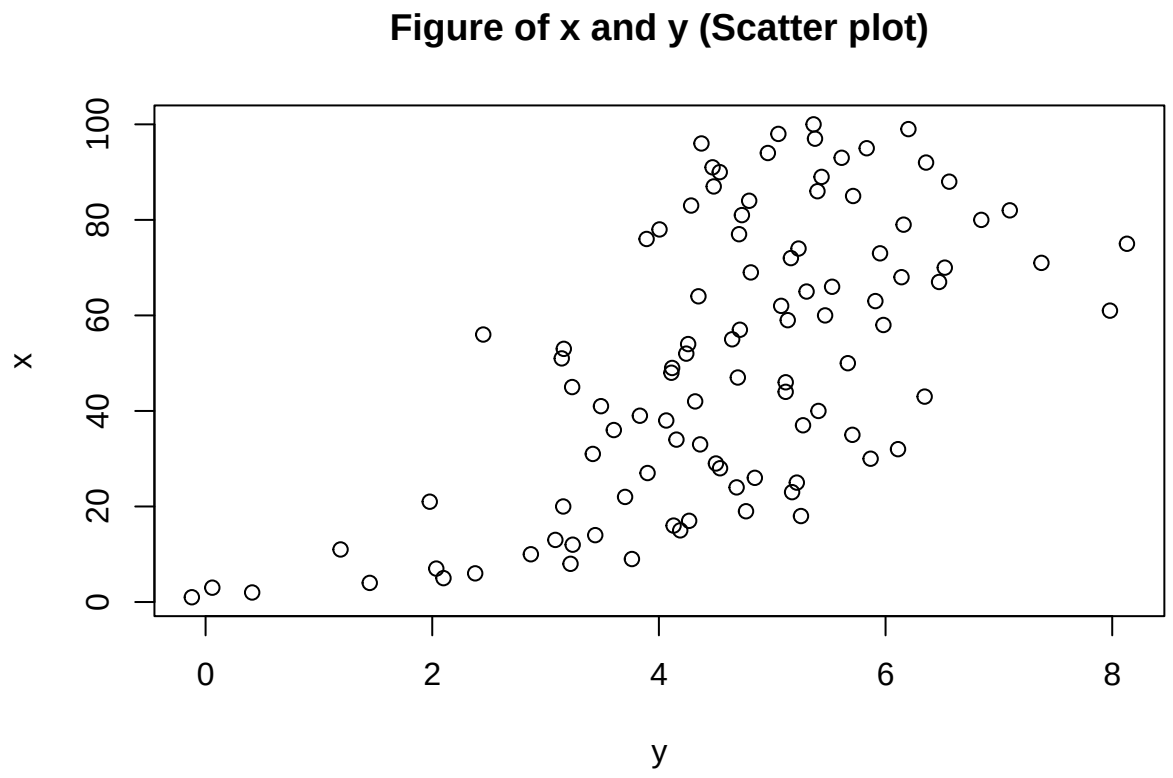
a. (2 points)

Consider the code below. Create a figure of x and y. What linear regression assumption does this data violate?

The linear regression assumption does this data violate is the linearity assumption. This is becuase of the model having a $log(x)$ component in it. From the second figure, there is a presence of curvature as data points increases. Also, the data points are not symmetrically distributed and has a cone shape around the horizontal line in the plot of residuals versus predicted values.

```
n <- 100
x <- seq(1,100, length.out = n)
sigma <- 1
beta <- c(1, 1)
x_star <- log(x)
y <- rnorm(n, beta[1] + beta[2] * x_star, sd = sigma)
combined <- tibble(y = y, x = x)

plot(combined, main="Figure of x and y (Scatter plot)")
```




```
figure<-ggplot(combined, aes(y = y, x = x)) +
  geom_point()+
  geom_smooth(method = 'loess', formula = 'y~x') +
  labs(title="Figure of x and y",
       y="x", x="x",
       caption="Figure of x and y")+
  theme_minimal()
figure
```

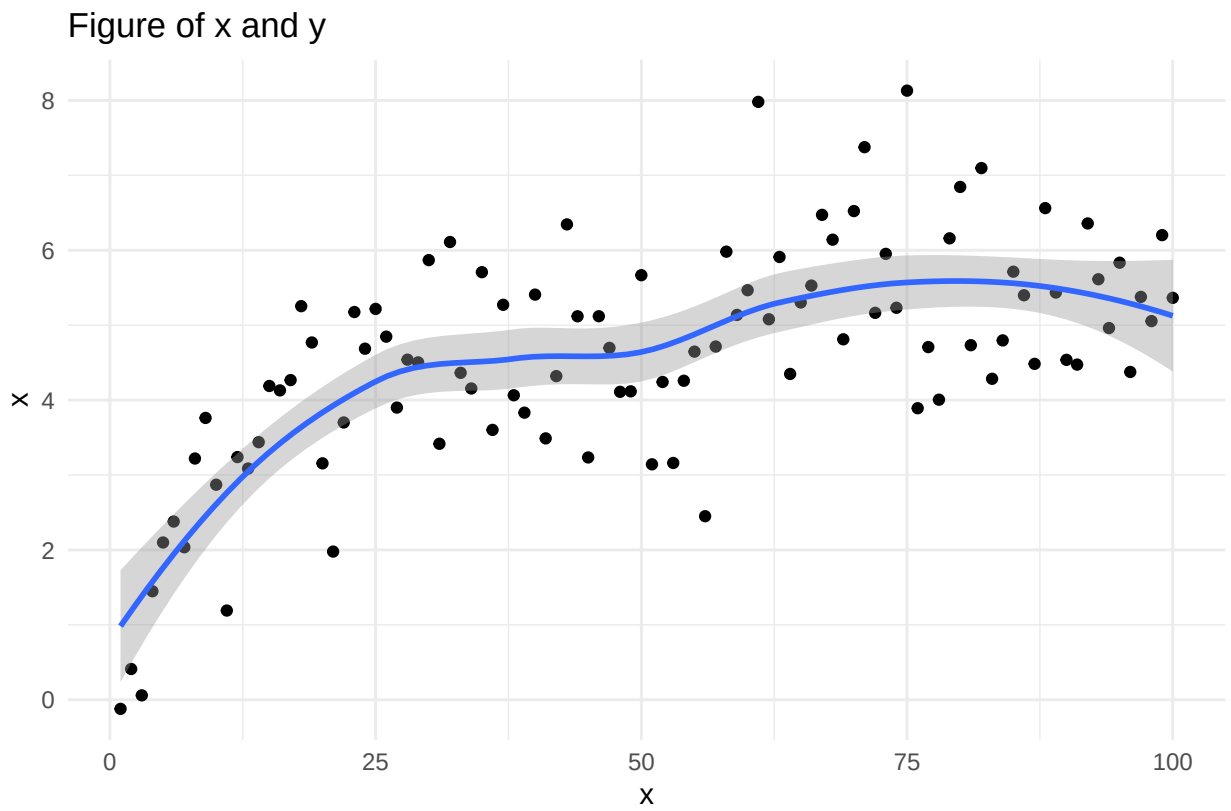


Figure of x and y

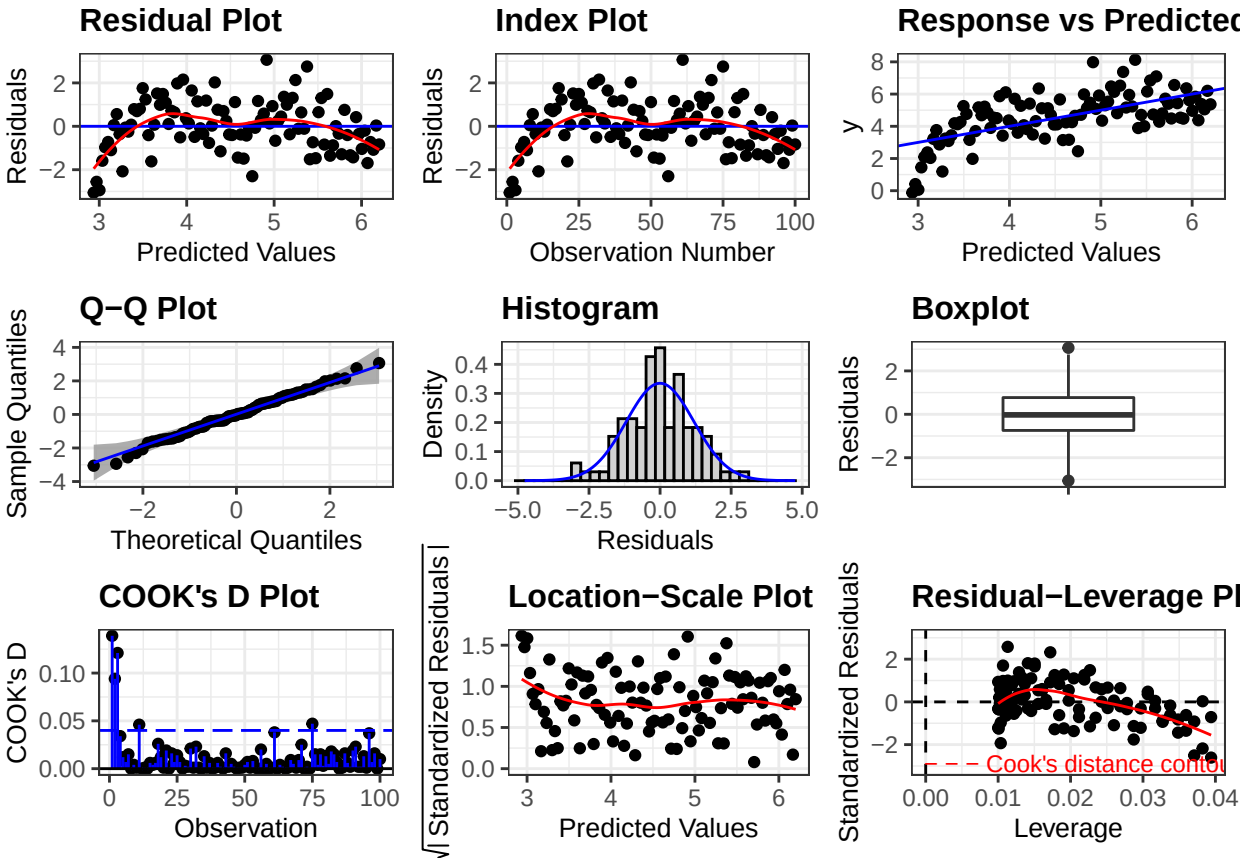
```
#Testing the assumptions
lm1 <- lm(y~x, data = combined)
summary(lm1)
```

```
##
## Call:
## lm(formula = y ~ x, data = combined)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.05765 -0.74278 -0.02619  0.77070  3.06531
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.903490   0.241208  12.037  < 2e-16 ***
## x           0.032970   0.004147   7.951 3.23e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.197 on 98 degrees of freedom
## Multiple R-squared:  0.3921, Adjusted R-squared:  0.3859
## F-statistic: 63.22 on 1 and 98 DF,  p-value: 3.229e-12
```

```
#library(devtools)
#devtools::install_github("goodekat/ggResidpanel")
```

```
library(ggResidpanel)
resid_panel(lm1, plots = 'all', smoother = T, qqbands = T)
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



b. (4 points)

How well does a linear regression model (y ~ x) recover the point estimates of beta? Justify your answer by using several (~1000) replications of simulated data.

We can make our model linear by;
1. Finding the log (x).
2. Increasing the number of simulation (~1000).
3. Investigating the betas.

We can observe from the table (At investigating the betas) as the sample size increases, the coefficients of the new linear models approximate to the beta in the initial model.

When n = 1000

```
set.seed(2000)

n <- 1000
x <- seq(1,1000, length.out = n)
sigma <- 1
beta <- c(1, 1)
x_star <- log(x)
y <- rnorm(n, beta[1] + beta[2] * x_star, sd = sigma)
combined <- tibble(y = y, x = log(x))

lm2 <- lm(y~x, data = combined)
summary(lm2)
```

```
##
## Call:
## lm(formula = y ~ x, data = combined)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3345 -0.6662 -0.0026  0.7107  2.9808
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.98801    0.19409    5.09 4.27e-07 ***
## x            1.01119    0.03238   31.23  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.009 on 998 degrees of freedom
## Multiple R-squared:  0.4942, Adjusted R-squared:  0.4937
## F-statistic: 975.1 on 1 and 998 DF,  p-value: < 2.2e-16
```

```
beta_0_2 <- coef(lm2)[1] #beta_0 for model 2
beta_1_2 <- coef(lm2)[2] #beta_1 for model 2

beta_0_2
```

```
## (Intercept)
##   0.9880065
```

```
beta_1_2
```

```
##        x
## 1.011187
```

When n = 10000

```
set.seed(2001)

n <- 10000
x <- seq(1,10000, length.out = n)
sigma <- 1
beta <- c(1, 1)
x_star <- log(x)
y <- rnorm(n, beta[1] + beta[2] * x_star, sd = sigma)
combined <- tibble(y = y, x = log(x))

lm3 <- lm(y~x, data = combined)
summary(lm3)
```

```
##
## Call:
## lm(formula = y ~ x, data = combined)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2065 -0.6793 -0.0038  0.6728  3.4924
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.01116    0.08281   12.21   <2e-16 ***
## x            0.99837    0.01001   99.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9986 on 9998 degrees of freedom
## Multiple R-squared:  0.4987, Adjusted R-squared:  0.4986
## F-statistic:  9944 on 1 and 9998 DF,  p-value: < 2.2e-16
```

```
beta_0_3 <- coef(lm3)[1] #beta_0 for model 3
beta_1_3 <- coef(lm3)[2] #beta_1 for model 3

beta_0_3
```

```
## (Intercept)
##    1.011158
```

```
beta_1_3
```

```
##         x
## 0.9983712
```

3. Investigating the Betas

```
df <- data.frame(Beta=c('Intercept (Beta_0)', 'Intercept (Beta_0)'),
                 n_1000=c('0.98801', '1.01116'),
                 n_10000=c('1.01119', '0.99837'))
df
```

```
##                 Beta  n_1000 n_10000
## 1 Intercept (Beta_0) 0.98801 1.01119
## 2 Intercept (Beta_0) 1.01116 0.99837
```

We can observe from the table (At investigating the betas) as the sample size increases, the coefficients of the new linear models approximate to the beta in the initial model.

c. (4 points)

How well does a linear regression model capture the uncertainty in a predictions for y conditional on

- x = 1
- x = 50
- x = 100

When x = 1

```
set.seed(1001)
n <- 100
x <- seq(1,100, length.out = n)
sigma <- 1
beta <- c(1, 1)
x_star <- log(x)
y <- rnorm(n, beta[1] + beta[2] * x_star, sd = sigma)
combined <- tibble(y = y, x = x)
lm4 <- stan_glm( y ~ x, data = combined, refresh = 0)

new_model <- data.frame(x=1)
y_pred <- posterior_linpred(lm4, newdata = new_model) %>% quantile(prob = c(0.25,0.975))
y_pred
```

```
##      25%    97.5%
## 3.122193 3.791457
```

```
tab<-as.matrix(y_pred)
y <- rnorm(n, beta[1] + beta[2] * log(1), sd = sigma) # set 1 in log(x)
z <-(which(y <= tab[2,]))
q <-which(y >= tab[1,])
prediction_1<-length(which(z %in% q))/n

prediction_1
```

```
## [1] 0.02
```

When x = 50

```
set.seed(1002)
n <- 100
x <- seq(1,100, length.out = n)
sigma <- 1
beta <- c(1, 1)
x_star <- log(x)
y <- rnorm(n, beta[1] + beta[2] * x_star, sd = sigma)
combined <- tibble(y = y, x = x)
lm4 <- stan_glm( y ~ x, data = combined, refresh = 0)
```

```
new_model <- data.frame(x=50)
y_pred <- posterior_linpred(lm4, newdata = new_model) %>% quantile(prob = c(0.25,0.975))
y_pred
```

```
##      25%    97.5%
## 4.706903 4.976972
```

```
tab<-as.matrix(y_pred)
y <- rnorm(n, beta[1] + beta[2] * log(50), sd = sigma) # set 1 in log(x)
z <-(which(y <= tab[2,]))
q <-which(y >= tab[1,])
prediction_1<-length(which(z %in% q))/n

prediction_1
```

```
## [1] 0.07
```

When x = 100

```
set.seed(1003)
n <- 100
x <- seq(1,100, length.out = n)
sigma <- 1
beta <- c(1, 1)
x_star <- log(x)
y <- rnorm(n, beta[1] + beta[2] * x_star, sd = sigma)
combined <- tibble(y = y, x = x)
lm4 <- stan_glm( y ~ x, data = combined, refresh = 0)

new_model <- data.frame(x=100)
y_pred <- posterior_linpred(lm4, newdata = new_model) %>% quantile(prob = c(0.25,0.975))
y_pred
```

```
##      25%    97.5%
## 5.869056 6.403297
```

```
tab<-as.matrix(y_pred)
y <- rnorm(n, beta[1] + beta[2] * log(100), sd = sigma) # set 1 in log(x)
z <-(which(y <= tab[2,]))
q <-which(y >= tab[1,])
prediction_1<-length(which(z %in% q))/n

prediction_1
```

```
## [1] 0.22
```

```
df <- data.frame(x=c('Percentage'),
                 x_1=c('0.02'),
                 x_50=c('0.07'),
                 x_100=c('0.22'))
df
```

```
##            x  x_1 x_50 x_100
## 1 Percentage 0.02 0.07  0.22
```

As x increases, more percentage of points are captured in the interval. The interval is provided by the posterior_linpred function but the model still doesn't predict well.