green_milk_bayes

2026-01-16

```r
# Load packages
library(bayesrules)
library(rstanarm)
```

```
## Loading required package: Rcpp

## This is rstanarm version 2.32.1

## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend calling

##   options(mc.cores = parallel::detectCores())
```

```r
library(bayesplot)
```

```
## This is bayesplot version 1.12.0

## - Online documentation and vignettes at mc-stan.org/bayesplot

## - bayesplot theme set to bayesplot::theme_default()

##    * Does _not_ affect other ggplot2 plots

##    * See ?bayesplot_theme_set for details on theme setting
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(tidybayes)
library(broom.mixed)
library(ggplot2)
```

```r
data <- read.csv("tea_sales.csv")
data$date <- as.Date(data$date)
data$weekend_holiday <- factor(
  data$weekend_holiday,
  levels = c(0, 1)
)
data <- na.omit(data)
```

```r
green <- data %>%
  filter(tea == "green")

milk <- data %>%
  filter(tea == "milk")
```

I work at a restaurant where one of my responsibilities is to make tea orders as they come in.

To help me with my job, I want to create a model to predict daily tea sales to help me meet customer demands. If my model predicts that many customers will order tea today, I would want to prepare more tea in advance. On the other hand, if my model predicts that only a few customers will order tea today, then I would not want to prepare so much tea to minimize waste.

I collected daily data over about four months on the number of customers, cups of green tea sold, cups of milk tea sold, and the status of that day as a weekend or holiday.

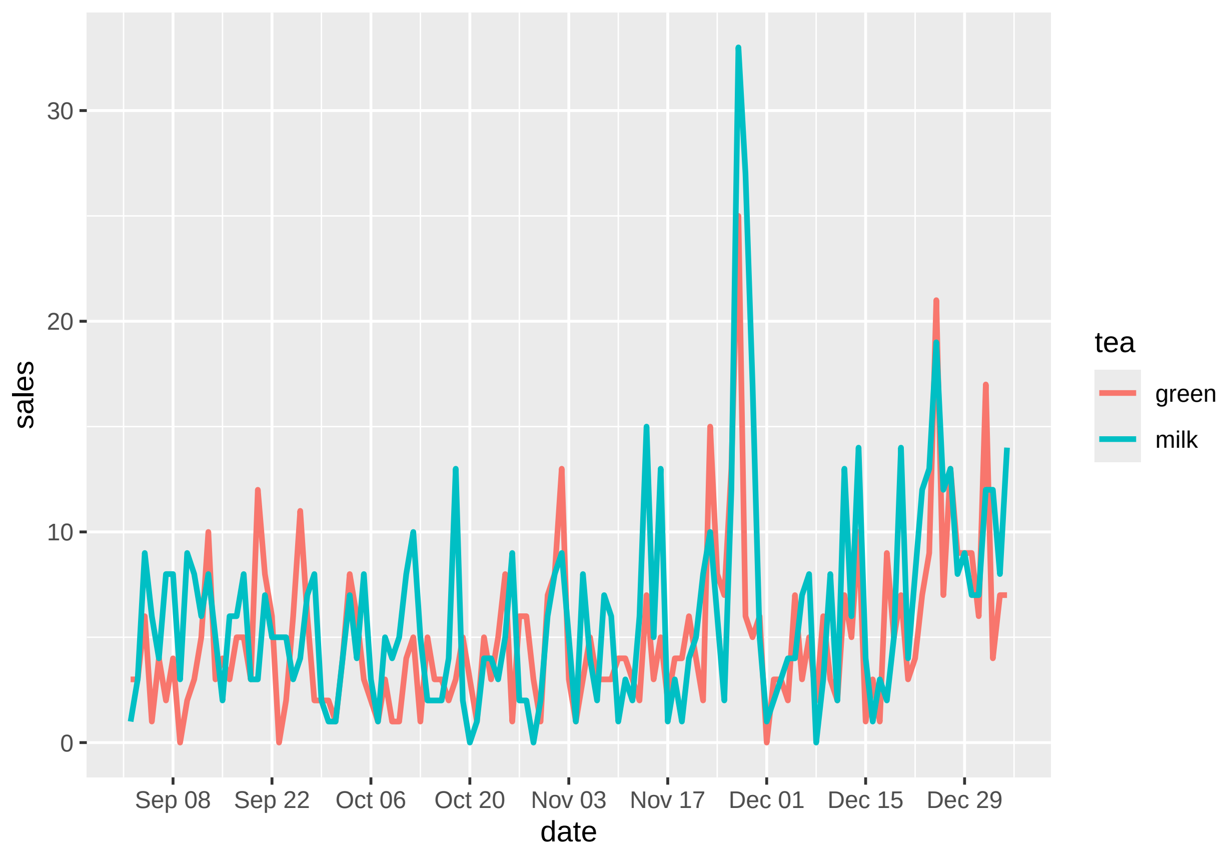
First, we can look at how some of these variables are related to each other. From intuition, it makes sense that tea sales would be correlated with the number of customers, since more customers means more opportunities to make a sale. However, at the same of prediction (the start of the day), we do not know how many customers there will be for that day. Instead of looking at the number of customers today, we can instead look at if historic numbers of customers can help us predict tea sales today. Looking at the number of customers yesterday might be a good starting point, with the reasoning that yesterday's customers and today's customers may be correlated. Similarly, we can also look at yesterday's tea sales.

In addition to these lag variables, we can also explore whether sales increase during weekends and holidays. On weekends and holidays, our restaurant has a special menu, resulting in a higher volume of customers. It might be the case that tea sales are greater on weekends and holidays than regular weekdays.

Of special interest is differentiating between green tea and milk tea sales. Since the preparation steps for milk tea and green tea are different, I need to know how many sales of each separate type of tea to expect.

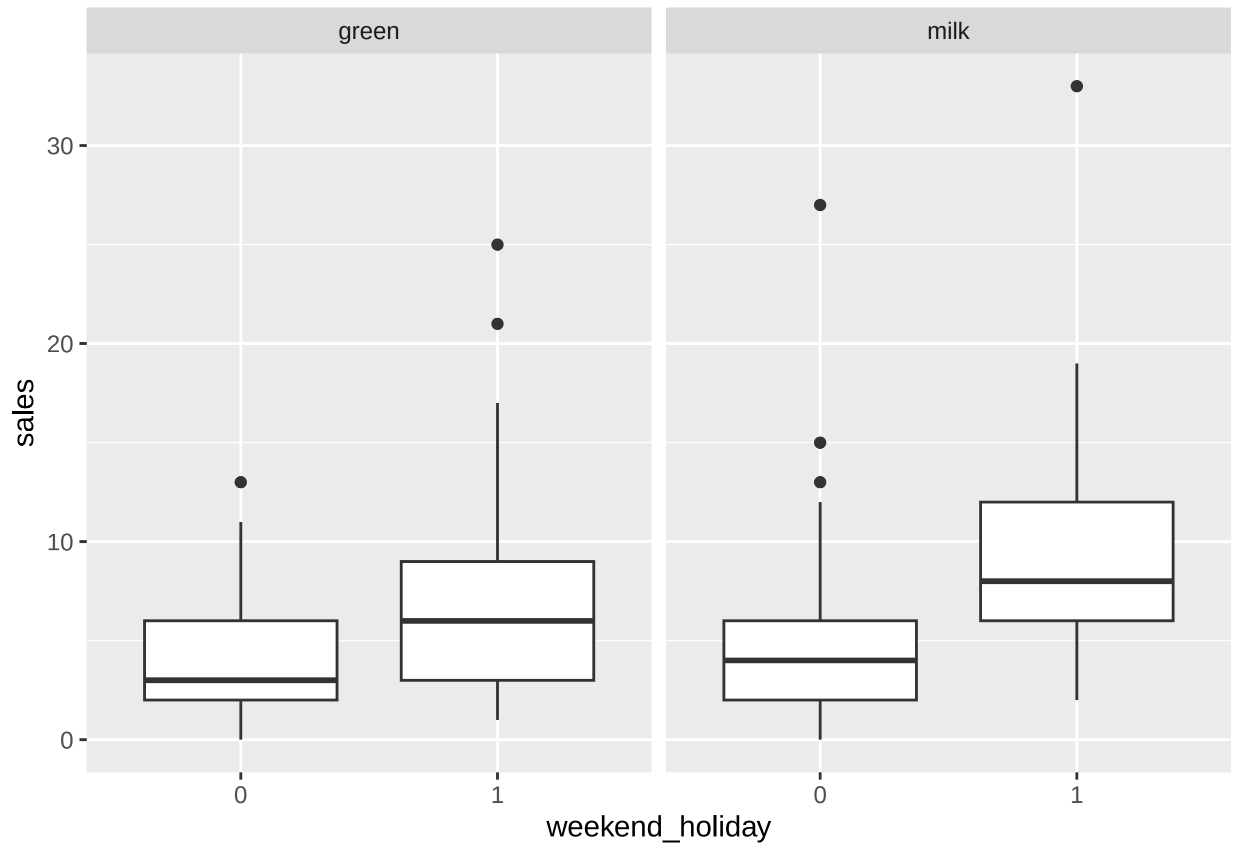
To start, we look at how tea sales have changed over time.

```r
ggplot(data, aes(x = date, y = sales, color = tea, group=tea)) +
  geom_line(linewidth = 1) +
  scale_x_date(date_breaks = "2 weeks", date_labels = "%b %d")
```



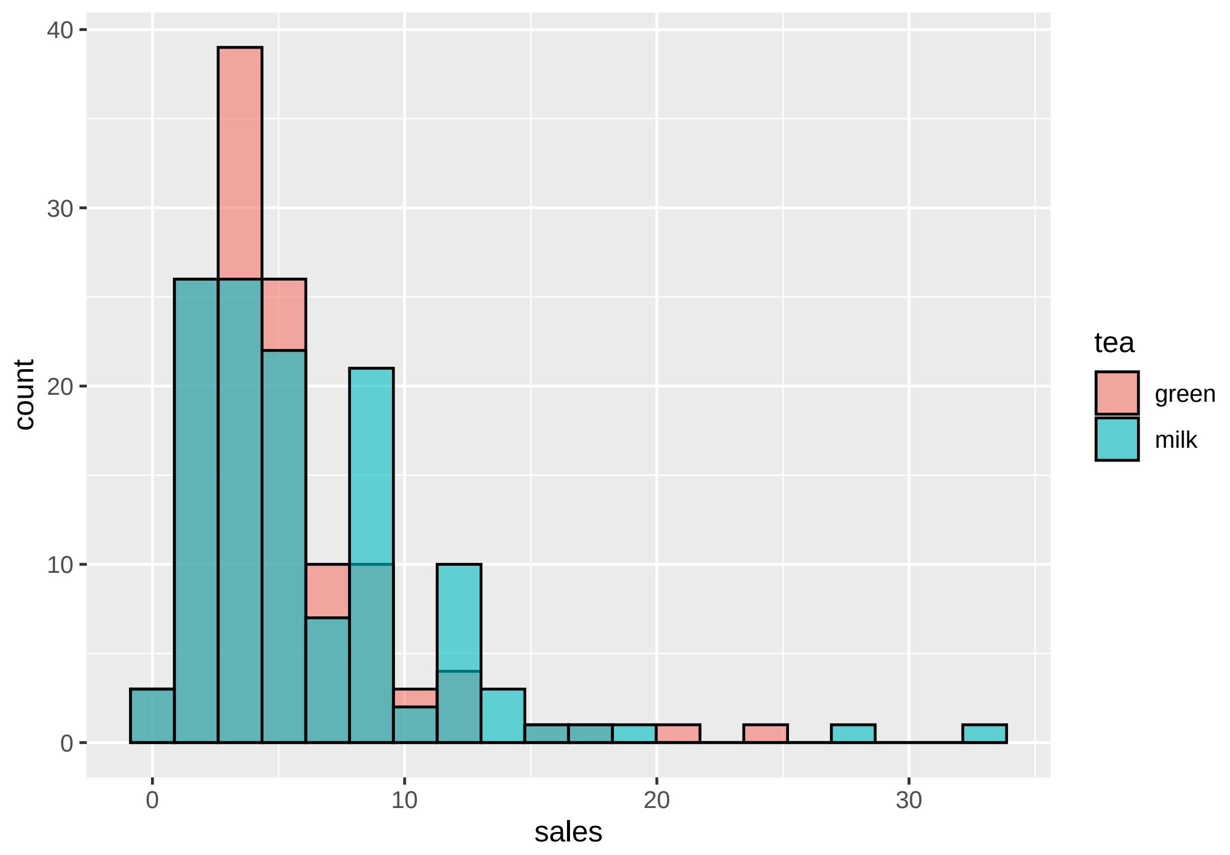
From the graph, green tea and milk tea sales are on a similar level to each other over time. We also see a large spike in sales on what appears to be days near holidays, which suggests that holiday_weekend may be a useful predictor.

```
ggplot(data, aes(x = weekend_holiday, y = sales)) +
  geom_boxplot() +
  facet_wrap(~ tea)
```



From the boxplots, it looks like sales are indeed higher on weekends and holidays, especially so for milk tea. Now let's look at the general distribution of our data.

```r
ggplot(data, aes(x = sales, fill = tea)) +
  geom_histogram(
    bins = 20,
    position = "identity",
    alpha = 0.6,
    color = "black"
  )
```

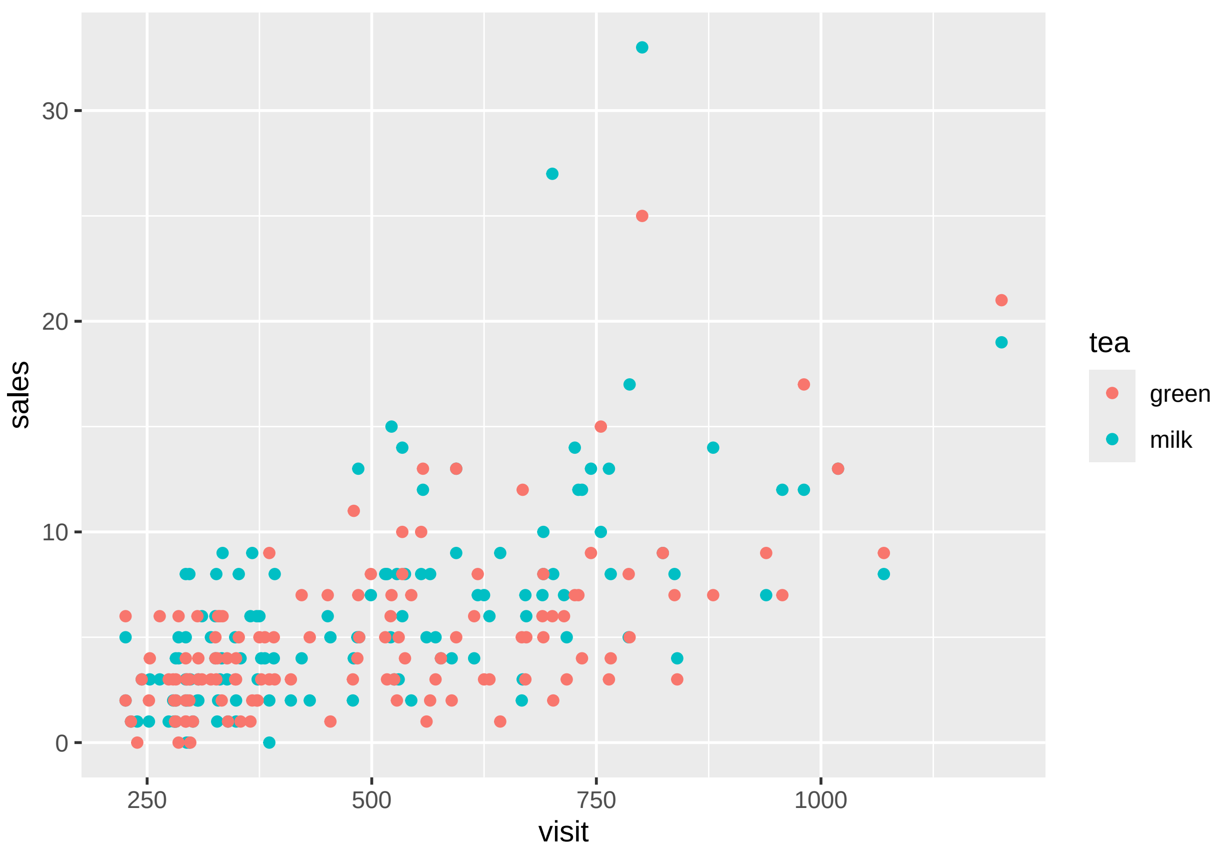


We already know that our data are non-negative counts. Looking at the distribution, we can see that our data is right-skewed with most observations being low counts and a few observations having very high counts. This structure suggests that a Poisson or negative binomial model could be a good fit.

We also see that there is a large peak for green tea sales at around 4. Comparatively, milk tea sales are more spread apart with sales around 2 to 9 having roughly the same count. This suggests that on average, milk tea sales are greater than green tea sales.

Taking a look at the number of customers vs sales, we can see the expected strong relationship.

```r
ggplot(data, aes(x = visit, y = sales, color = tea)) +
  geom_point()
```

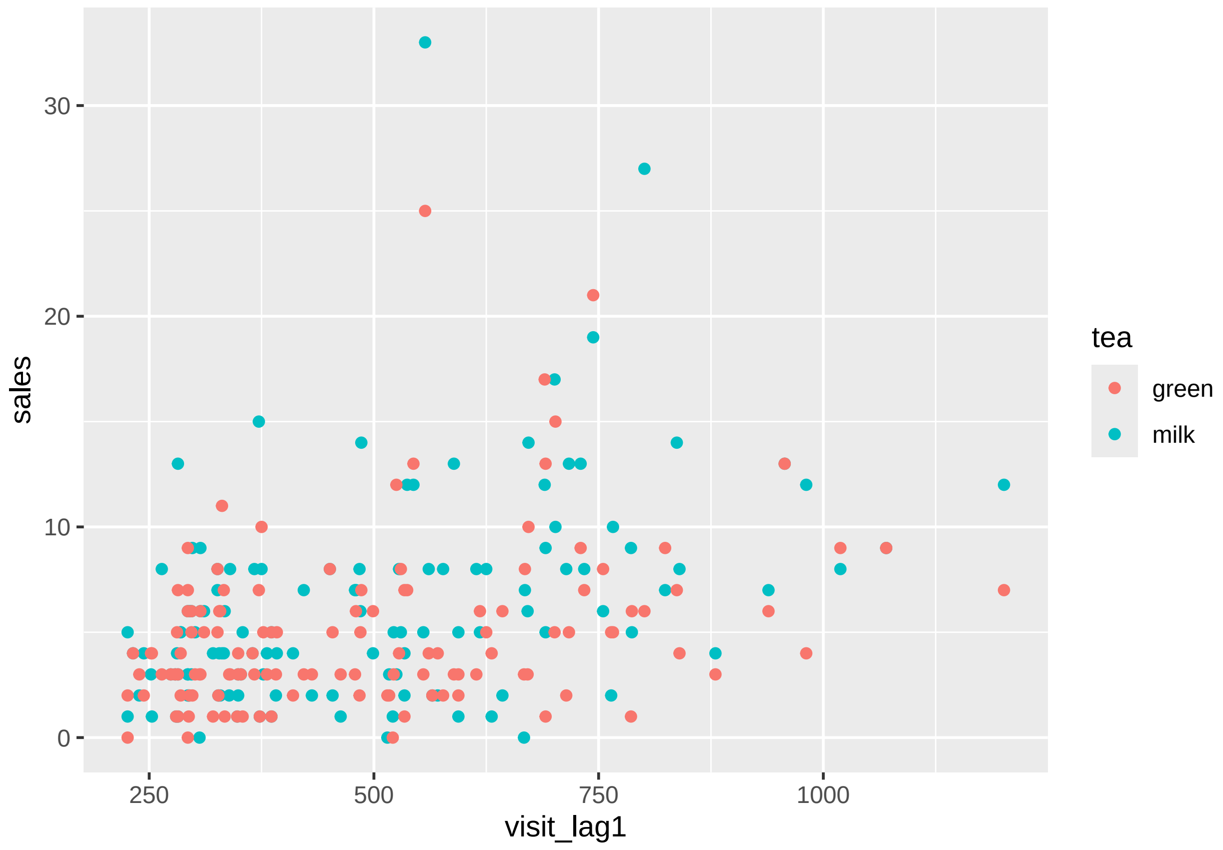


```r
cor(data$sales, data$visit)
```

```
## [1] 0.5978092
```

However, as we noted, we can't use the number of customers today to predict today's sales, so we have to look at the number of customers yesterday. We can still observe a positive relationship, but it isn't as strong. Still, it could be a useful predictor.

```r
ggplot(data, aes(x = visit_lag1, y = sales, color = tea)) +
  geom_point()
```

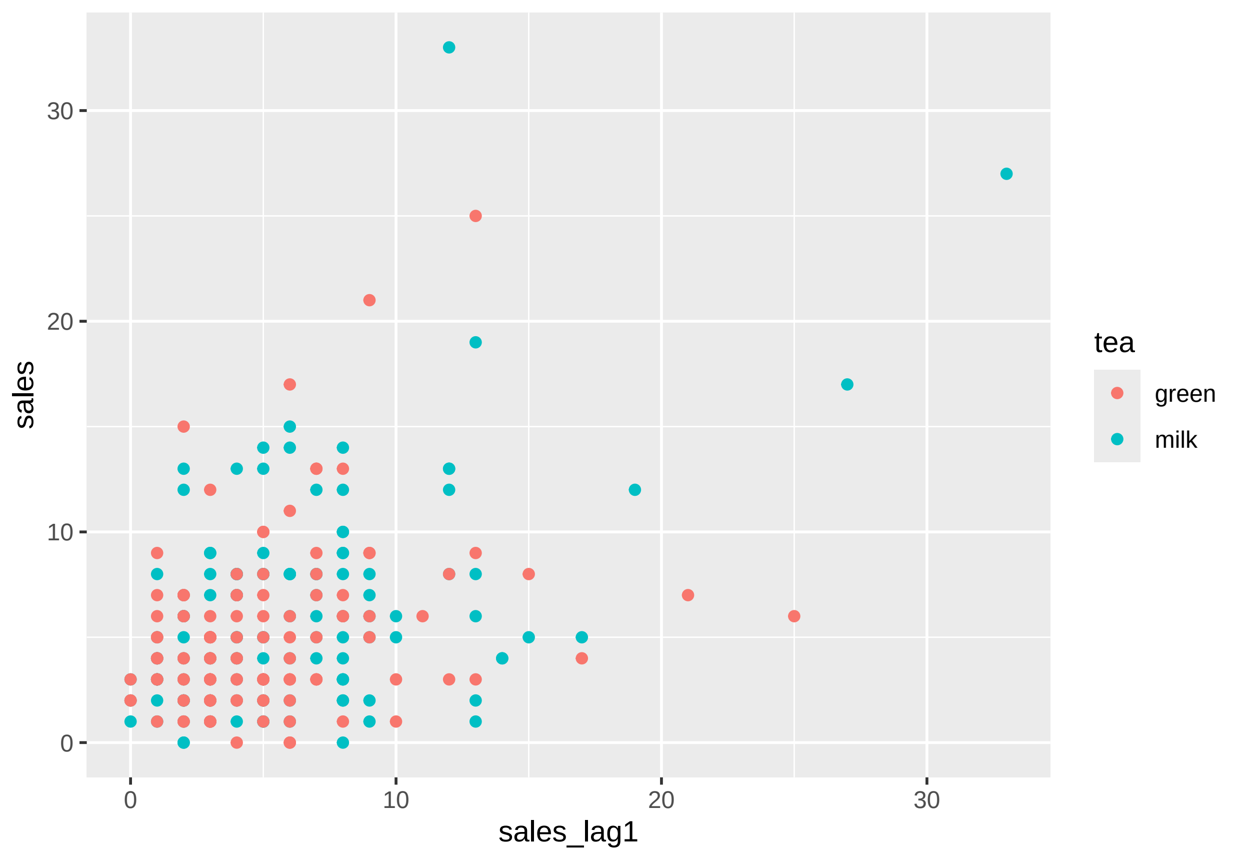


```r
cor(data$sales, data$visit_lag1)
```

```
## [1] 0.3813843
```

We can also do the same for yesterday's sales. Here we are comparing today's milk tea sales to yesterday's milk tea sales and today's green tea sales to yesterday's green tea sales. In this case, we also observe a positive relationship similar to visit_lag1.

```r
ggplot(data, aes(x = sales_lag1, y = sales, color = tea)) +
  geom_point()
```



```
cor(data$sales, data$sales_lag1)
```

```
## [1] 0.4154355
```

Based on our initial exploration, a Poisson regression model seems like a good starting point.

The Poisson model is uesd for modeling non-negative counts and is especially useful in the case where we have right-skewed data that can't be effectively modeled by a Normal model.

Using all of the predictors we described above, the structure of the model is as such:

The relationship between today's (day i) tea sales (Y_i) and yesterday's customers (X_{i1}), yesterday's sales (X_{i2}), whether today is a weekend/holiday (X_{i3}), and whether we are looking at green tea or milk tea (X_{i4}) can be written as

$$Y_i|\beta_0, \beta_1, \beta_2, \beta_3, \beta_4 \overset{ind}{\sim} Pois(\lambda_i)$$

with

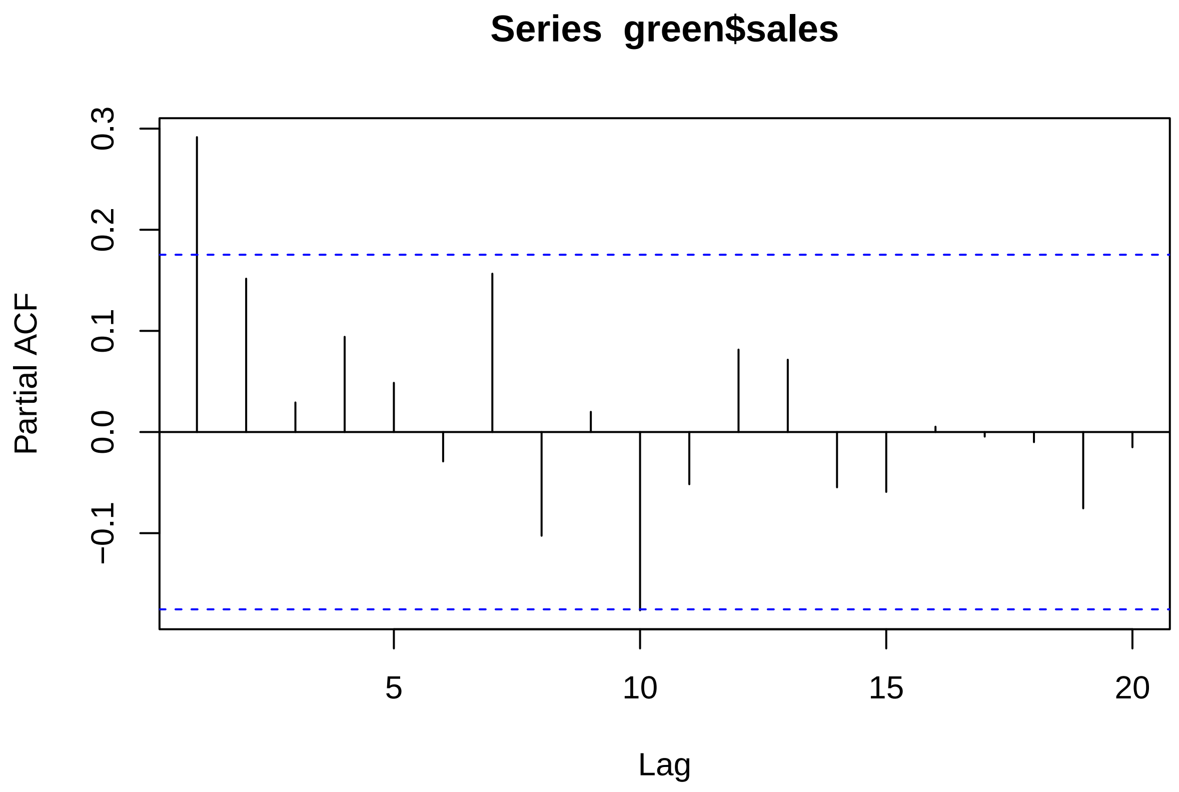
$$\log(\lambda_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \beta_4 X_{i4}$$

where λ_i is the average or mean sales for today.

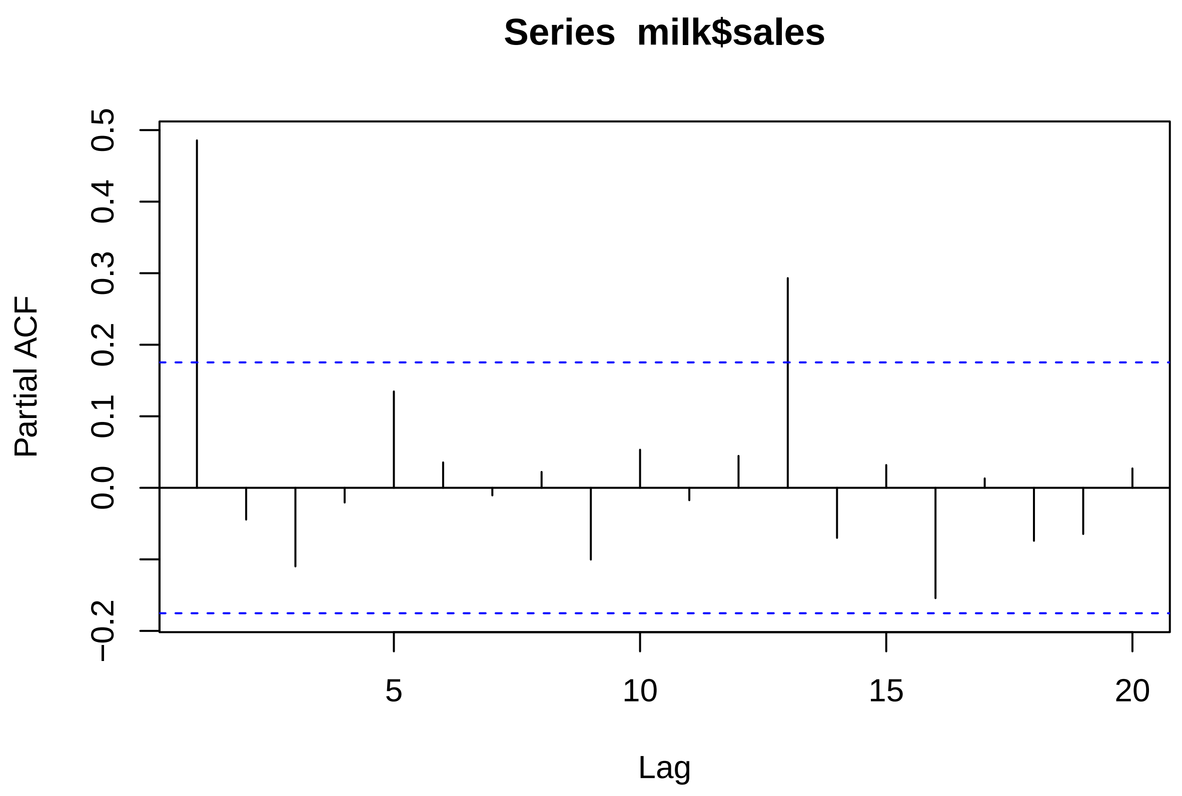
Before we proceed, let's describe some model assumptions.

The Poisson model assumes that conditioned on the predictors X, the observed sales Y_i for day i is indepedent of the observed sales Y_j for any other day j. In our structure, we wrote that the average sales for today depended on the observed sales from yesterday. Thus our model assumes that accounting for this dependence and holding the other predictors constant, the tea sales for day i and day j are indepedent. We can partially assess whether this assumption has merit by looking at the partial autocorrelation (PACF) plot for tea sales. The PACF tells us whether sales are correlated to sales at lag times after accounting for the correlation at earlier lags. If the condition holds, then we should see a drop and no significant autocorrelation in the PACF plot at lag 2, continuing for all future lags. This means that after accounting for the autocorrelation at lag 1, tea sales are not correlated with each other.

```r
pacf(green$sales)
```



```r
pacf(milk$sales)
```



The PACF for green tea meets our expectation. The PACF for milk tea has a significant spike at lag 13, but since the time is somewhat random and the autocorrelation is not so large, this could be due to random chance. We conclude that this first condition is reasonably fulfilled.

Another condition for the Poisson model regards the variability in Y. Conditioned on predictors X, the expected tea sales $E(Y|X)$ is roughly equal to the variance in tea sales $V(Y|X)$. This is a bit hard to test when we have multiple predictors, but we can look at specific predictors to see whether average sales is roughly equal to the variance in sales across the same level of a certain predictor.

```
data %>%
  group_by(cut(sales_lag1, 5), tea) %>%
  summarize(mean = mean(sales), var=var(sales))
```

```
## ‘summarise()‘ has grouped output by ’cut(sales_lag1, 5)’. You can override
## using the ‘.groups‘ argument.
```

```
## # A tibble: 8 x 4
## # Groups:   cut(sales_lag1, 5) [5]
##   ‘cut(sales_lag1, 5)‘ tea    mean   var
##   <fct>                <chr> <dbl> <dbl>
## 1 (-0.033,6.6]         green  4.21  9.09
## 2 (-0.033,6.6]         milk   5.10 13.2
## 3 (6.6,13.2]           green  7.59 31.4
## 4 (6.6,13.2]           milk   7.44 33.7
## 5 (13.2,19.8]          green  6     8
## 6 (13.2,19.8]          milk   6    11.5
## 7 (19.8,26.4]          green  6.5   0.5
## 8 (26.4,33]            milk  22    50
```

```
data %>%
  group_by(cut(sales_lag1, 5), weekend_holiday) %>%
  summarize(mean = mean(sales), var=var(sales))
```

```
## `summarise()` has grouped output by 'cut(sales_lag1, 5)'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 9 x 4
## # Groups:   cut(sales_lag1, 5) [5]
##   `cut(sales_lag1, 5)` weekend_holiday  mean    var
##   <fct>                <fct>           <dbl>  <dbl>
## 1 (-0.033,6.6]         0                3.81  7.26
## 2 (-0.033,6.6]         1                7.17 14.8
## 3 (6.6,13.2]           0                5.35 11.9
## 4 (6.6,13.2]           1                9.30 42.9
## 5 (13.2,19.8]          0                7    14.7
## 6 (13.2,19.8]          1                4.67  0.333
## 7 (19.8,26.4]          0                6.5   0.5
## 8 (26.4,33]            0               27    NA
## 9 (26.4,33]            1               17    NA
```

Based on these findings, the application of this condition is questionable as the variance is much higher than the mean in many cases. When the observed variability in Y is greater than the expected variability in Y by the assumed probability model (in this case, Poisson), then we say that Y is overdispersed. In the case where Y is overdispersed in the context of the Poisson model, a better alternative is the negative binomial model, which relaxes the condition that $E(Y|X) = V(Y|X)$. Still, we can continue with the Poisson model for now and visit the negative binomial model later.

Recall that our model assumes that Y and X are related by the equation

$$\log(\lambda_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \beta_4 X_{i4}$$

where λ_i is the expected sales for day i.

In a Bayesian framework, before our model is fit with our data, we need to supply our prior beliefs about the relationship between X and Y. Then, after our model is fit with the data, our posterior model will be a balance between our prior beliefs and the observed values. This is especially useful if we have strong prior beliefs that we want incorporated in our final model and to prevent overfitting to the observed data.

Let us define priors for some of our parameters. Our prior belief is that the average daily number of tea sales is between 3 and 20. Since $\exp(1) \approx 3$ and $\exp(3) \approx 20$, we can define the prior centered intercept

$$\beta_{0c} \sim N(2, 0.5^2)$$

It is often hard to predict the effects of changes in one predictor when there are multiple other predictors present. To keep things simple, we can use default normal priors for the other coefficients.

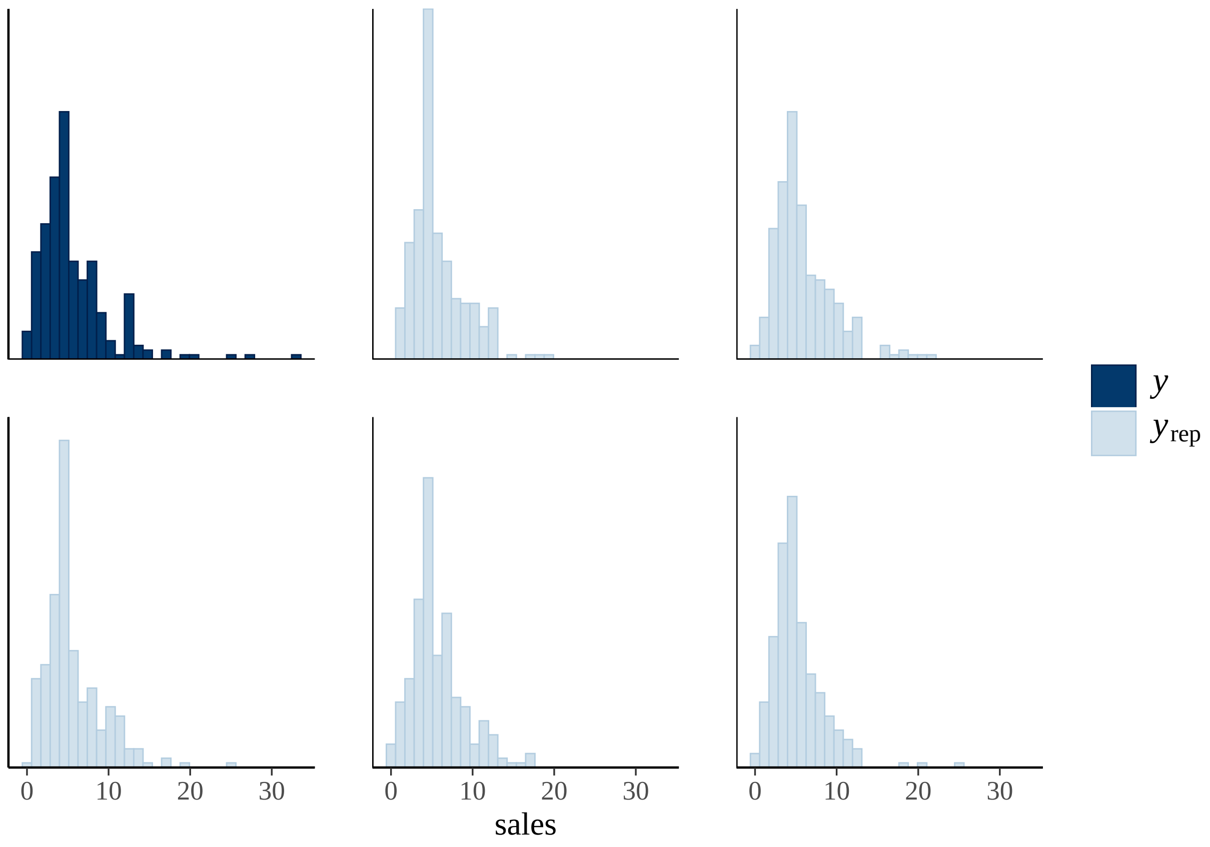
$$\beta_k \sim N(0, s_k^2), k \in 1, 2, 3, 4$$

```
pois_model1 <- stan_glm(sales ~ visit_lag1 + sales_lag1 + weekend_holiday + tea,
                        data = data,
                        family = poisson,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0)
```

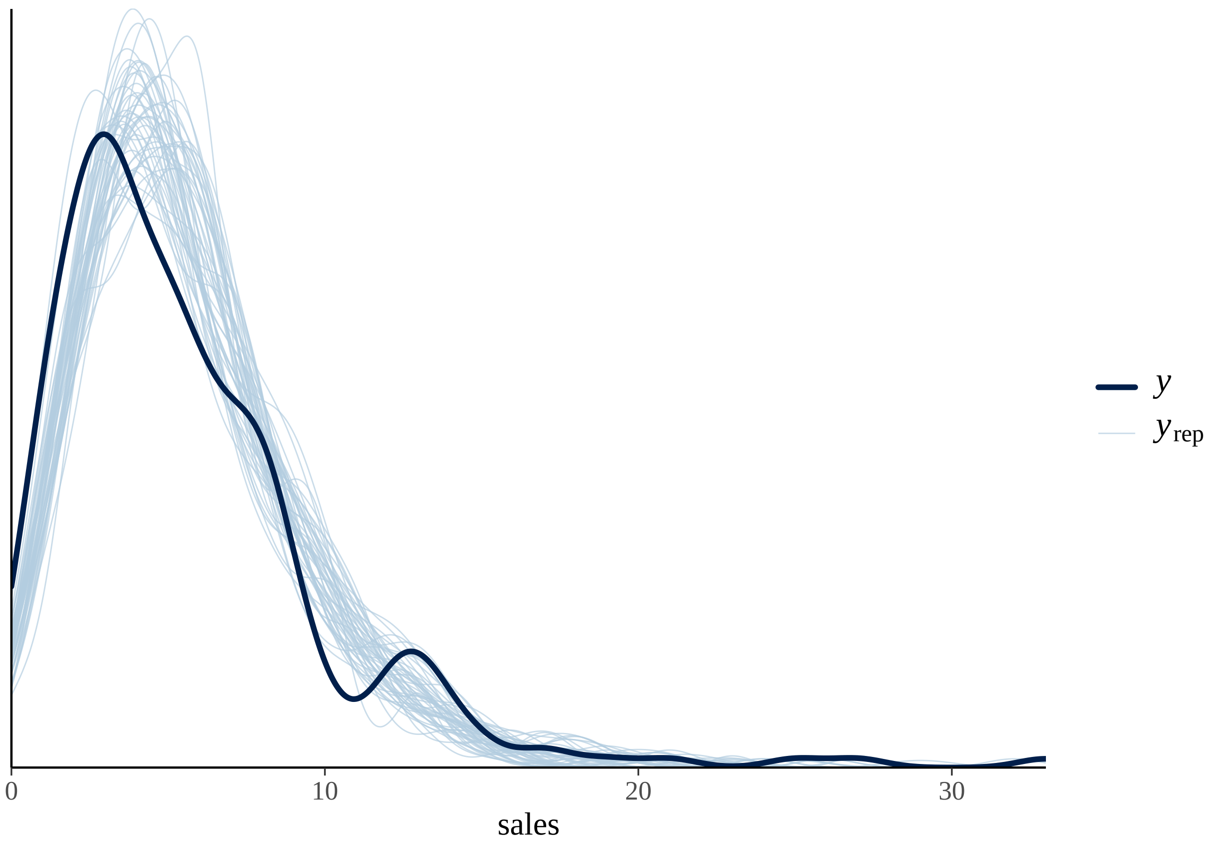
The post posterior check allows us to compare 50 of our simulated datasets with the actual data. A good model will produce distributions that look like the actual data.

```
set.seed(1)
pp_check(pois_model1, plotfun = "hist", nreps = 5) +
  xlab("sales")
```

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



```
pp_check(pois_model1) +
  xlab("sales")
```



From the post posterior check, our model produces results that are reasonable. It seems like our model overestimates the number of days with middling sales (around 5) and underestimates the number of days with lower sales.

The Poisson model is not bad, but recall that our data is overdispersed. To address this problem, we can fit a negative binomial model and see if it produces comparatively better results.

For a negative binomial model, the relationship between today's (day i) tea sales (Y_i) and yesterday's customers (X_{i1}), yesterday's sales (X_{i2}), whether today is a weekend/holiday (X_{i3}), and whether we are looking at green tea or milk tea (X_{i4}) can be written as

$$Y_i|\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, r \overset{ind}{\sim} NegBin(\mu_i, r)$$

with

$$\log(\mu_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \beta_4 X_{i4}$$

where μ_i is the expected sales for day i.

Recall that a negative binomial random variable $Y|\mu, r$ has expected value $E(Y|\mu, r) = \mu$ and variance $V(Y|\mu, r) = \mu + \frac{\mu^2}{r}$.

The r is a reciprocal dispersion parameter that allows the relaxation of the equal means and variance condition from Poisson regression. For large r, $\frac{\mu^2}{r} \to 0$ and $E(Y) \approx V(Y)$, and the negative binomial model will produce similar results to the Poisson model. For small r, $V(Y) > E(Y)$, so the data is overdispersed when considering the Poisson model so the negative binomial model will produce a better fit.

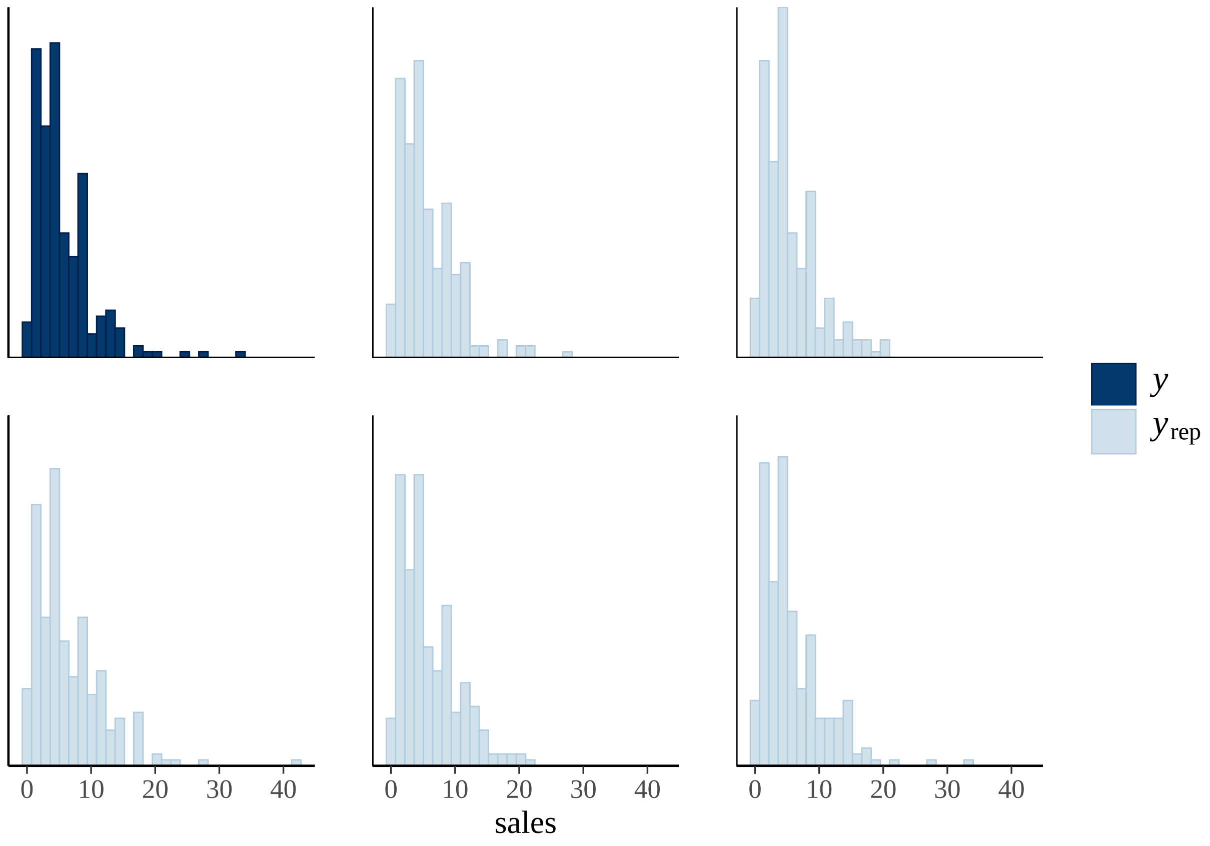
Since we introduced a new parameter r, we need to specify a prior for r. Since we have the restriction that $r > 0$, we can use a default Exponential prior.

$$r \sim Exp(1)$$

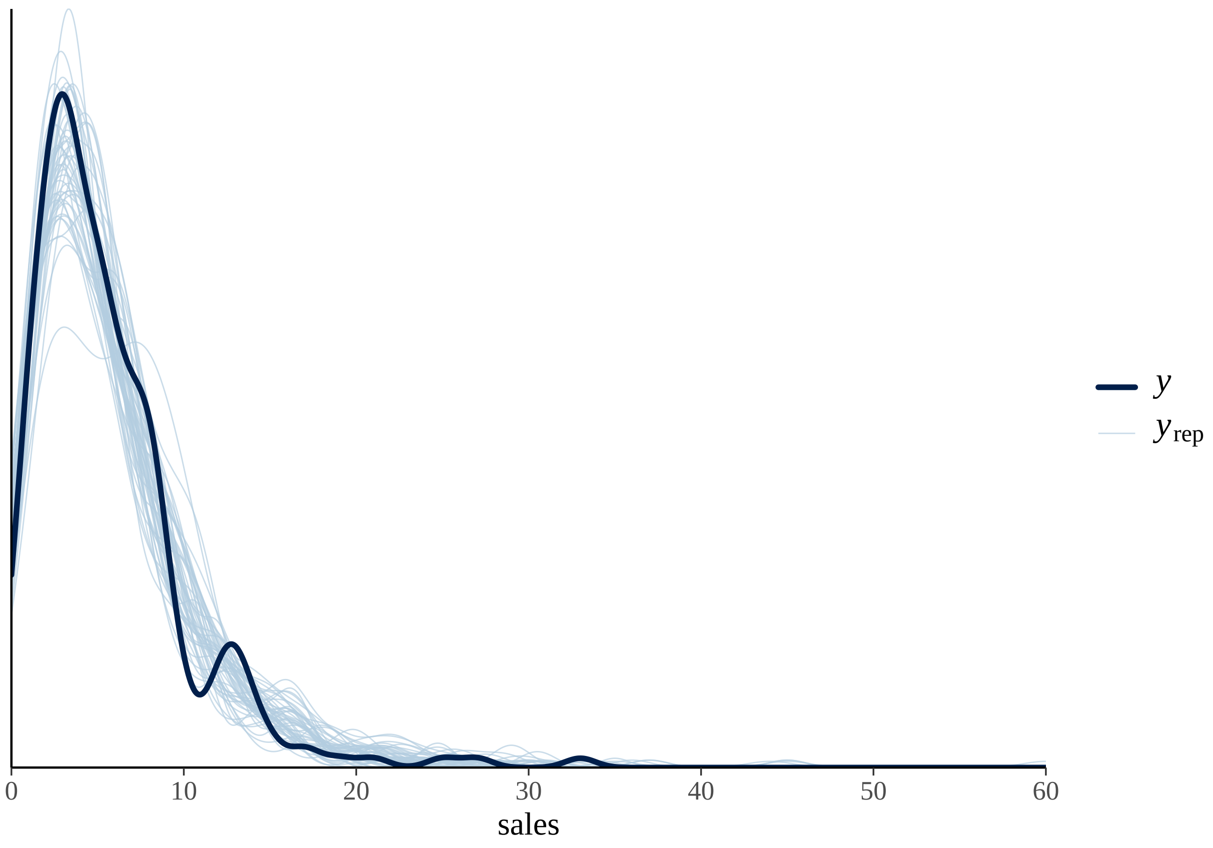
```r
negb_model1 <- stan_glm(sales ~ visit_lag1 + sales_lag1 + weekend_holiday + tea,
                        data = data,
                        family = neg_binomial_2,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        prior_aux = exponential(1, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0
                        )
```

```r
set.seed(1)
pp_check(negb_model1, plotfun = "hist", nreps = 5) +
  xlab("sales")
```

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



```r
pp_check(negb_model1) +
  xlab("sales")
```



The simulated datasets from the negative binomial model more closely match the observed data than the simulated datasets from the Poisson model do. It looks like the problems of overestimating and underestimating we discussed previously are less pronounced.

For a more objective comparison between these models, we can compare their expected log-predictive densities (ELPD). ELPD is a measure of how well a model predicts new data, with larger ELPD indicating better predictive performance.

```
set.seed(84735)
pois_loo_1 <- loo(pois_model1)
```

```
## Warning: Found 1 observation(s) with a pareto_k > 0.7. We recommend calling 'loo' again with argumen
```

```
negb_loo_1 <- loo(negb_model1)

c(pois_loo_1$estimates[1], negb_loo_1$estimates[1])
```

```
## [1] -690.2913 -633.6535
```

```
loo_compare(pois_loo_1, negb_loo_1)
```

```
##             elpd_diff se_diff
## negb_model1   0.0       0.0
## pois_model1 -56.6      18.2
```

Our results indicate that the negative binomial model is a better predictive model than the Poisson model.

Before we move on, we can try to fit different models using different predictors to hopefully improve our model.

```r
negb_model2 <- stan_glm(sales ~ sales_lag1 + weekend_holiday + tea,
                        data = data,
                        family = neg_binomial_2,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        prior_aux = exponential(1, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0
                        )
```

```r
negb_model3 <- stan_glm(sales ~ sales_lag1*tea + weekend_holiday,
                        data = data,
                        family = neg_binomial_2,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        prior_aux = exponential(1, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0
                        )
```

```r
negb_model4 <- stan_glm(sales ~ sales_lag1 + weekend_holiday*tea,
                        data = data,
                        family = neg_binomial_2,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        prior_aux = exponential(1, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0
                        )
```

```r
set.seed(84735)

negb_loo_2 <- loo(negb_model2)
negb_loo_3 <- loo(negb_model3)
negb_loo_4 <- loo(negb_model4)

c(negb_loo_1$estimates[1], negb_loo_2$estimates[1],
  negb_loo_3$estimates[1], negb_loo_4$estimates[1])
```

```
## [1] -633.6535 -633.6703 -634.5904 -634.6912
```

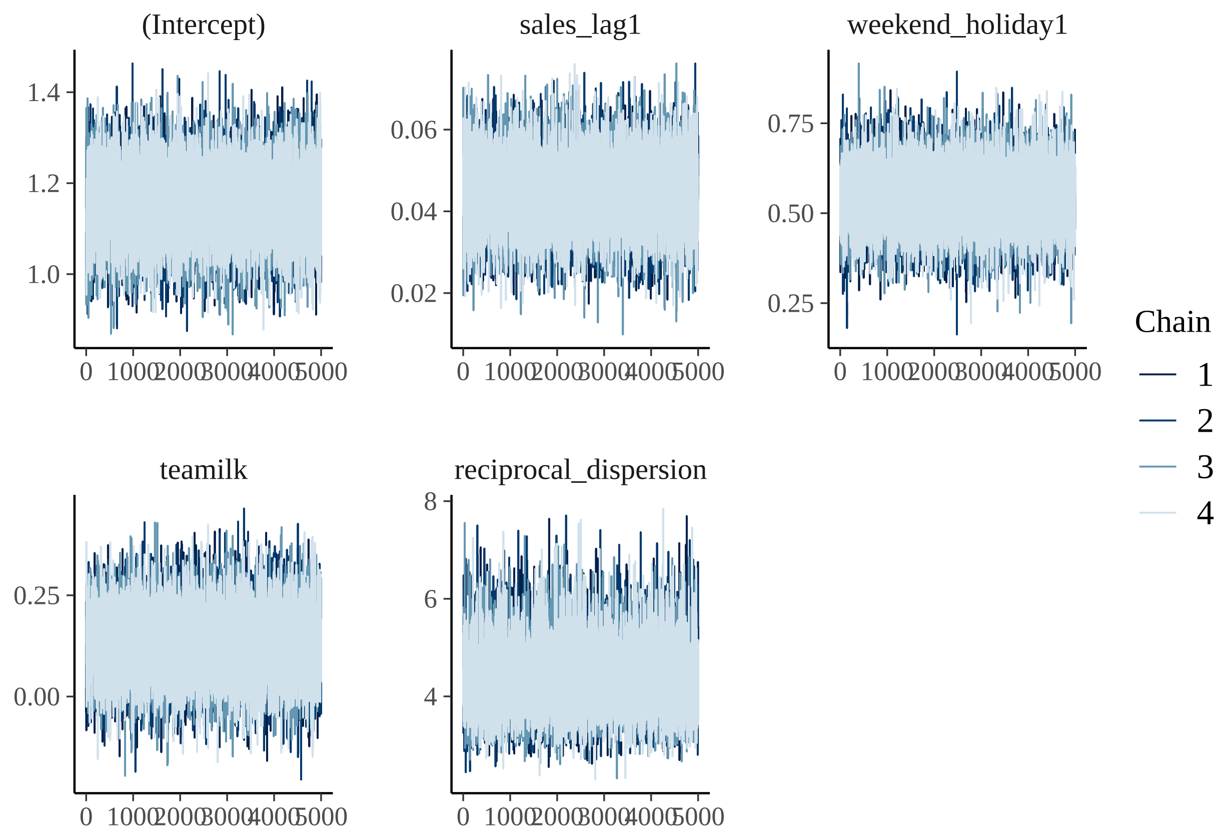
```r
loo_compare(negb_loo_1, negb_loo_2, negb_loo_3, negb_loo_4)
```

```
##             elpd_diff se_diff
## negb_model1  0.0       0.0
## negb_model2  0.0       1.4
## negb_model3 -0.9       1.5
## negb_model4 -1.0       1.5
```

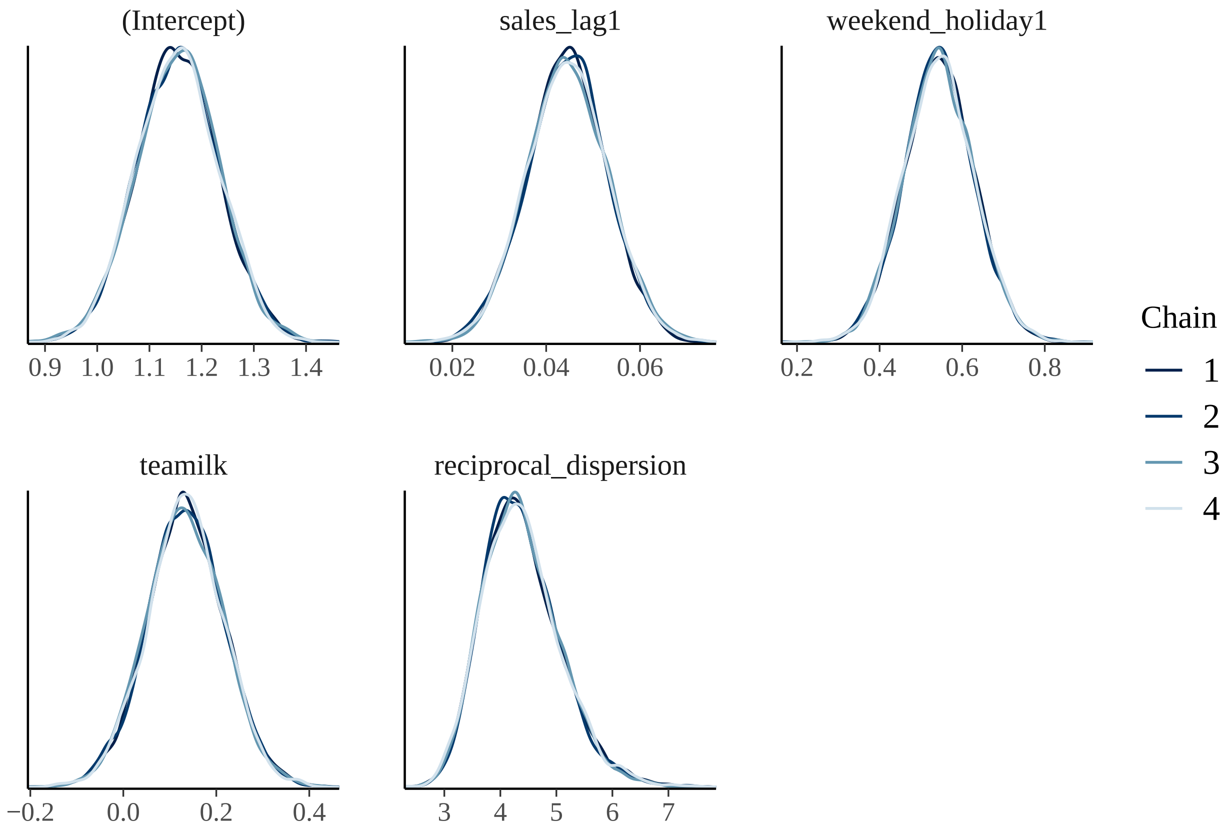
The ELPD indicates that there is not a significant difference in predictive power across any of the four models. For the sake of simplicity, we can choose to proceed with the simplest model with the fewest predictors, negb_model2.

Before we continue, let's check diagnostic plots to ensure our simulations are stable.

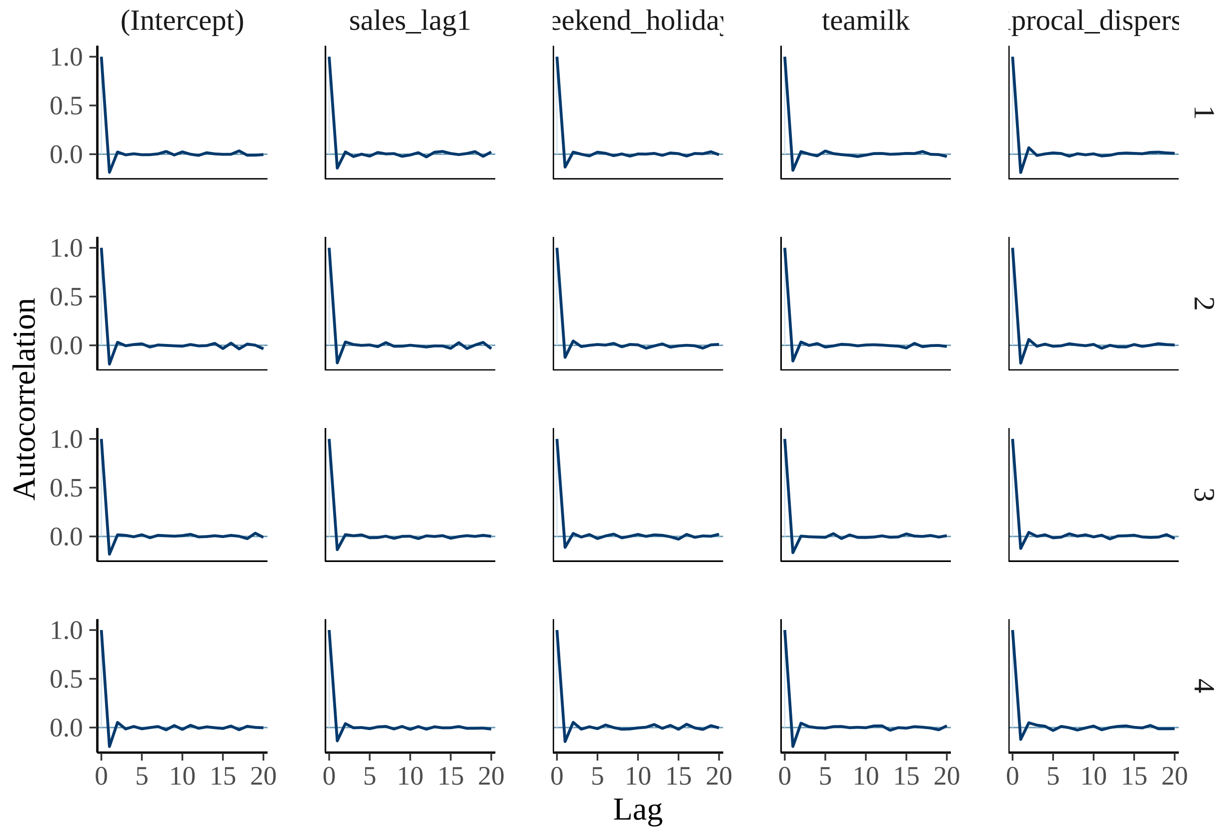
```
mcmc_trace(negb_model2)
```



```
mcmc_dens_overlay(negb_model2)
```



```
mcmc_acf(negb_model2)
```



The chains look random with no long-term trend or flat areas where the algorithm gets stuck for long periods of time, the four chains produce similar distributions for posterior plausible values of the intercept, coefficient, and reciprocol dispersion parameters, and the autocorrelations drop quickly. These all suggest that our simulation is stable.

In negb_model2, we have 3 predictors: yesterday's sales (X_{i1}), whether today is a weekend/holiday (X_{i2}), and whether we are looking at green tea or milk tea (X_{i3}).

The relationship between today's (day i) tea sales (Y_i) and $X = (X_{i1}, X_{i2}, X_{i3})$ can be written as

$$Y_i|\beta_0, \beta_1, \beta_2, \beta_3, r \overset{ind}{\sim} NegBin(\mu_i, r)$$

with

$$\log(\mu_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3}$$

where μ_i is the expected sales for day i.

```
tidy(negb_model2, conf.int = TRUE, conf.level = 0.80)
```

```
## # A tibble: 4 x 5
##   term             estimate std.error conf.low conf.high
##   <chr>               <dbl>     <dbl>    <dbl>     <dbl>
## 1 (Intercept)        1.15      0.0789    1.05      1.26
## 2 sales_lag1         0.0445    0.00852   0.0335    0.0556
## 3 weekend_holiday1   0.543     0.0849    0.434     0.653
## 4 teamilk            0.133     0.0818    0.0271    0.240
```

```r
draws <- as_draws_df(negb_model2)

quantile(draws$reciprocal_dispersion, probs = c(0.10, 0.5, 0.90))
```

```
##      10%      50%      90% 
## 3.548551 4.316868 5.295913
```

The posterior median relationship is

$$\log(\mu_i) = 1.155 + .045X_{i1} + 0.543X_{i2} + 0.133X_{i3}$$

The baseline for this model is green tea on a weekday. Thus, the intercept tells us for a weekday with no green tea sales yesterday, the expected green tea sales today is $\exp(1.155) = 3.174$.

We can also interpret the 80% credible intervals.

```r
exp(posterior_interval(negb_model2, prob = 0.8))
```

```
##                             10%        90%
## (Intercept)            2.867133   3.511718
## sales_lag1             1.034096   1.057155
## weekend_holiday1       1.543315   1.922105
## teamilk                1.027519   1.270838
## reciprocal_dispersion 34.762899 199.519662
```

There is an 80% posterior probability that the number of green tea sales on a weekday with no green tea sales yesterday is between 2.867 and 3.512.

There is an 80% posterior probability that when controlling for weekend/holiday status and type of tea, for every additional tea sale yesterday, the number of tea sales increases by between 1.034 and 1.057 times.

There is an 80% posterior probability that when controlling for yesterday's sales and type of tea, if today is a weekend/holiday, the number of tea sales increases by between 1.543 and 1.922 times.

There is an 80% posterior probability that when controlling for yesterday's sales and weekend/holiday status, the number of milk tea sales is between 1.028 and 1.271 times the number the number of green tea sales.

Now, we are ready to use our model to make predictions. The last date in our dataset is Jan 4 2026. We can use our model to predict the tea sales for the next day, Jan 5 2026.

On Jan 4 2026, there were 7 green tea sales and 14 milk tea sales. We also know that Jan 5 2026 is a regular weekday. Based on the posterior median relationship, the predicted mean number of green tea and milk tea sales tomorrow are

$$\mu_g = \exp(1.155 + .045(7) + 0.543(0) + 0.133(0)) = 4.349$$
$$\mu_m = \exp(1.155 + .045(14) + 0.543(0) + 0.133(1)) = 6.653$$

These are only the expected sales for tomorrow – the range of possible sales tomorrow is much larger. For a single possibility for tomorrow's tea sales, we can use the posterior median estimate for $r = 4.317$ and draw from a negative binomial distribution.

```r
set.seed(123)
rnbinom(n=1, size=4.317, mu=4.349)
```

```
## [1] 5
```

```r
rnbinom(n=1, size=4.317, mu=6.653)
```

```
## [1] 11
```

The model proposes that it is possible that we will make 5 green tea sales and 11 milk tea sales tomorrow. These are just possible values from the posterior median model. Since we had 4 chains with each chain producing 5000 posterior plausible relationships (5000 models), we can repeat this procedure for all 20000 simulated models. This gives us a better understanding of the range of posterior plausible sales and allows us to answer questions like "What is the probability that I make between such and such sales tomorrow?"

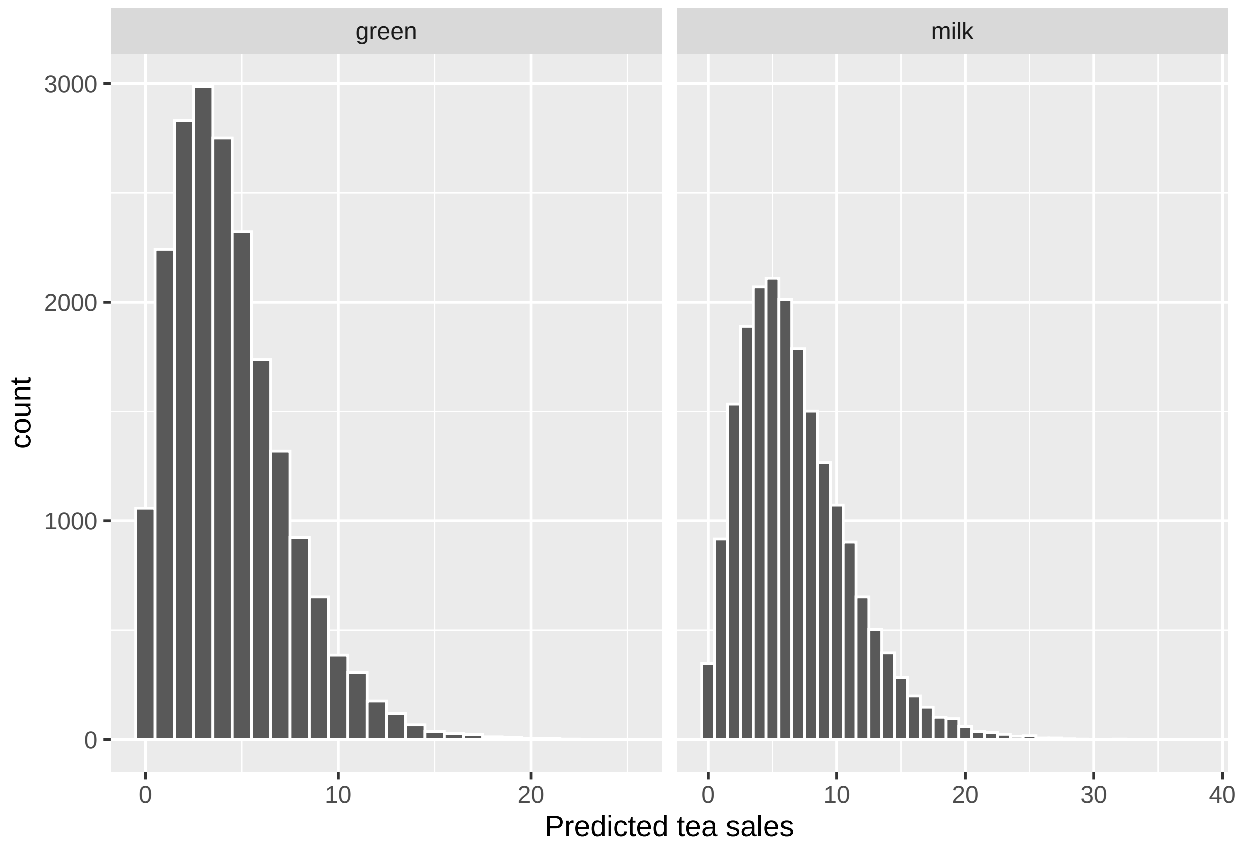
First, we need to generate multiple possibilities for tomorrow's tea sales. From our simulation, we produced 4 chains of 5000 tuples each of $(\beta_0, \beta_1, \beta_2, \beta_3, r)$. Using these values and plugging in the information needed for tomorrow's prediction, we end up with 20000 estimates each for μ_g and μ_m. Then, for each tea, we generate 20000 possibilities for tomorrow's sales from a negative binomial distribution from the 20000 μ estimates and the 20000 r estimates. This process is compactly conducted by the posterior_predict function.

```r
# Calculate posterior predictions
set.seed(84735)
g_prediction <- posterior_predict(
  negb_model2, newdata = data.frame(sales_lag1 = 7,
                                    weekend_holiday = as.factor(0),
                                    tea = 'green'))
m_prediction <- posterior_predict(
  negb_model2, newdata = data.frame(sales_lag1 = 14,
                                    weekend_holiday = as.factor(0),
                                    tea = 'milk'))
```

Now we plot the predictions. This shows us the distribution and range of posterior plausible values.

```r
pred_df <- tibble(
  value = c(g_prediction, m_prediction),
  tea   = rep(c("green", "milk"),
              times = c(length(g_prediction), length(m_prediction)))
)

ggplot(pred_df, aes(x = value)) +
  geom_histogram(binwidth = 1, color = "white") +
  facet_wrap(~ tea, scales = "free_x") +
  xlab("Predicted tea sales")
```



```r
quantile(g_prediction, probs = c(0.10, 0.5, 0.90))
```

```
## 10% 50% 90%
##   1   4   8
```

```r
quantile(m_prediction, probs = c(0.10, 0.5, 0.90))
```

```
## 10% 50% 90%
##   2   6  12
```

There is an 80% posterior probability that we will make between 1 and 8 green tea sales tomorrow. There is an 80% posterior probability that we will make between 2 and 12 milk tea sales tomorrow.

Now, from sales data not included in the dataset, the actual sales on January 5 2026 were 5 green tea sales and 5 milk tea sales. These counts are well within our prediction intervals.

In fact, we can test our model against many more known observations to evaluate its accuracy.

```r
prediction_summary(model = negb_model2, data = data)
```

```
##        mae mae_scaled within_50 within_95
## 1 2.153325  0.6419407     0.604     0.972
```

Here, we are only testing our model on observations in the training set. On average, our predictions stray away from actual tea sales values by about 2.153 sales or 0.642 standard deviations. In the context of our problem, this error isn't perfectly accurate nor is it too large. This suggests that our model provides reasonable estimates for tea sales, though we have to keep in mind that this measure only accounted for seen data. Our intervals are also good, with 60.4% of observations lying within their 50% posterior prediction interval and 97.2% of observations lying within their 95% interval.

The limitation with our previous procedure was that we tested our model on data that it was trained on. When making future predictions, we want a model that does well on unseen data. In this case, we can estimate the error for new observations by training on the first k days then predicting and calculating the error on day k+1. Then, we can train on the first k+1 rows and so on. We expect the accuracy to be lower (error to be higher), but if the model is a good predictive model, the change should not be too drastic. Below we test on the last week of data.

```r
train_daily <- data %>%
  filter(date < "2025-12-29")

test_daily <- data %>%
  filter(date >= "2025-12-29")
```

```r
forward_chaining_folds <- function(df, time_var) {
  times <- sort(unique(df[[time_var]]))
  folds <- list()

  for (i in 2:length(times)) {
    train_time <- times[1:(i-1)]
    test_time <- times[i]

    train_idx <- which(df[[time_var]] %in% train_time)
    test_idx <- which(df[[time_var]] == test_time)

    folds[[i-1]] <- list(train = train_idx, test = test_idx)
  }

  return(folds)
}

folds <- forward_chaining_folds(test_daily, "date")
```

```r
set.seed(123)
results <- list()

for (i in seq_along(folds)) {
  fold <- folds[[i]]
  train_data <- bind_rows(train_daily, test_daily[fold$train, ])
  test_data <- test_daily[fold$test, ]

  model <- stan_glm(sales ~ sales_lag1 + weekend_holiday + tea,
                        data = train_data,
                        family = neg_binomial_2,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        prior_aux = exponential(1, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0
                        )
  preds <- posterior_predict(model, newdata = test_data)

  # for each column calculate the mean across all posterior draws
  test_data$pred <- apply(preds, 2, mean)
  qs = apply(preds, 2, quantile, probs = c(0.025, 0.25, 0.5, 0.75, 0.975))
  qs_df <- as.data.frame(t(qs))
  test_data = cbind(test_data, qs_df)

  results[[i]] <- test_data
}

cv_results <- do.call(rbind, results)
cv_results <- cv_results %>%
  rename(q25 = "25%",
         q75 = "75%",
         q2_5 = "2.5%",
         q97_5 = "97.5%")
```

```r
mean(abs(cv_results$sales - cv_results$pred))
```

```
## [1] 3.375283
```

```r
mean(cv_results$sales >= cv_results$q25 & cv_results$sales <= cv_results$q75)
```

```
## [1] 0.5833333
```

```r
mean(cv_results$sales >= cv_results$q2_5 & cv_results$sales <= cv_results$q97_5)
```

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

On average, for unseen data, our predictions stray away from actual tea sales values by about 3.375 sales. We also see that 58.3% of the observations fell within their 50% prediction interval, and 100% of the observations fell within their 95% prediction interval. The model did worse at predicting new tea sales, but it still gives us a general idea of what tomorrow's sales might be.

Finally, let's see if our model can actually help us meet customer demands and reduce spoilage.

We will focus on green tea since the preparation for all types of green teas are similar.

At a baseline, we make enough green tea for 6 sales at the start of each day. If another green tea order comes in after 6 sales have been made, then we restock (which can take 15 minutes) and make enough green tea for 6 more sales.

Using the model, we will make enough green tea for the predicted sales at the start of each day. If another order comes in after our green tea is depleted, then we will similarly restock with enough to make 6 sales.

We will keep track of the amount of wasted green tea (in sales) and the number of restocking events at the end of each day and compare our baseline procedure to using the model.

For simplicity, let's train the model up until the last two weeks of data and make predictions on the last two weeks. If we continuously updated our model during these last weeks instead, we may get better results.

```r
compare_train <- data %>%
  filter(date < "2025-12-22")

compare_test <- data %>%
  filter(date >= "2025-12-22")

compare_model <- stan_glm(sales ~ sales_lag1 + weekend_holiday + tea,
                        data = compare_train,
                        family = neg_binomial_2,
                        prior_intercept = normal(2, 0.5),
                        prior = normal(0, 2.5, autoscale=TRUE),
                        prior_aux = exponential(1, autoscale=TRUE),
                        chains = 4, iter = 5000*2, seed = 84735,
                        refresh = 0
                        )
```

```r
green_test <- compare_test %>%
  filter(tea=="green")
compare_preds <- posterior_predict(compare_model, newdata = green_test)

green_preds <- apply(compare_preds, 2, mean)
green_preds <- data.frame(day = names(green_preds), pred = green_preds)
green_preds$round_pred <- round(green_preds$pred)
green_preds$actual <- green_test$sales
green_preds$over <- pmax(green_preds$actual - green_preds$round_pred, 0)
green_preds$restock <- ceiling(green_preds$over / 6)
green_preds$total_made <- green_preds$round_pred + green_preds$restock * 6
green_preds$waste <- green_preds$total_made - green_preds$actual
```

```r
baseline <- data.frame(day = c(1:nrow(green_test)), pred = 6)
baseline$actual <- green_test$sales
baseline$over <- pmax(baseline$actual - baseline$pred, 0)
baseline$restock <- ceiling(baseline$over / 6)
baseline$total_made <- baseline$pred + baseline$restock * 6
baseline$waste <- baseline$total_made - baseline$actual
```

Now let's see how our model does.

```r
sum(baseline$restock)
```

```
## [1] 15
```

```r
sum(baseline$waste)
```

```
## [1] 45
```

```r
sum(green_preds$restock)
```

```
## [1] 12
```

```r
sum(green_preds$waste)
```

```
## [1] 28
```

We see that using our model resulted in both less frequent restocking and lower waste. We restocked 20% less often and reduced waste by around 37.78%.

```r
1- sum(green_preds$restock) / sum(baseline$restock)
```

```
## [1] 0.2
```

```r
1- sum(green_preds$waste) / sum(baseline$waste)
```

```
## [1] 0.3777778
```