

## Lecture 2

### Diagnostics and Model Evaluation

---

Colin Rundel

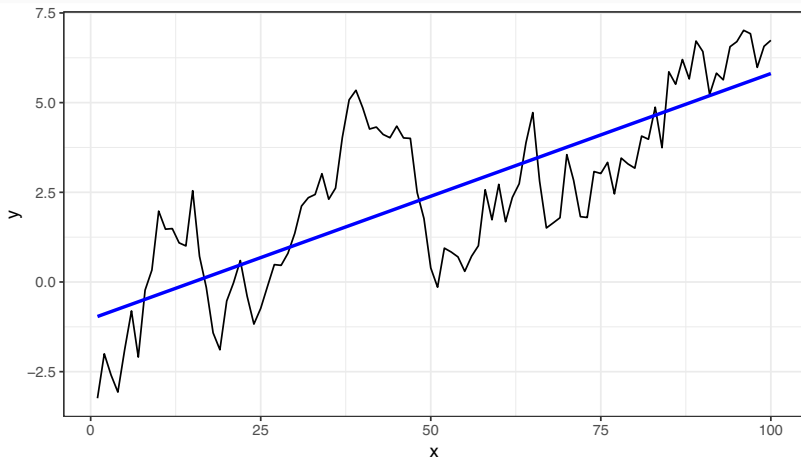
10/05/2018

## Some more linear models

---

# Linear model and data

```
ggplot(d, aes(x=x,y=y)) +  
  geom_line() +  
  geom_smooth(method="lm", color="blue", se = FALSE)
```



## Linear model

```
l = lm(y ~ x, data=d)
```

```
summary(l)
```

```
##
```

```
## Call:
```

```
## lm(formula = y ~ x, data = d)
```

```
##
```

```
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
```

```
## -2.6041 -1.2142 -0.1973  1.1969  3.7072
```

```
##
```

```
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) -1.030315   0.310326   -3.32  0.00126 **
```

```
## x              0.068409   0.005335   12.82 < 2e-16 ***
```

```
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
```

```
## Residual standard error: 1.54 on 98 degrees of freedom
```

```
## Multiple R-squared:  0.6266, Adjusted R-squared:  0.6227
```

```
## F-statistic: 164.4 on 1 and 98 DF, p-value: < 2.2e-16
```

## Bayesian model specification (JAGS)

```
model =  
"model{  
  # Likelihood  
  for(i in 1:length(y)){  
    y[i] ~ dnorm(mu[i], tau)  
    mu[i] = beta[1] + beta[2]*x[i]  
  }  
  
  # Prior for beta  
  for(j in 1:2){  
    beta[j] ~ dnorm(0,1/100)  
  }  
  
  # Prior for sigma / tau2  
  tau ~ dgamma(1, 1)  
  sigma2 = 1/tau  
}"
```

## Bayesian model fitting (JAGS)

```
n_burn = 1000; n_iter = 5000

m = rjags::jags.model(
  textConnection(model), data=d,
  quiet=TRUE, n.chains = 4
)
update(m, n.iter=n_burn, progress.bar="none")
samp = rjags::coda.samples(
  m, variable.names=c("beta", "sigma2"),
  n.iter=n_iter, progress.bar="none"
)

str(samp, max.level=1)
## List of 4
## $ : 'mcmc' num [1:5000, 1:3] -0.747 -0.677 -0.911 -0.906 -0.794 ...
## .. attr(*, "dimnames")=List of 2
## .. attr(*, "mcpars")= num [1:3] 1001 6000 1
## $ : 'mcmc' num [1:5000, 1:3] -1.57 -1.55 -1.55 -1.69 -1.32 ...
## .. attr(*, "dimnames")=List of 2
## .. attr(*, "mcpars")= num [1:3] 1001 6000 1
## $ : 'mcmc' num [1:5000, 1:3] -0.981 -1.087 -0.768 -0.831 -0.907 ...
## .. attr(*, "dimnames")=List of 2
## .. attr(*, "mcpars")= num [1:3] 1001 6000 1
## $ : 'mcmc' num [1:5000, 1:3] -0.67 -0.854 -0.959 -1.169 -1.375 ...
## .. attr(*, "dimnames")=List of 2
## .. attr(*, "mcpars")= num [1:3] 1001 6000 1
## - attr(*, "class")= chr "mcmc.list"
```

```
df_mcmc = tidybayes::gather_draws(samp, beta[i], sigma2) %>%
  mutate(parameter = paste0(.variable, ifelse(is.na(i), "", paste0("[", i, "]")))) %>%
  group_by(parameter, .chain)
```

```
head(df_mcmc)
```

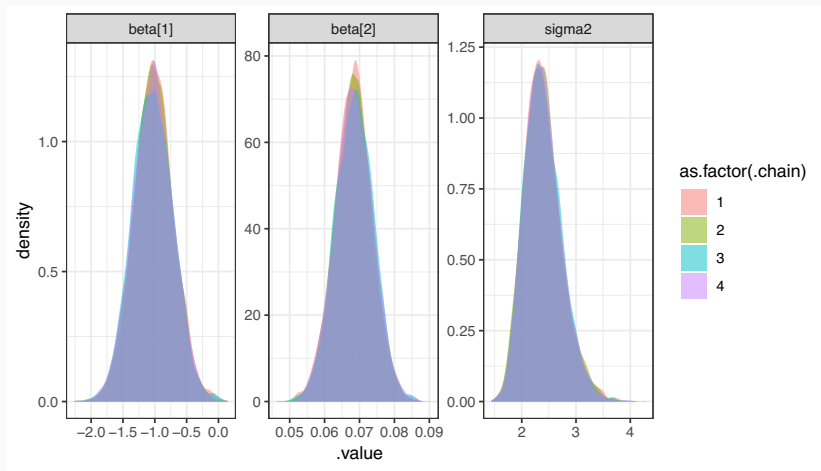
```
## # A tibble: 6 x 7
## # Groups:   parameter, .chain [2]
##   .chain .iteration .draw   i .variable .value parameter
##   <int>     <int> <int> <int> <chr>     <dbl> <chr>
## 1       1         1     1     1  beta     -0.747 beta[1]
## 2       1         1     1     2  beta      0.0646 beta[2]
## 3       1         2     2     1  beta     -0.677 beta[1]
## 4       1         2     2     2  beta      0.0581 beta[2]
## 5       1         3     3     1  beta     -0.911 beta[1]
## 6       1         3     3     2  beta      0.0625 beta[2]
```

```
tail(df_mcmc)
```

```
## # A tibble: 6 x 7
## # Groups:   parameter, .chain [1]
##   .chain .iteration .draw   i .variable .value parameter
##   <int>     <int> <int> <int> <chr>     <dbl> <chr>
## 1       4       4995 19995  NA sigma2     1.69 sigma2
## 2       4       4996 19996  NA sigma2     2.43 sigma2
## 3       4       4997 19997  NA sigma2     2.20 sigma2
## 4       4       4998 19998  NA sigma2     2.65 sigma2
## 5       4       4999 19999  NA sigma2     1.81 sigma2
## 6       4       5000 20000  NA sigma2     2.43 sigma2
```

## Posterior plots

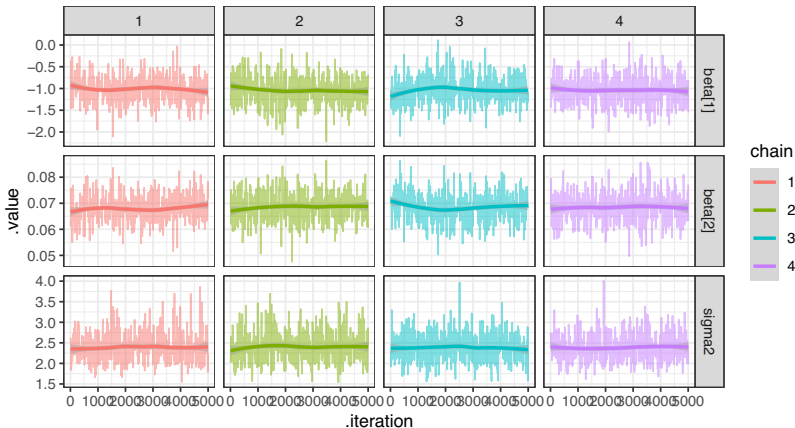
```
ggplot(df_mcmc, aes(fill=as.factor(.chain), group=.chain, x=.value)) +  
  geom_density(alpha=0.5, color=NA) +  
  facet_wrap(~parameter, scales = "free")
```





## Trace plots

```
df_mcmc %>% filter(.iteration %% 10 == 0) %>%  
ggplot(aes(x=.iteration, y=.value, color=as.factor(.chain))) +  
  geom_line(alpha=0.5) +  
  facet_grid(parameter~.chain, scale="free_y") +  
  geom_smooth(method="loess") + labs(color="chain")
```



# Credible Intervals

```
df_ci = tidybayes::mean_hdi(df_mcmc, .value, .width=c(0.8, 0.95))
```

```
df_ci
```

```
## # A tibble: 24 x 8
```

```
## # Groups:   parameter [3]
```

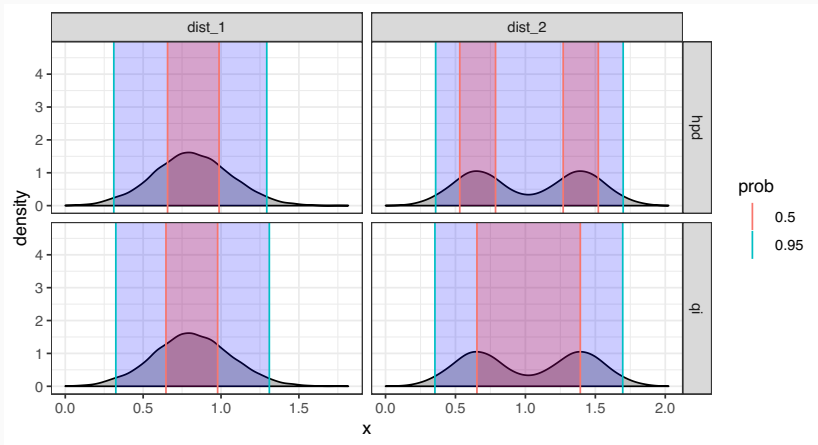
```
##   parameter .chain .value .lower .upper .width .point .interval
##   <chr>      <int>  <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
## 1 beta[1]      1 -1.01  -1.42  -0.635  0.8 mean  hdi
## 2 beta[1]      2 -1.03  -1.42  -0.627  0.8 mean  hdi
## 3 beta[1]      3 -1.03  -1.44  -0.622  0.8 mean  hdi
## 4 beta[1]      4 -1.03  -1.42  -0.606  0.8 mean  hdi
## 5 beta[2]      1  0.0680  0.0614  0.0748  0.8 mean  hdi
## 6 beta[2]      2  0.0683  0.0612  0.0748  0.8 mean  hdi
## 7 beta[2]      3  0.0685  0.0613  0.0751  0.8 mean  hdi
## 8 beta[2]      4  0.0684  0.0613  0.0752  0.8 mean  hdi
## 9 sigma2       1  2.39    1.91    2.78    0.8 mean  hdi
## 10 sigma2      2  2.39    1.92    2.78    0.8 mean  hdi
## # ... with 14 more rows
```

## Aside - mean\_qi vs mean\_hdi

These differ in the use of the quantile interval vs. the highest-density interval.

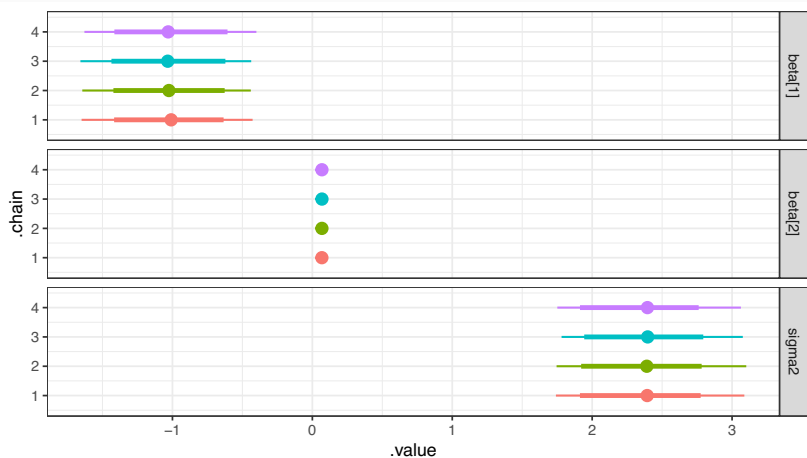
## Aside - mean\_qi vs mean\_hdi

These differ in the use of the quantile interval vs. the highest-density interval.



# Caterpillar Plots

```
df_ci %>%  
  ggplot(aes(x=.value, y=.chain, color=as.factor(.chain))) +  
  facet_grid(parameter~.) +  
  tidybayes::geom_pointintervalh() +  
  ylim(0.5,4.5)
```



## Prediction

---

## Revised model

```
model_pred =  
"model{  
  # Likelihood  
  for(i in 1:length(y)){  
    mu[i] = beta[1] + beta[2]*x[i]  
    y[i] ~ dnorm(mu[i], tau)  
    y_pred[i] ~ dnorm(mu[i], tau)  
  }  
  
  # Prior for beta  
  for(j in 1:2){  
    beta[j] ~ dnorm(0,1/100)  
  }  
  
  # Prior for sigma / tau2  
  tau ~ dgamma(1, 1)  
  sigma2 = 1/tau  
}"
```

## Revised fitting

```
n_burn = 1000; n_iter = 5000

m = rjags::jags.model(
  textConnection(model_pred), data=d,
  quiet=TRUE, n.chains = 1
)

update(m, n.iter=n_burn, progress.bar="none")

pred = rjags::coda.samples(
  m, variable.names=c("beta", "sigma2", "mu", "y_pred", "y", "x"),
  n.iter=n_iter, progress.bar="none"
)
```



## Predictions

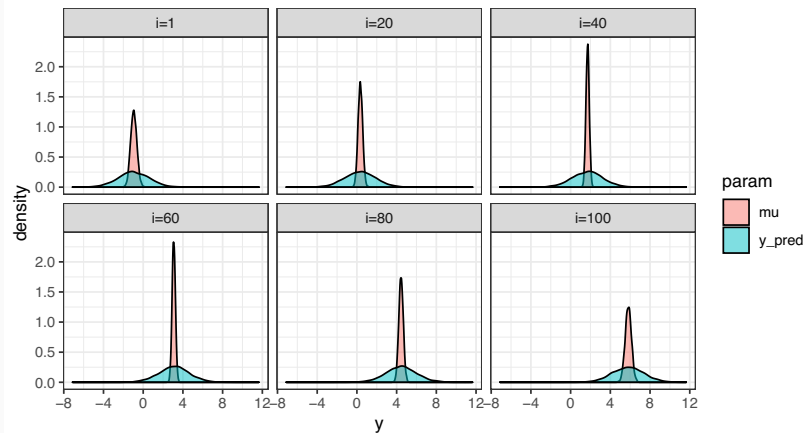
```
df_pred = tidybayes::spread_draws(pred, y_pred[i], y[i], x[i], mu[i]) %>%  
  mutate(resid = y - mu)
```

```
df_pred
```

```
## # A tibble: 500,000 x 9
```

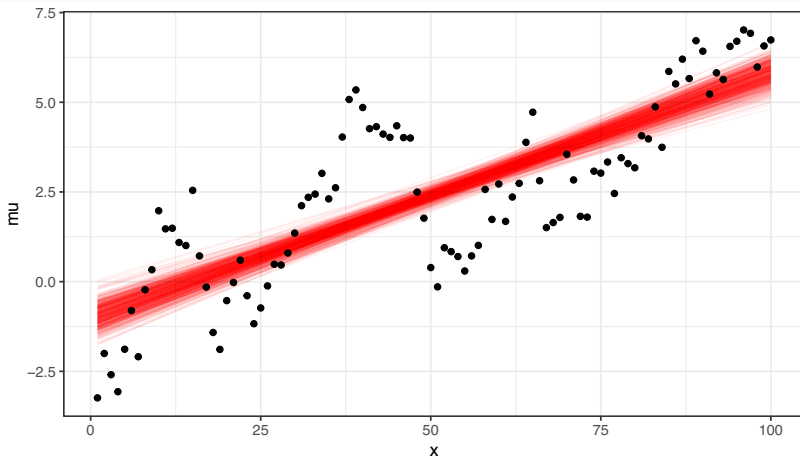
```
## # Groups:   i [100]
```

```
##   .chain .iteration .draw    i y_pred    y    x    mu resid  
##   <int>    <int> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1     1      1      1     1 -1.67  -3.24  1 -0.825 -2.42  
## 2     1      2      2     1  0.340  -3.24  1 -0.984 -2.26  
## 3     1      3      3     1 -0.554  -3.24  1 -0.800 -2.44  
## 4     1      4      4     1 -3.26  -3.24  1 -0.727 -2.51  
## 5     1      5      5     1  0.396  -3.24  1 -0.718 -2.52  
## 6     1      6      6     1  0.172  -3.24  1 -0.696 -2.54  
## 7     1      7      7     1 -1.24  -3.24  1 -0.797 -2.44  
## 8     1      8      8     1 -0.251  -3.24  1 -0.736 -2.50  
## 9     1      9      9     1 -0.829  -3.24  1 -0.863 -2.38  
## 10    1     10     10     1 -0.503  -3.24  1 -0.573 -2.67  
## # ... with 499,990 more rows
```

$\hat{i} = \text{obs } i$ 

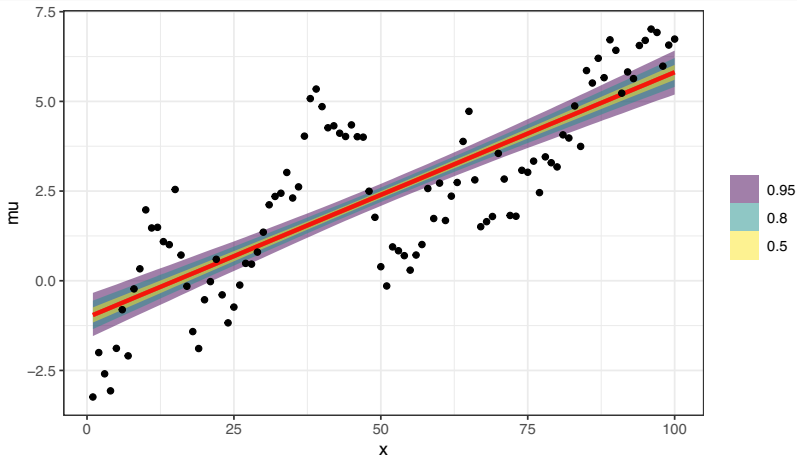
# Predictions

```
df_pred %>% ungroup() %>% filter(.iteration %% 10 == 0) %>%  
ggplot(aes(x=x)) +  
  geom_line(aes(y=mu, group=.iteration), color="red", alpha=0.05) +  
  geom_point(data=d, aes(y=y))
```



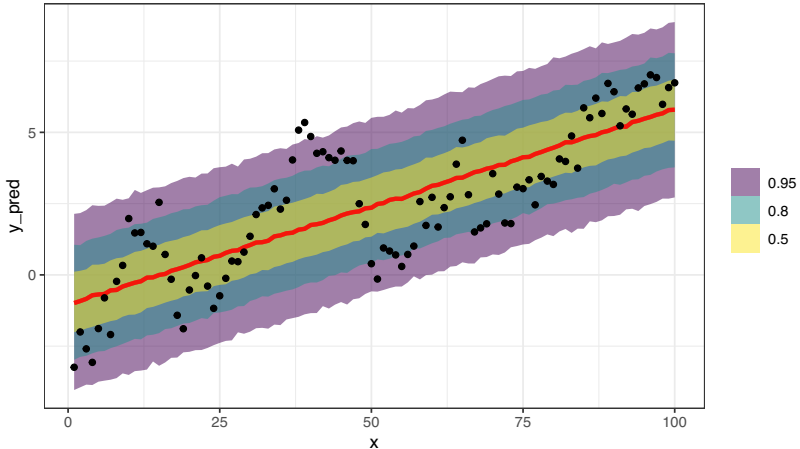
# Posterior distribution ( $\mu$ )

```
df_pred %>% ungroup() %>%  
ggplot(aes(x=x)) +  
  tidybayses::stat_lineribbon(aes(y=mu), alpha=0.5) +  
  geom_point(data=d, aes(y=y))
```



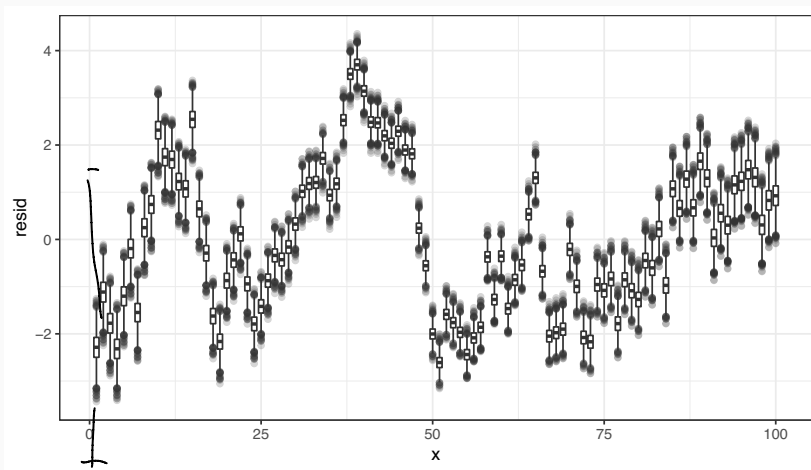
# Posterior predictive distribution ( $y_{pred}$ )

```
df_pred %>% ungroup() %>%  
ggplot(aes(x=x)) +  
  tidybayer::stat_lineribbon(aes(y=y_pred), alpha=0.5) +  
  geom_point(data=d, aes(y=y))
```



# Residual plot

```
df_pred %>% ungroup() %>%  
  ggplot(aes(x=x, y=resid)) +  
  geom_boxplot(aes(group=x), outlier.alpha = 0.2)
```



# Model Evaluation

---

## Model assessment?

If we think back to our first regression class, one common option is  $R^2$  which gives us the variability in  $Y$  explained by our model.

Quick review:




## Model assessment?

If we think back to our first regression class, one common option is  $R^2$  which gives us the variability in  $Y$  explained by our model.

Quick review:

$$\sum_{i=1}^n (Y_i - \bar{Y})^2 = \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2 + \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Total                      Model                      Error



## Model assessment?

If we think back to our first regression class, one common option is  $R^2$  which gives us the variability in  $Y$  explained by our model.

Quick review:

$$\sum_{i=1}^n (Y_i - \bar{Y})^2 \stackrel{\text{OLS}}{=} \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2 + \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Total    Model    Error

$$R^2 = \frac{SS_{model}}{SS_{total}} = \frac{\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2} = \frac{\text{Var}(\hat{\mathbf{Y}})}{\text{Var}(\mathbf{Y})} = \text{Cor}(\mathbf{Y}, \hat{\mathbf{Y}})^2$$

When we compute any statistic for our model we want to do so at each iteration so that we can obtain the posterior distribution of that particular statistic (e.g. the posterior distribution of  $R^2$  in this case).

```
df_R2 = df_pred %>%  
  group_by(.iteration) %>%  
  summarize(  
    R2_classic = var(mu) / var(y),  
    R2_bayes   = var(mu) / (var(mu) + var(resid))  
  )
```

```
df_R2  
## # A tibble: 5,000 x 3  
##   .iteration R2_classic R2_bayes  
##       <int>     <dbl>   <dbl>  
## 1         1     0.622    0.625  
## 2         2     0.569    0.603  
## 3         3     0.587    0.611  
## 4         4     0.584    0.609  
## 5         5     0.560    0.599  
## 6         6     0.520    0.579  
## 7         7     0.504    0.570  
## 8         8     0.610    0.620
```

```
summary(df_R2$R2_classic)
```

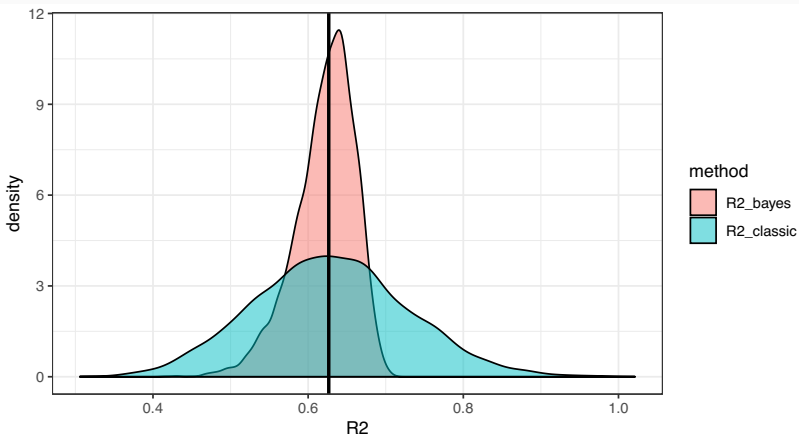
```
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 0.3059  0.5614 0.6271  0.6288  0.6930  1.0211
```

```
summary(df_R2$R2_bayes)
```

```
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 0.4155  0.5994 0.6268  0.6212  0.6488  0.7079
```

# Visually

```
df_R2 %>% tidyr::gather(method, R2, -.iteration) %>%  
  ggplot(aes(x=R2, fill=method)) +  
    geom_density(alpha=0.5) +  
    geom_vline(xintercept=modelr::rsquare(l, d), size=1)
```



## What if we collapsed first?

```
df_pred %>%
  group_by(i) %>%
  summarize(mu = mean(mu), y=mean(y), resid=mean(resid)) %>%
  summarize(
    R2_classic = var(mu) / var(y),
    R2_bayes    = var(mu) / (var(mu) + var(resid))
  )
## # A tibble: 1 x 2
##   R2_classic R2_bayes
##   <dbl>     <dbl>
## 1      0.625     0.626

modelr::rsquare(l, data=d)
## [1] 0.6265565
```

## Some problems with $R^2$

Some new issues,

- $R^2$  doesn't really make sense in the Bayesian context
  - multiple possible definitions with different properties
  - fundamental equality doesn't hold anymore

## Some problems with $R^2$

Some new issues,

- $R^2$  doesn't really make sense in the Bayesian context
  - multiple possible definitions with different properties
  - fundamental equality doesn't hold anymore

Some old issues,

- $R^2$  always increases (or stays the same) when a predictor is added
- $R^2$  is highly susceptible to over fitting
- $R^2$  is sensitive to outliers
- $R^2$  depends heavily on current values of  $Y$
- $R^2$  can differ drastically for two equivalent models (i.e. nearly identical inferences about key parameters)



## Some Other Metrics

---

## Root Mean Square Error

The traditional definition of rmse is as follows

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2}$$

either  $M_i$   
 $Y_{\text{pred}_i}$

The traditional definition of rmse is as follows

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2}$$

In the bayesian context, we have posterior samples from each parameter / prediction of interest so we can express this as

$$\frac{1}{m} \sum_{s=1}^m \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i^s)^2}$$

where  $m$  is the number of iterations and  $\hat{Y}_i^s$  is the prediction for  $Y_i$  at iteration  $s$ .

## Continuous Rank Probability Score

Another approach is the continuous rank probability score which comes from the probabilistic forecasting literature, it compares the full posterior predictive distribution to the observation / truth.

$$\text{CRPS} = \int_{-\infty}^{\infty} (F_{\hat{Y}}(z) - \mathbf{1}_{z \geq Y})^2 dz$$

where  $F_{\hat{Y}}$  is the CDF of  $\hat{Y}$  (the posterior predictive distribution for  $Y$ ) and  $\mathbf{1}_{z \geq Y}$  is the indicator function which equals 1 when  $z \geq Y$ , the true/observed value of  $Y$ .

## Continuous Rank Probability Score

Another approach is the continuous rank probability score which comes from the probabilistic forecasting literature, it compares the full posterior predictive distribution to the observation / truth.

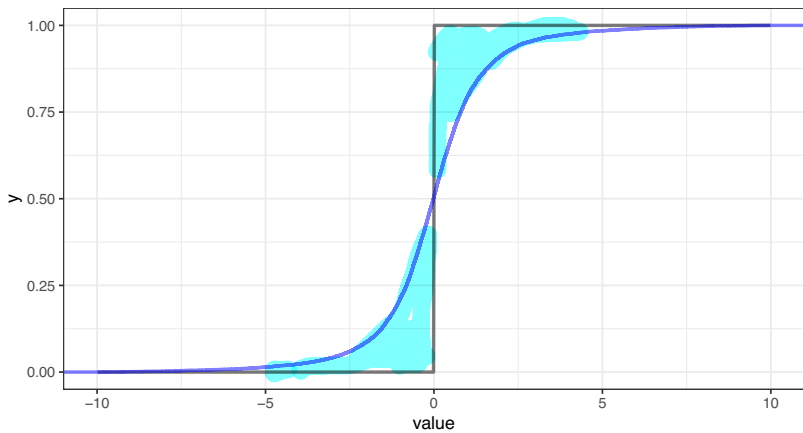
$$\text{CRPS} = \int_{-\infty}^{\infty} (F_{\hat{Y}}(z) - 1_{z \geq Y})^2 dz$$

where  $F_{\hat{Y}}$  is the CDF of  $\hat{Y}$  (the posterior predictive distribution for  $Y$ ) and  $1_{z \geq Y}$  is the indicator function which equals 1 when  $z \geq Y$ , the true/observed value of  $Y$ .

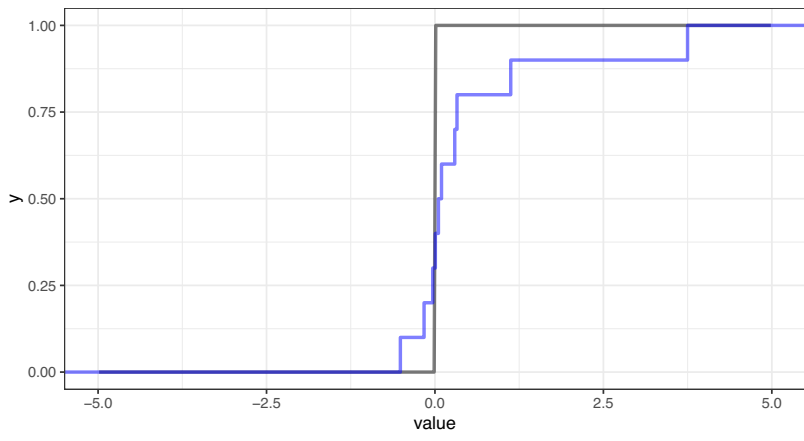
Since this calculates a score for a single probabilistic prediction we can naturally extend it to multiple predictions by calculating an average CRPS

$$\frac{1}{n} \sum_{i=1}^n \int_{-\infty}^{\infty} (F_{\hat{Y}_i}(z) - 1_{z \geq Y_i})^2 dz$$

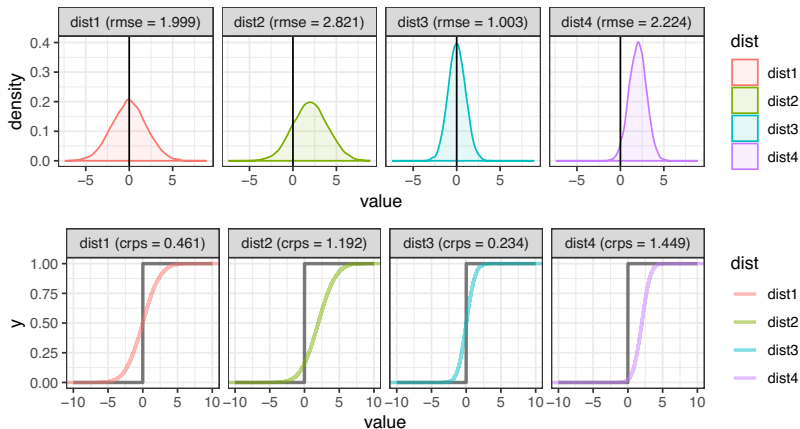
## CDF vs Indicator



## Empirical CDF vs Indicator



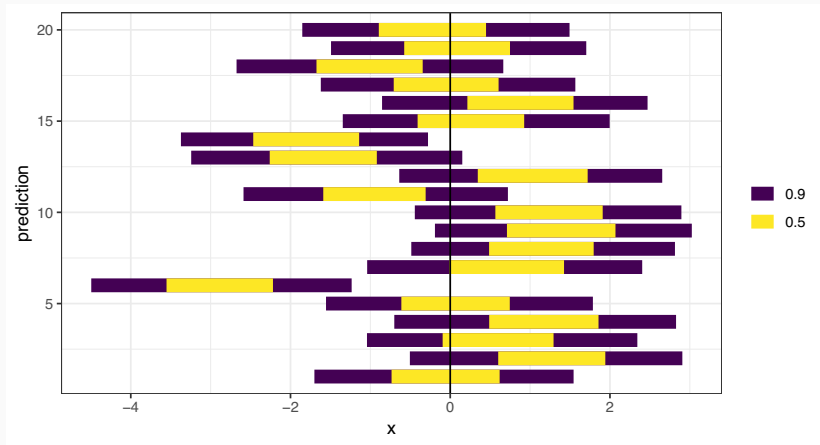
# Accuracy vs. Precision





# Empirical Coverage

One final method, which assesses model calibration is to examine how well credible intervals, derived from the posterior predictive distributions of the  $Y$ s, capture the true/observed values.



Back to our example

---

```
rmse = df_pred %>%  
  group_by(.iteration) %>%  
  summarize(rmse = sqrt( sum( (y - y_pred)^2 ) / n())) %>%  
  pull(rmse)  
  
length(rmse)  
## [1] 5000  
  
head(rmse)  
## [1] 1.973967 1.976420 2.064705 2.181217 2.208194 2.006912  
  
mean(rmse)  
## [1] 2.177006  
  
modelr::rmse(l, data = d)  
## [1] 1.524528
```

↳  $\mu$

```
rmse = df_pred %>%  
  group_by(.iteration) %>%  
  summarize(rmse = sqrt( sum( (y - mu)^2 ) / n())) %>%  
  pull(rmse)
```

```
length(rmse)
```

```
## [1] 5000
```

```
head(rmse)
```

```
## [1] 1.529720 1.537993 1.526788 1.530596 1.529479 1.534994
```

```
mean(rmse)
```

```
## [1] 1.540472
```

```
modelr::rmse(l, data = d)
```

```
## [1] 1.524528
```

```
crps = df_pred %>%  
  group_by(i) %>%  
  summarise(crps = calc_crps(y_pred, y)) %>%  
  pull(crps)
```

```
length(crps)
```

```
## [1] 100
```

```
head(crps)
```

```
## [1] 1.4972794 0.6559097 1.0840541 1.5436106 0.7118661 0.3648843
```

```
mean(crps)
```

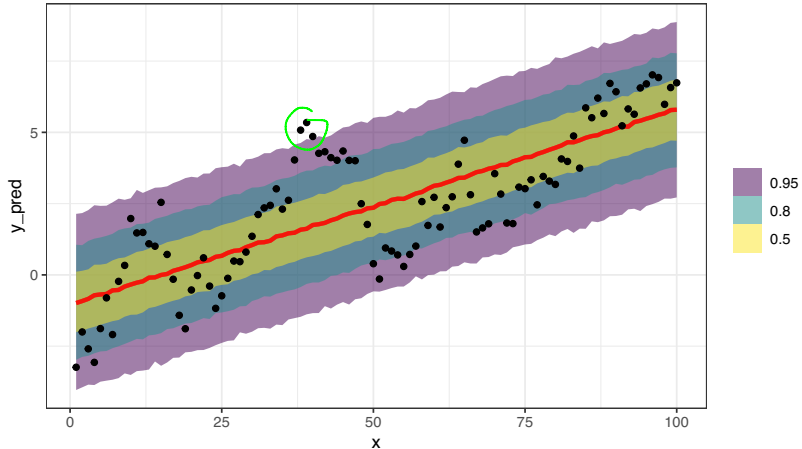
```
## [1] 0.8794432
```

## Empirical Coverage

```
df_cover = df_pred %>%  
  group_by(x,y) %>%  
  tidybayses::mean_hdi(y_pred, .prob = c(0.5,0.9,0.95))  
  
df_cover %>%  
  mutate(contains = y >= .lower & y <= .upper) %>%  
  group_by(prob=.width) %>%  
  summarize(emp_cov = sum(contains)/n())  
## # A tibble: 3 x 2  
##   prob emp_cov  
##   <dbl> <dbl>  
## 1 0.5     0.42  
## 2 0.9     0.94  
## 3 0.95    0.97
```

# Posterior predictive distribution ( $y_{pred}$ )

```
df_pred %>% ungroup() %>%  
ggplot(aes(x=x)) +  
  tidybayer::stat_lineribbon(aes(y=y_pred), alpha=0.5) +  
  geom_point(data=d, aes(y=y))
```



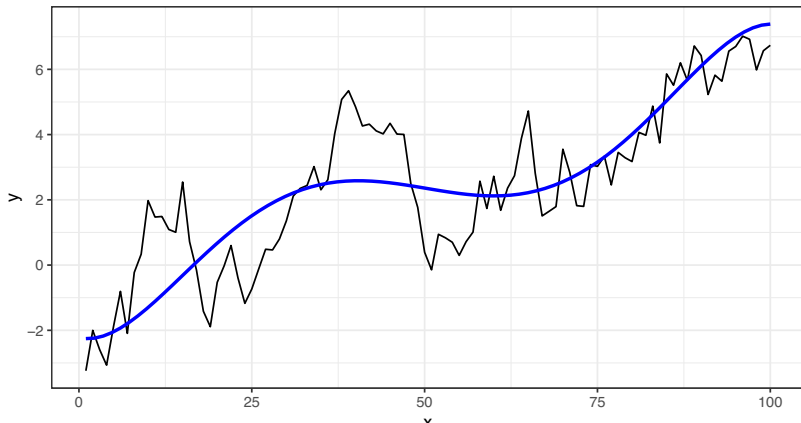
Compared to what?

---



# Polynomial fit

```
ggplot(d, aes(x=x,y=y)) +  
  geom_line() +  
  geom_smooth(  
    method='lm', color="blue", se = FALSE,  
    formula = y~poly(x,5,simple=TRUE)  
  )
```



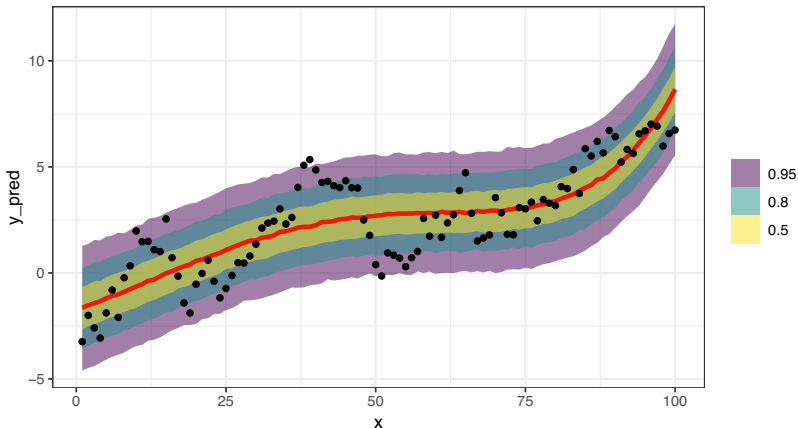
# Model

```
l_p = lm(y~poly(x,5,simple=TRUE), data=d)
summary(l_p)
##
## Call:
## lm(formula = y ~ poly(x, 5, simple = TRUE), data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.63687 -0.84691 -0.06592  0.76911  2.89913
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.4244     0.1325  18.294 < 2e-16 ***
## poly(x, 5, simple = TRUE)1 19.7471     1.3252  14.901 < 2e-16 ***
## poly(x, 5, simple = TRUE)2  0.6263     1.3252   0.473  0.6376
## poly(x, 5, simple = TRUE)3  7.4914     1.3252   5.653 1.68e-07 ***
## poly(x, 5, simple = TRUE)4 -0.0290     1.3252  -0.022  0.9826
## poly(x, 5, simple = TRUE)5 -3.2899     1.3252  -2.483  0.0148 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.325 on 94 degrees of freedom
## Multiple R-squared:  0.7348, Adjusted R-squared:  0.7206
## F-statistic: 52.08 on 5 and 94 DF,  p-value: < 2.2e-16
```

```
poly_model =  
"model{  
  # Likelihood  
  for(i in 1:length(y)){  
    y[i] ~ dnorm(mu[i], tau)  
    y_pred[i] ~ dnorm(mu[i], tau)  
    mu[i] = beta[1] + beta[2]*x[i] + beta[3]*x[i]^2 +  
            beta[4]*x[i]^3 + beta[5]*x[i]^4 + beta[6]*x[i]^5  
  }  
  
  # Prior for beta  
  for(j in 1:6){  
    beta[j] ~ dnorm(0,1/1000)  
  }  
  
  # Prior for sigma / tau2  
  tau ~ dgamma(1, 1)  
  sigma2 = 1/tau  
}"
```

# Posterior Predictive Distribution

```
df_poly %>% ungroup() %>%  
ggplot(aes(x=x)) +  
  tidybayer::stat_linribbon(aes(y=y_pred), alpha=0.5) +  
  geom_point(data=d, aes(y=y))
```



## Comparing Results

```
## Joining, by = c("parameter", "model")  
## Joining, by = c("parameter", "model")
```

metric	parameter	y~x	y~poly(x,5)
rmse	mu	1.540	1.416
rmse	y_pred	2.177	2.003
crps	mu	NA	NA
crps	y_pred	0.879	0.798
emp_cov (90%)	mu	NA	NA
emp_cov (90%)	y_pred	0.940	0.930