Regression: predictive modelling

ENVX2001 Applied Statistical Methods

Liana Pozza

Feb 2026

Predictive modelling

> “The best way to predict the future is to create it.”
>
> – Peter Ferdinand Drucker, 1909–2005

Our workflow so far

Workflow

1. Model development
   - Explore: visualise, summarise
   - Transform predictors: linearise, reduce skewness/leverage
   - Model: fit, check assumptions, interpret, transform. Repeat.
2. Variable selection
   - VIF: remove predictors with high variance inflation factor
   - Model selection: stepwise selection, AIC, principle of parsimony, assumption checks
3. Predictive modelling
   - **Predict**: Use the model to predict new data
   - **Validate**: Evaluate the model's performance

Previously on ENVX2001...

We fitted a multiple linear regression model to the data.

```
fit ← lm(log(Ozone) ~ Temp + Solar.R + Wind, data = airquality)
summary(fit)
```

```

Call:
lm(formula = log(Ozone) ~ Temp + Solar.R + Wind, data = airquality)

Residuals:
     Min       1Q   Median       3Q      Max 
-2.06193 -0.29970 -0.00231  0.30756  1.23578 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.2621323  0.5535669  -0.474 0.636798    
Temp         0.0491711  0.0060875   8.077 1.07e-12 ***
Solar.R      0.0025152  0.0005567   4.518 1.62e-05 ***
Wind        -0.0615625  0.0157130  -3.918 0.000158 ***
```

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

Residual standard error: 0.5086 on 107 degrees of freedom
  (42 observations deleted due to missingness)
Multiple R-squared:  0.6644,    Adjusted R-squared:  0.655
F-statistic: 70.62 on 3 and 107 DF,  p-value: < 2.2e-16
```

$$\widehat{log(Ozone)} = -0.262 + 0.0492 \cdot Temp + 0.00252 \cdot Solar.R - 0.0616 \cdot Wind$$

Predictions by hand

$$\widehat{log(Ozone)} = -0.262 + 0.0492 \cdot Temp + 0.00252 \cdot Solar.R - 0.0616 \cdot Wind$$

On a certain day, we measured (*in imperial units*):

- temperature `Temp` to be 80 degrees Fahrenheit
- solar radiation `Solar.R` to be 145 units (Langleys)
- wind speed `Wind` to be 10.9 miles per hour

What is the predicted ozone level?

$$\widehat{log(Ozone)} = -0.262 + 0.0492 \cdot 80 + 0.00252 \cdot 145 - 0.0616 \cdot 10.9$$

Easy! The two things we need to think about are...

- **What is the uncertainty in this prediction?**
- **Can this model be used to predict ozone if we collect new data in the future?**

Uncertainty

- **Confidence interval**: uncertainty in the **mean** response at a given predictor value.
- **Prediction interval**: uncertainty in a **single** response at a given predictor value.

What it means

> 95% confidence interval: Given the parameters of the model, we are 95% confident that the *mean* response at a given predictor value is between y_1 and y_2.

> 95% prediction interval: Given the parameters of the model, we are 95% confident that a *single* response at a given predictor value is between y_1 and y_2.

Equations

The equation to calculate any prediction interval is:

$$\hat{y} \pm t_{\alpha/2} \cdot se(\hat{y})$$

where:

- $\hat{y}$ is the predicted value
- $t_{\alpha/2}$ is the critical value of the t-distribution for a given confidence level
- $se(\hat{y})$ is the standard error of the prediction

The difference in calculating a confidence interval and a prediction interval is in the standard error of the prediction.

Equations

CI: standard error of the fit

$$se(\hat{y}) = \sqrt{MSE \cdot \left(\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \right)}$$

PI: standard error of the prediction

$$se(\hat{y}) = \sqrt{MSE \cdot \left(1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \right)}$$

- x_0 is value of the predictor for which we want a response
- MSE is the mean squared error of the fit (SS_{xx})
- $\sum_{i=1}^{n} (x_i - \bar{x})^2$ is the sum of squares of the predictor values
- n is the number of observations
- $\bar{x}$ is the mean of the predictor values

The prediction interval formula has an additional term $(1 \cdot MSE)$. There is uncertainty that the *mean* prediction will be similar to the observed, and additional uncertainty/variability for a *single* response (equivalent to the MSE). **Thus the confidence interval is always narrower than the prediction interval.**

Predictions in R

- First, we need to create a new data frame with the predictor values we want to predict at – it must include all variables in the fitted model.

```
predict_df ← data.frame(Temp = 80, Solar.R = 145, Wind = 10.9)
```

- We use the `predict()` function to obtain the predicted value.
- Specifying `interval = "confidence"` or `interval = "prediction"` also calculates the confidence or prediction interval.

```
predict(fit, newdata = predict_df) # the predicted value
```

```
       1
3.365227
```

```
predict(fit, newdata = predict_df, interval = "confidence") # predicted value and CI
```

```
       fit      lwr      upr
1 3.365227 3.246265 3.484189
```

```
predict(fit, newdata = predict_df, interval = "prediction") # predicted value and PI
```

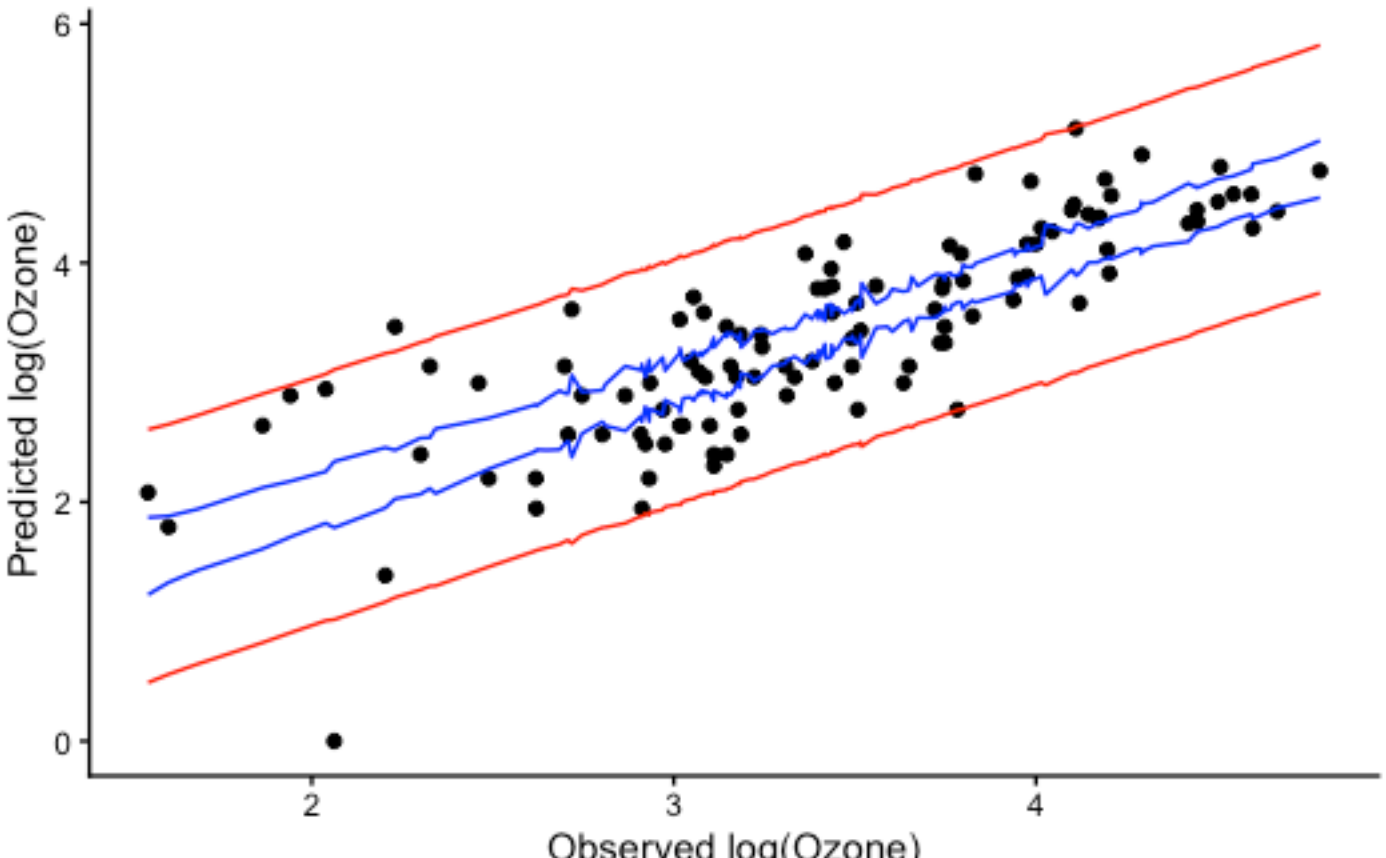
```
       fit      lwr      upr
1 3.365227 2.350051 4.380404
```

Visualising CI and PI

```
airquality$pred ← predict(fit, newdata = airquality) # predict for existing data

preds_ci ← predict(fit, newdata = airquality, interval = "confidence") # confidence interval
preds_pi ← predict(fit, newdata = airquality, interval = "prediction") # prediction interval
```

- We can now plot the CI and PI as shaded areas around the predicted line

```
p ←
  ggplot(airquality, aes(pred, log(Ozone))) +
  geom_point() +
  geom_line(data = preds_ci, aes(fit, lwr), color = "blue") +
  geom_line(data = preds_ci, aes(fit, upr), color = "blue") +
  geom_line(data = preds_pi, aes(fit, lwr), color = "red") +
  geom_line(data = preds_pi, aes(fit, upr), color = "red") +
  labs(x = "Observed log(Ozone)", y = "Predicted log(Ozone)") +
  theme_classic()
p
```


Predicted log(Ozone)
Observed log(Ozone)
6
4
2
0
2
3
4

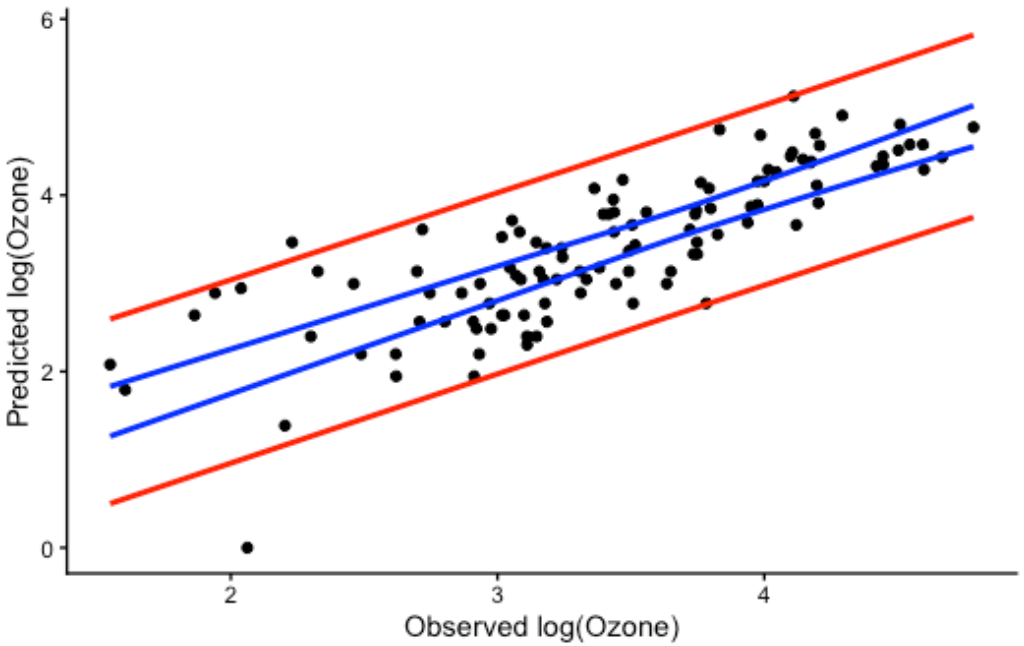
Visualising with `geom_smooth()`

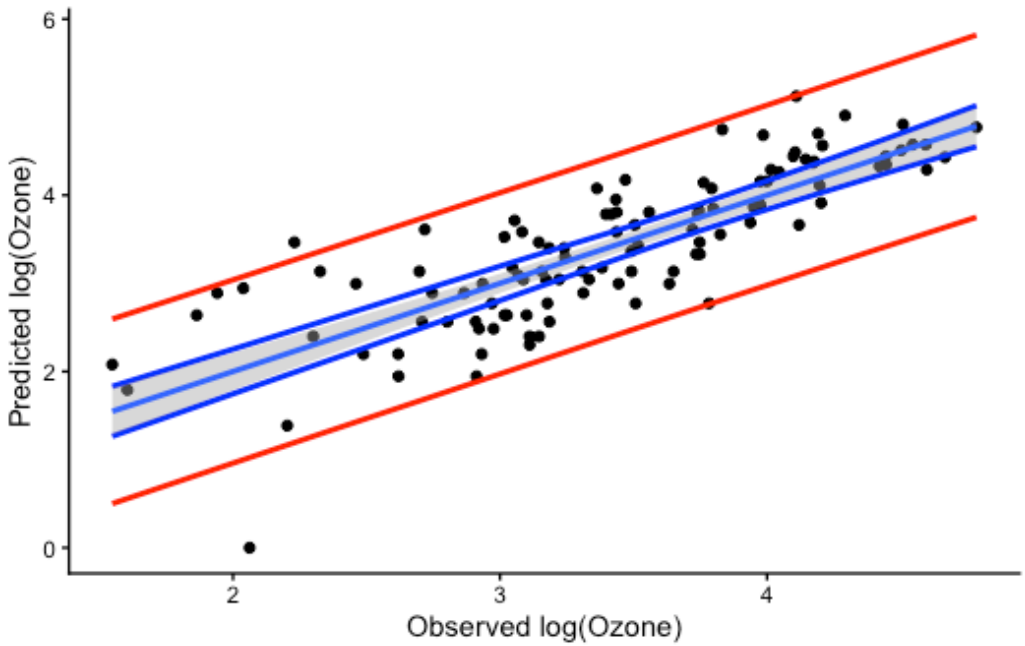
- `geom_smooth()` fits a linear model to obtain a smooth line
- For visualisation we can use `geom_smooth()` instead of `geom_line()` to fit smoothed lines

```
p ←
  ggplot(airquality, aes(pred, log(Ozone))) +
  geom_point() +
  geom_smooth(data = preds_ci, aes(fit, lwr),
color = "blue", se = F) +
  geom_smooth(data = preds_ci, aes(fit, upr),
color = "blue", se = F) +
  geom_smooth(data = preds_pi, aes(fit, lwr),
color = "red", se = F) +
  geom_smooth(data = preds_pi, aes(fit, upr),
color = "red", se = F) +
  labs(x = "Observed log(Ozone)", y = "Predicted
log(Ozone)") +
  theme_classic()
p
```

- The hyperparameter `se = TRUE` fits a smoothed CI around the predicted line
- Smoothed with loess functions
- Cannot fit the prediction interval

```
p + geom_smooth(method = "lm", se = TRUE)
```


Predicted log(Ozone)
Observed log(Ozone)
0
2
4
6
2
3
4


Predicted log(Ozone)
Observed log(Ozone)
0
2
4
6
2
3
4

Calibration and validation

All is good when we want to assess uncertainty in a model that we have already fit.

What if we want to know how well the model predicts **new** data, i.e. data that we did not use to fit the model?

Why separate calibration and validation data?

In model development we try many models and choose the one that *fits that specific dataset* very well.

So our model *may be* too complex and overfits the data. If we predict onto new data (in the real world) **the model does not give plausible predictions**.

Why might this be a problem?

- Predict the wrong **ozone levels** (and people with respiratory issues are not warned)
- Predict the wrong **disease numbers** (and local health services are not prepared)
- Predict the wrong **crop yield** (and farmers under/overapply fertiliser)

Predictions can directly be used for decision-making, which has consequences.

General Idea

What if we want to know how well the model predicts **new** data, i.e. data that we did not use to fit the model?

- We build the model with one dataset (calibration).
- We validate the model's predictions with an **independent dataset**.
  - If the model is good, we expect the predictions to be *close* to the actual values.
  - If the model is bad, we expect the predictions to be *far* from the actual values.
- The dataset can be obtained by:
  - Collecting new data.
  - Splitting the existing data into two parts before model building.
    - Data splitting
    - Cross-validation

Definitions

Sometimes the terms for calibration and validation can get muddled.

Best practice:

- Calibration/Training Dataset: the data used to train the model
- Validation: the data used to fine tune the model (e.g. variable selection, hyperparameters in machine learning)
- Test: remaining data that has not been used in any kind of model training

To keep things simple (and if datasets are small), also common:

- Calibration/Training: the data used to train the model
- Validation/Test: the data used to assess the model's prediction performance

Our data

Dataset

Collecting new data

Dataset (train)

+

New dataset (test)

- The best way to assess how well a model predicts new data is to collect new data.
  - **Training set**: used to fit the model.
  - **Test set**: used to assess how well the model predicts new data.

Collecting new data

Pros

- The new data is completely independent of the data used to fit the model.
- More data to *fit* and *validate* compared to data splitting.

Cons

- It can be expensive and time-consuming to collect new data.
- Some data may be impossible to collect (e.g. historical data).

Data splitting

Dataset

Data splitting

Dataset

(Training)

Subset (Test)

Data splitting

Dataset

(Training)

Subset (Test)

- Split the existing dataset into training and test datasets (80:20, 70:30, 60:40, etc.)
  - **Training set**: used to fit the model.
  - **Test set**: used to assess how well the model predicts new data.

Data splitting

Dataset

(Training)

Subset (Test)

- Only possible for larger datasets (hundreds of observations)
- Split the existing dataset into calibration, validation and test datasets (70:15:15, etc.)
  - **Calibration set**: used to fit the model.
  - **Validation set**: used to test model development (prevent overfitting).
  - **Test set**: used to assess how well the model predicts new data.

Data splitting

Pros

- Compared to collecting new data, it is cheaper and faster to split existing data.

Cons

- We have *less* data to fit the model and *less* data to validate the model.
- How do we split the data? Randomly? By time? By location?

k-fold cross-validation

k-fold cross-validation

k-fold cross-validation

k-fold cross-validation

Iteration 1

Iteration 2

Iteration 3

And so on...

k-fold cross-validation

Iteration 1 Iteration 2 Iteration 3 3-fold cross-validation Fold 1 Fold 2 Fold 3

- Like data splitting, where existing data is split into two parts:
  - **Training set**: used to fit the model.
  - **Test set**: used to assess how well the model predicts new data.
- The **difference** is that the splitting is done *multiple* times, and the model is fit and validated *multiple* times.
- Each iteration or fold is used for testing once.

k-fold cross-validation

Pros

- Same as data splitting, but also:
  - The model is fit and validated *multiple* times, so we can get a better estimate of how well the model predicts new data.
  - Greatly reduces overfitting as the model's performance is not just a result of the particular way the data was split.

Cons

- Bias in small datasets: each fold may contain too little data to provide a representative sample.
- Each fold fits a **new** model so it is not used for interpretation, only for prediction quality.
  - Computationally more expensive.

Cross-validation

k-fold cross-validation splits data in each fold randomly.

If there is some underlying structure to the data, consider:

- Spatial cross-validation (e.g. fields on a farm, each fold is one field)
- Temporal cross-validation (e.g. time series data, each fold is a time period)
- Stratified cross-validation (e.g. each fold has the same proportion of each category)

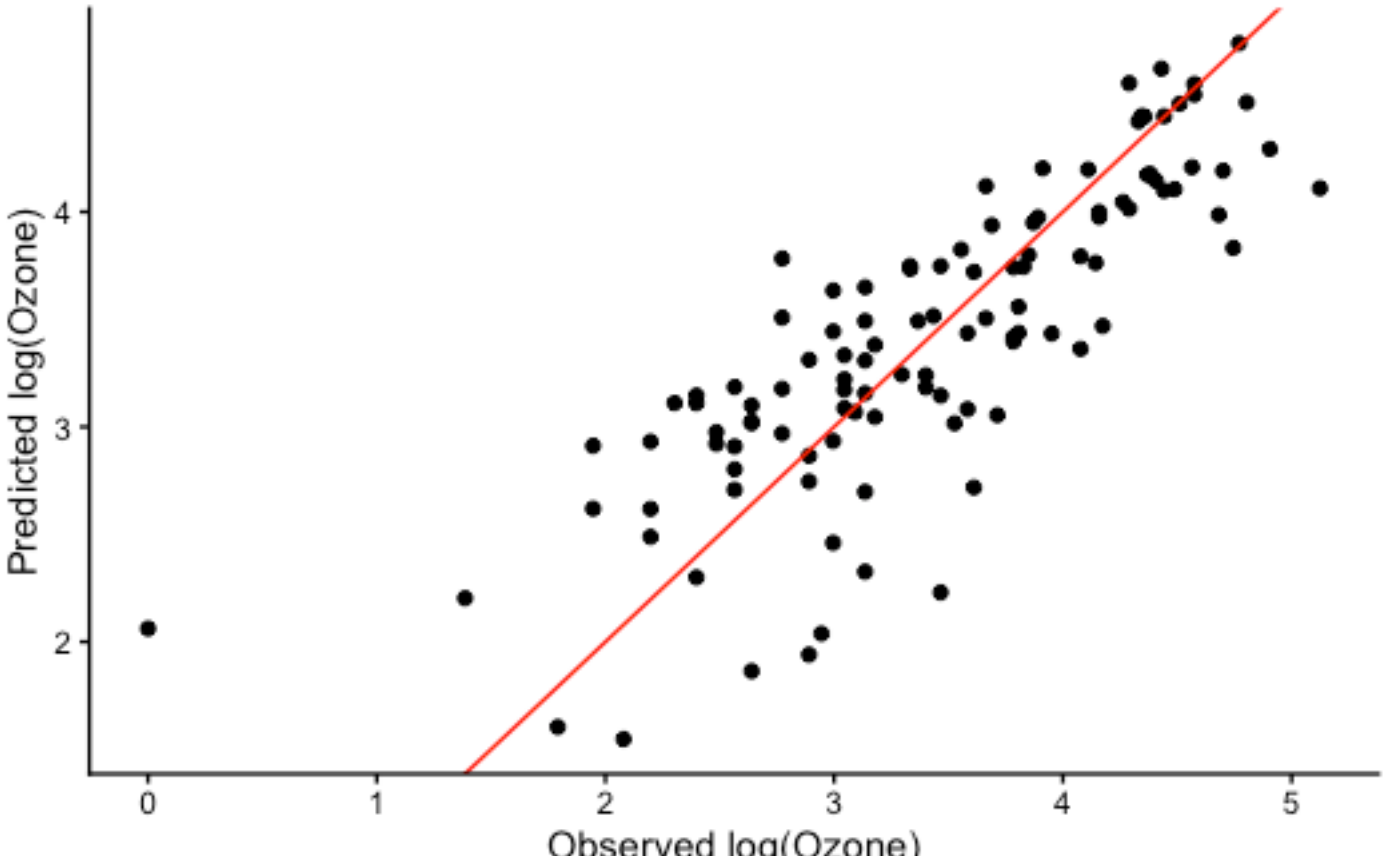
Assessing prediction quality

How ‘good’ are the predictions? Observed vs predicted.

Visually

- **Plot**: observed (y_i) vs predicted ($\hat{y}$) values

```
ggplot(data = airquality, aes(x = log(Ozone), y = pred)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "Observed log(Ozone)", y = "Predicted log(Ozone)") +
  theme_classic()
```



We can also use metrics to quantify how well the model predicts new data:

- How close points are to the 1:1 line
- How *linear* the relationship is between observed and predicted values

Error

The smaller the error, the better the model.

Mean error: the average difference between observed and predicted values.

- Can be positive or negative to indicate over- or under-estimation (a measure of bias)

$$ME = \frac{1}{n}\sum_{i=1}^{n} y_i - \hat{y}_i$$

(in y units)

Mean absolute error: the average (absolute) difference between observed and predicted values (residual).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

(in y units)

Mean squared error: the average of the squared residuals

- Squared so positive and negative errors do not cancel each other out

- Penalises poor predictions more

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root mean squared error: the standard deviation of the residuals

- Squaring the error penalises poor predictions more

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

(in y units)

Linearity

The more linear the relationship between observed and predicted values, the better the model.

Pearson's correlation coefficient (r):

- A measure of the strength and direction of a linear relationship between two variables.
- Ranges from −1 to 1, with 0 indicating no relationship and 1 indicating a perfect positive linear relationship.

$$r = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2 \sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}}$$

R^2:

- The proportion of variance explained by the variables in the model.
- When two variables are compared (e.g. observed vs predicted) it is the same as correlation squared.
- A value of 1 indicates a perfect linear relationship.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Lin's concordance correlation coefficient (ρ_c):

- A measure of agreement between two variables (based on covariance, variances, and difference in means).
- How well the points fit the 1:1 line.
- A value of 0 indicates no agreement and 1 indicating perfect agreement.

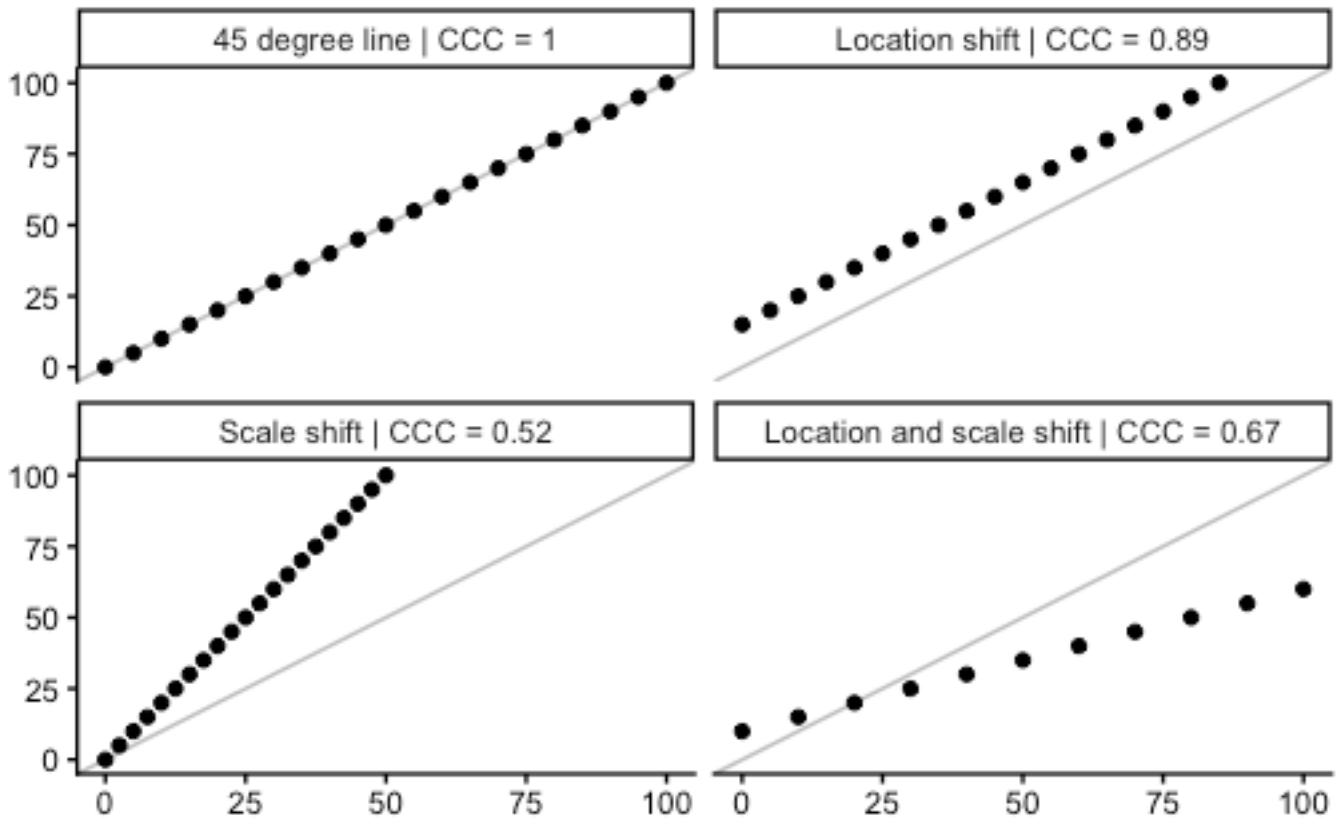
$$LCCC = \frac{2 \text{ Cov } (X, Y)}{\text{Var } (X) + \text{Var } (Y) + (\mu_X - \mu_Y)^2}$$

LCCC vs r and R^2

LCCC combines error and linearity so it better measures the fit to the 1:1 line.

```
df ← tibble(y = seq(0, 100, 5),
  "45 degree line | CCC = 1" = seq(0, 100, 5)) %>%
  mutate("Location shift | CCC = 0.89" = `45 degree line | CCC = 1` - 15) %>%
  mutate("Scale shift | CCC = 0.52" = y / 2) %>%
  mutate("Location and scale shift | CCC = 0.67" = y * 2 - 20)

# pivot
df_long ← df %>%
  pivot_longer(-1, values_to = "x") %>%
  mutate(name = factor(name,
    levels = c("45 degree line | CCC = 1",
      "Location shift | CCC = 0.89",
      "Scale shift | CCC = 0.52",
      "Location and scale shift | CCC = 0.67")))

ggplot(df_long, aes(x, y)) +
  geom_abline(intercept = 0, slope = 1, size = 0.5, colour = "grey") +
```

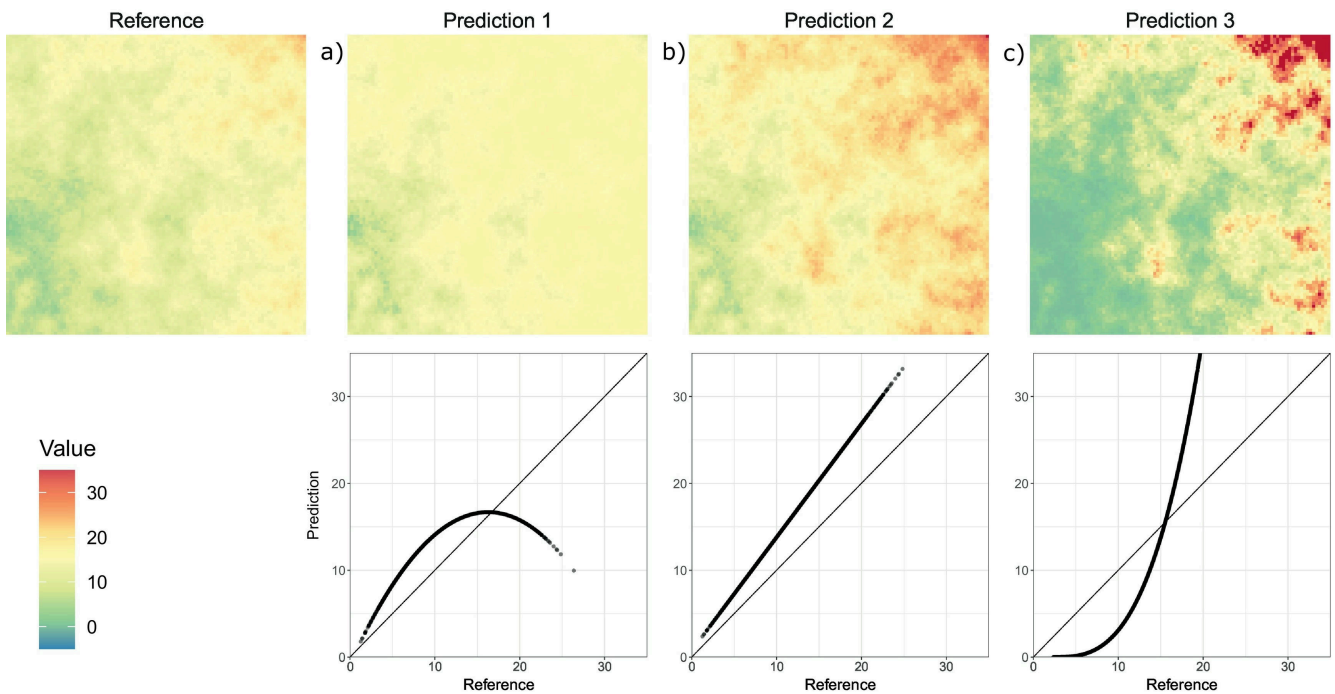
```
  facet_wrap(~name) +
  geom_point() +
  xlim(0, 100) +
  labs(x = "", y = "") +
  theme_classic() +
  geom_blank()
```


45 degree line | CCC = 1
Location shift | CCC = 0.89
Scale shift | CCC = 0.52
Location and scale shift | CCC = 0.67
0
25
50
75
100

Limitations

No one metric is perfect.

Each prediction below has an LCCC of 0.6.



Wadoux and Minasny 2024

Tip

Use multiple metrics to test prediction quality, and always plot the predicted vs observed.

Example: Loyn dataset

We will go through several examples to practice data splitting, cross-validation, and model evaluation.

About

Data on the relationship between bird abundance (bird ha^{-1}) and the characteristics of forest patches at 56 locations in SE Victoria.

The predictor variables are:

- `ALT` Altitude (m)
- `YR.ISOL` Year when the patch was isolated (years)
- `GRAZE` Grazing (coded 1-5 which is light to heavy)
- `AREA` Patch area (ha)
- `DIST` Distance to nearest patch (km)
- `LDIST` Distance to largest patch (km)

```
loyn ← read_csv("data/loyn.csv")
```

Dataset splitting

We will split the data into training and test sets.

As the dataset is quite small, we will use a 80:20 split.

```
set.seed(100)
indexes ← sample(1:nrow(loyn), size = 0.2 * nrow(loyn)) # randomly sample 20% of rows in the
dataset
loyn_train ← loyn[-indexes, ] # remove the 20% - training dataset
loyn_test ← loyn[indexes, ] # select the 20% - test dataset
```

Checking the split

Check out the `str()` of the data to see if the split worked (number of observations).

```
str(loyn_train)
```

```
tibble [45 × 7] (S3: tbl_df/tbl/data.frame)
 $ ABUND  : num [1:45] 5.3 2 1.5 17.1 13.8 3.8 2.2 3.3 27.6 1.8 ...
 $ AREA   : num [1:45] 0.1 0.5 0.5 1 1 1 1 1 2 2 ...
 $ YR.ISOL: num [1:45] 1968 1920 1900 1966 1918 ...
 $ DIST   : num [1:45] 39 234 104 66 246 467 284 156 66 93 ...
 $ LDIST  : num [1:45] 39 234 311 66 246 ...
 $ GRAZE  : num [1:45] 2 5 5 3 5 5 5 4 3 5 ...
 $ ALT    : num [1:45] 160 60 140 160 140 90 60 130 210 160 ...
```

```
str(loyn_test)
```

```
tibble [11 × 7] (S3: tbl_df/tbl/data.frame)
 $ ABUND  : num [1:11] 3 29.5 26 39.6 34.4 19.5 14.6 28.3 15.8 5 ...
 $ AREA   : num [1:11] 1 973 18 49 96 6 2 34 5 4 ...
```

```
$ YR.ISOL: num [1:11] 1900 1970 1966 1972 1976 ...
$ DIST   : num [1:11] 311 337 40 1427 39 ...
$ LDIST  : num [1:11] 571 1323 3188 1557 519 ...
$ GRAZE  : num [1:11] 5 1 2 1 2 3 1 1 3 5 ...
$ ALT    : num [1:11] 130 190 190 180 175 170 210 110 130 120 ...
```

Model development

From now on, we will work with the **training set only**.

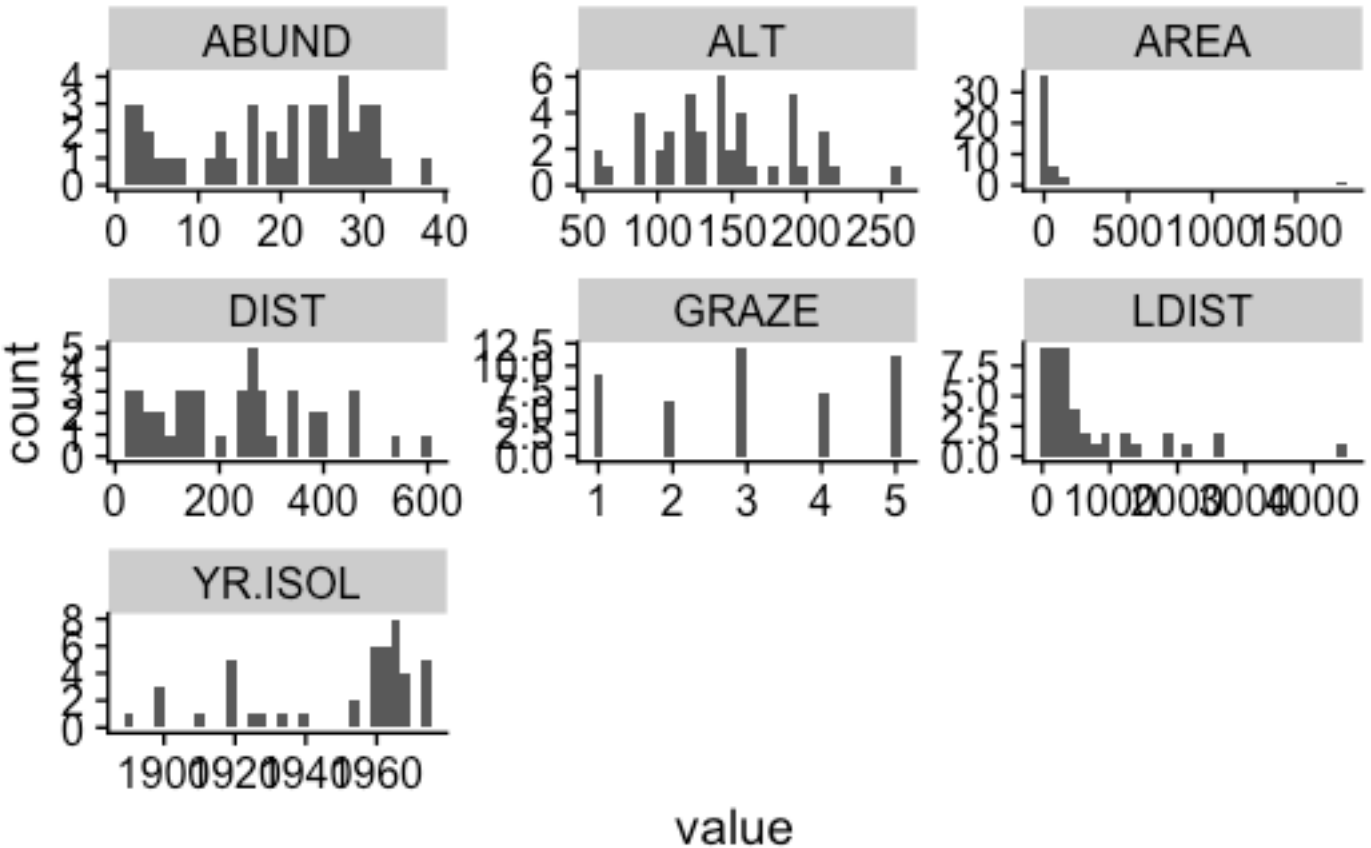
Exploratory data analysis

- The next step is to visualise the data.
- Expore relationships between the predictors and the response via histograms, scatterplots, boxplots, correlations etc.

In this lecture we will just look at histograms.

Histograms

```
loyn_train %>%
    pivot_longer(
    cols = everything(),
    names_to = "variable",
    values_to = "value"
  ) %>%
  ggplot(aes(x = value)) +
  geom_histogram() +
  facet_wrap(~ variable, scales = "free")
```



- Looks like `AREA` `LDIST` and `DIST` are skewed – we will transform them so that they are more normally distributed.

Transforming predictors

We will use `log10()` to transform the predictors. The `mutate()` function from the `dplyr` package is useful for this as it can create new columns in the data frame with the transformed values.

```
loyn_train ← loyn_train %>%
    mutate(
        AREA_L10 = log10(AREA),
        LDIST_L10 = log10(LDIST),
        DIST_L10 = log10(DIST)
    )
```

Then, remove the untransformed variables from the dataset. Here we can use the `select()` function from the `dplyr` package to "delselect" columns by using the `-` sign.

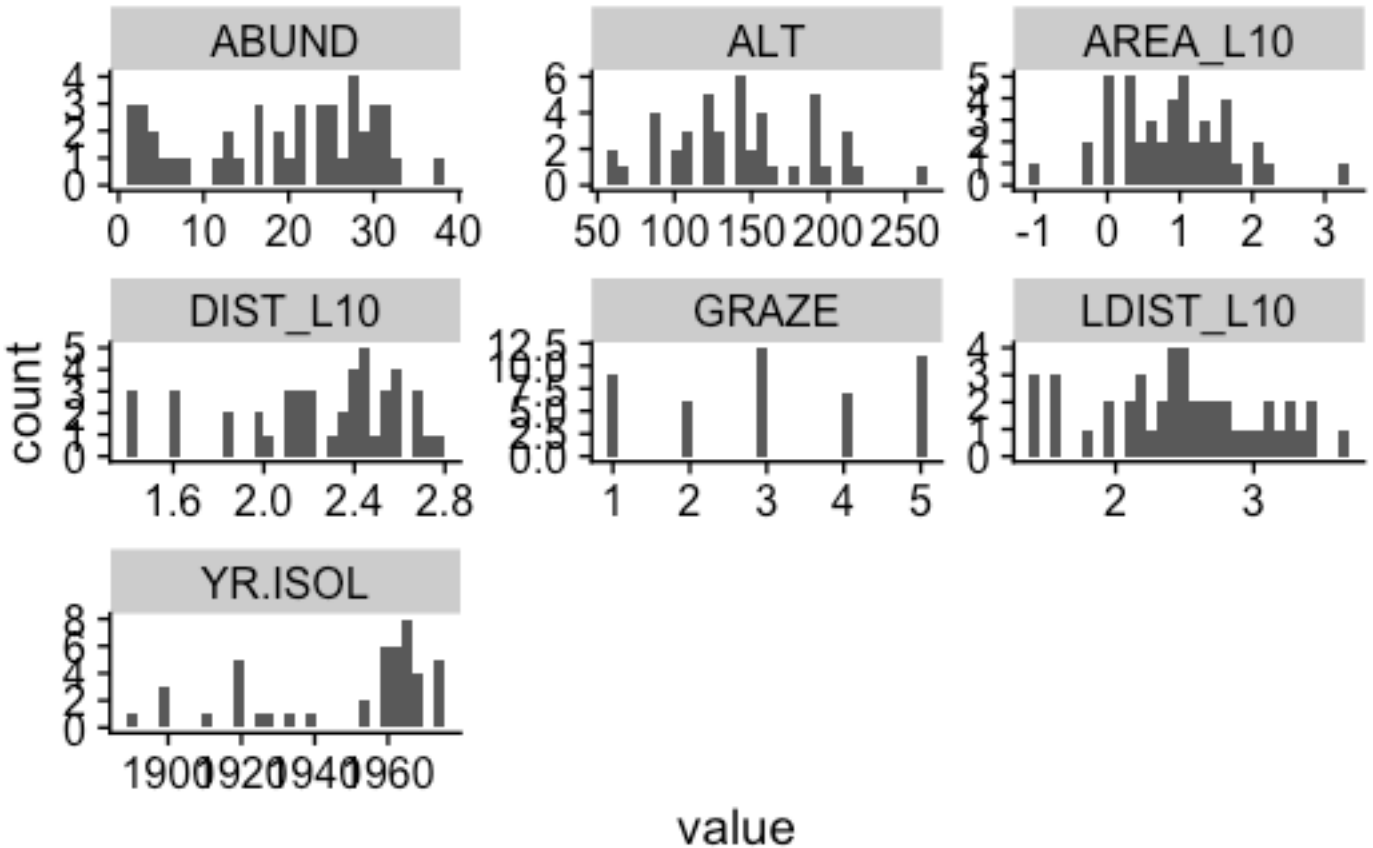
```
loyn_train ← loyn_train %>%
    select(-AREA, -LDIST, -DIST)

str(loyn_train)
```

```
tibble [45 × 7] (S3: tbl_df/tbl/data.frame)
 $ ABUND    : num [1:45] 5.3 2 1.5 17.1 13.8 3.8 2.2 3.3 27.6 1.8 ...
 $ YR.ISOL  : num [1:45] 1968 1920 1900 1966 1918 ...
 $ GRAZE    : num [1:45] 2 5 5 3 5 5 5 4 3 5 ...
 $ ALT      : num [1:45] 160 60 140 160 140 90 60 130 210 160 ...
 $ AREA_L10 : num [1:45] -1 -0.301 -0.301 0 0 ...
 $ LDIST_L10: num [1:45] 1.59 2.37 2.49 1.82 2.39 ...
 $ DIST_L10 : num [1:45] 1.59 2.37 2.02 1.82 2.39 ...
```

Final inspection

View the histograms again to check that the transformation worked.

```
loyn_train %>%
    pivot_longer(
    cols = everything(),
    names_to = "variable",
    values_to = "value"
  ) %>%
  ggplot(aes(x = value)) +
  geom_histogram() +
  facet_wrap(~ variable, scales = "free")
```


ABUND
ALT
AREA_L10
DIST_L10
GRAZE
LDIST_L10
YR.ISOL
count
value

Full model

We start with a full model that includes all the predictors.

```
full_fit ← lm(ABUND ~ ., data = loyn_train)
summary(full_fit)
```

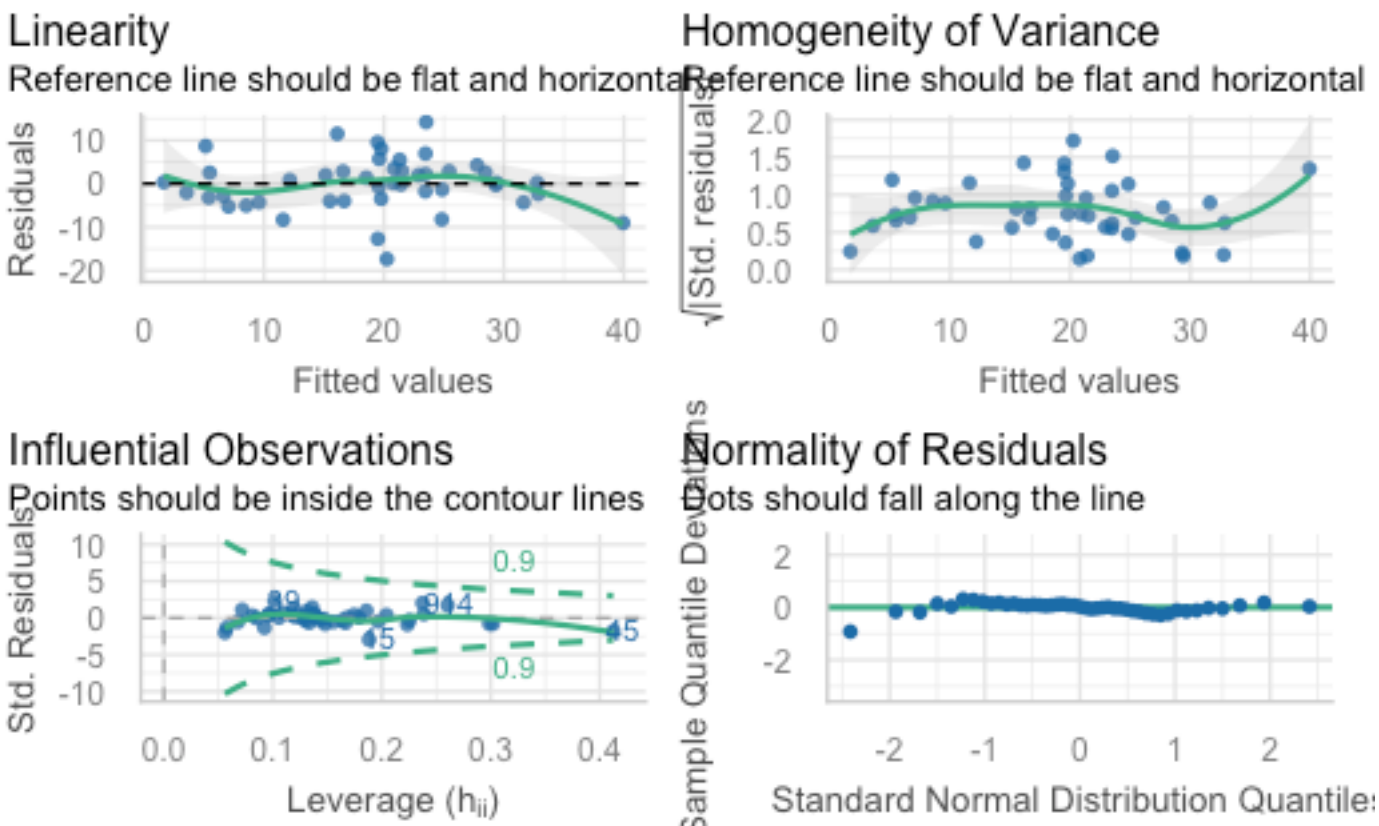
```

Call:
lm(formula = ABUND ~ ., data = loyn_train)

Residuals:
     Min       1Q   Median       3Q      Max
-17.3445  -3.4647   0.1991   2.8689  14.1844

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -159.27533  109.13660  -1.459   0.1527
YR.ISOL        0.09334    0.05392   1.731   0.0916 .
GRAZE         -1.40912    1.03653  -1.359   0.1820
ALT            0.01657    0.02810   0.589   0.5590
```

```
AREA_L10       8.09629    1.78591   4.533 5.63e-05 ***
LDIST_L10      2.05115    3.23927   0.633   0.5304
DIST_L10      -6.18596    4.83189  -1.280   0.2082
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.497 on 38 degrees of freedom
Multiple R-squared:  0.6752,    Adjusted R-squared:  0.6239
F-statistic: 13.17 on 6 and 38 DF,  p-value: 5.277e-08
```

Assumptions - Round 1

As usual, we should check the assumptions of the model (CLINE + outliers). We will use the `check_model()` function from the `perfomance` package (because it looks nice and has interpretation instructions).

```
performance::check_model(full_fit, check = c("linearity", "qq", "homogeneity", "outliers"))
```


Linearity
Reference line should be flat and horizontal
Residuals
10
0
-10
-20
0
10
20
30
40
Fitted values
Homogeneity of Variance
Reference line should be flat and horizontal
√|Std. residuals|
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
Fitted values
Influential Observations
Points should be inside the contour lines
Std. Residuals
10
5
0
-5
-10
0.9
0.9
0.0
0.1
0.2
0.3
0.4
Leverage (h_ii)
Normality of Residuals
Dots should fall along the line
Sample Quantile Deviations
2
0
-2
-2
-1
0
1
2
Standard Normal Distribution Quantiles

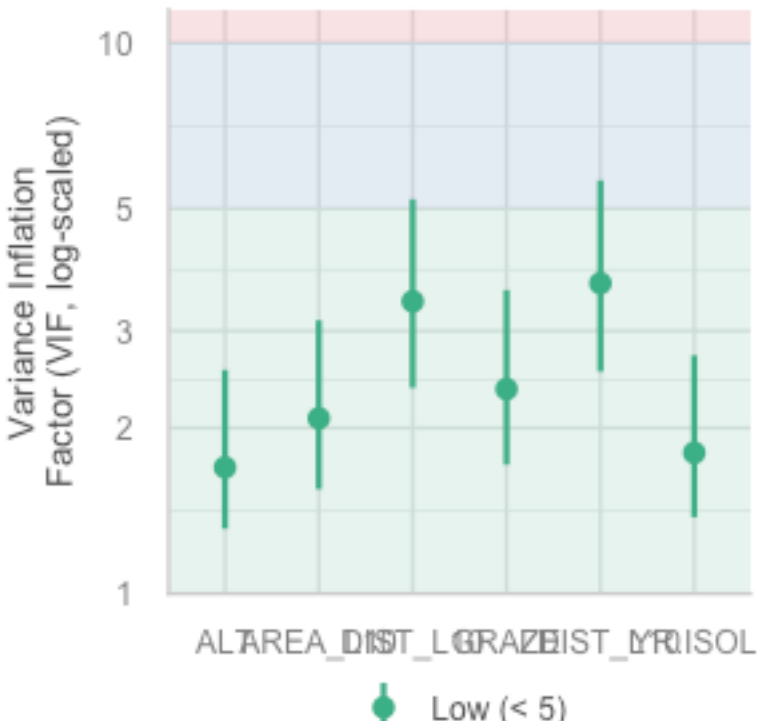
Assumptions - Round 1

We check multicollinearity with variable inflation factors (VIF) - VIFs are all < 10, so there is no multicollinearity. All assumptions are thus met.

```
check_model(full_fit, check = c("vif"))
```

Collinearity

High collinearity (VIF) may inflate parameter uncertainty



Backwards stepwise selection

Use the `step()` function perform backwards stepwise selection. This function uses AIC to select the best model.

Depending on the dataset splitting, the best model may be different each time we randomly sample the data. In this case we should all have the same results as we set the seed.

If we compare to the full model, the adjusted r-squared is slightly higher, and the AIC is lower.

```
step_fit ← step(full_fit, direction = "backward")
```

```
Start:  AIC=174.81
ABUND ~ YR.ISOL + GRAZE + ALT + AREA_L10 + LDIST_L10 + DIST_L10

             Df Sum of Sq    RSS    AIC
- ALT         1     14.67 1618.5 173.22
- LDIST_L10   1     16.92 1620.8 173.28
- DIST_L10    1     69.18 1673.0 174.71
<none>                    1603.9 174.81
- GRAZE       1     78.00 1681.9 174.94
- YR.ISOL     1    126.48 1730.3 176.22
```

```
- AREA_L10   1    867.44 2471.3 192.26

Step:  AIC=173.22
ABUND ~ YR.ISOL + GRAZE + AREA_L10 + LDIST_L10 + DIST_L10

            Df Sum of Sq    RSS    AIC
- LDIST_L10  1     10.76 1629.3 171.52
<none>                   1618.5 173.22
- DIST_L10   1     85.56 1704.1 173.54
- GRAZE      1     98.23 1716.8 173.87
- YR.ISOL    1    117.80 1736.3 174.38
- AREA_L10   1   1088.05 2706.6 194.35

Step:  AIC=171.52
ABUND ~ YR.ISOL + GRAZE + AREA_L10 + DIST_L10

           Df Sum of Sq    RSS    AIC
<none>                  1629.3 171.52
- GRAZE     1     93.97 1723.3 172.04
- YR.ISOL   1    107.73 1737.0 172.40
```

```
- DIST_L10  1    114.60 1743.9 172.57
- AREA_L10  1   1161.66 2791.0 193.74
```

The selected model

```
summary(full_fit)
```

```
Call:
lm(formula = ABUND ~ ., data = loyn_train)

Residuals:
     Min       1Q   Median       3Q      Max
-17.3445  -3.4647   0.1991   2.8689  14.1844

Coefficients:
              Estimate Std. Error t value Pr(>|
t|)
(Intercept) -159.27533  109.13660  -1.459
0.1527
YR.ISOL        0.09334    0.05392   1.731
0.0916 .
GRAZE         -1.40912    1.03653  -1.359
```

```
summary(step_fit)
```

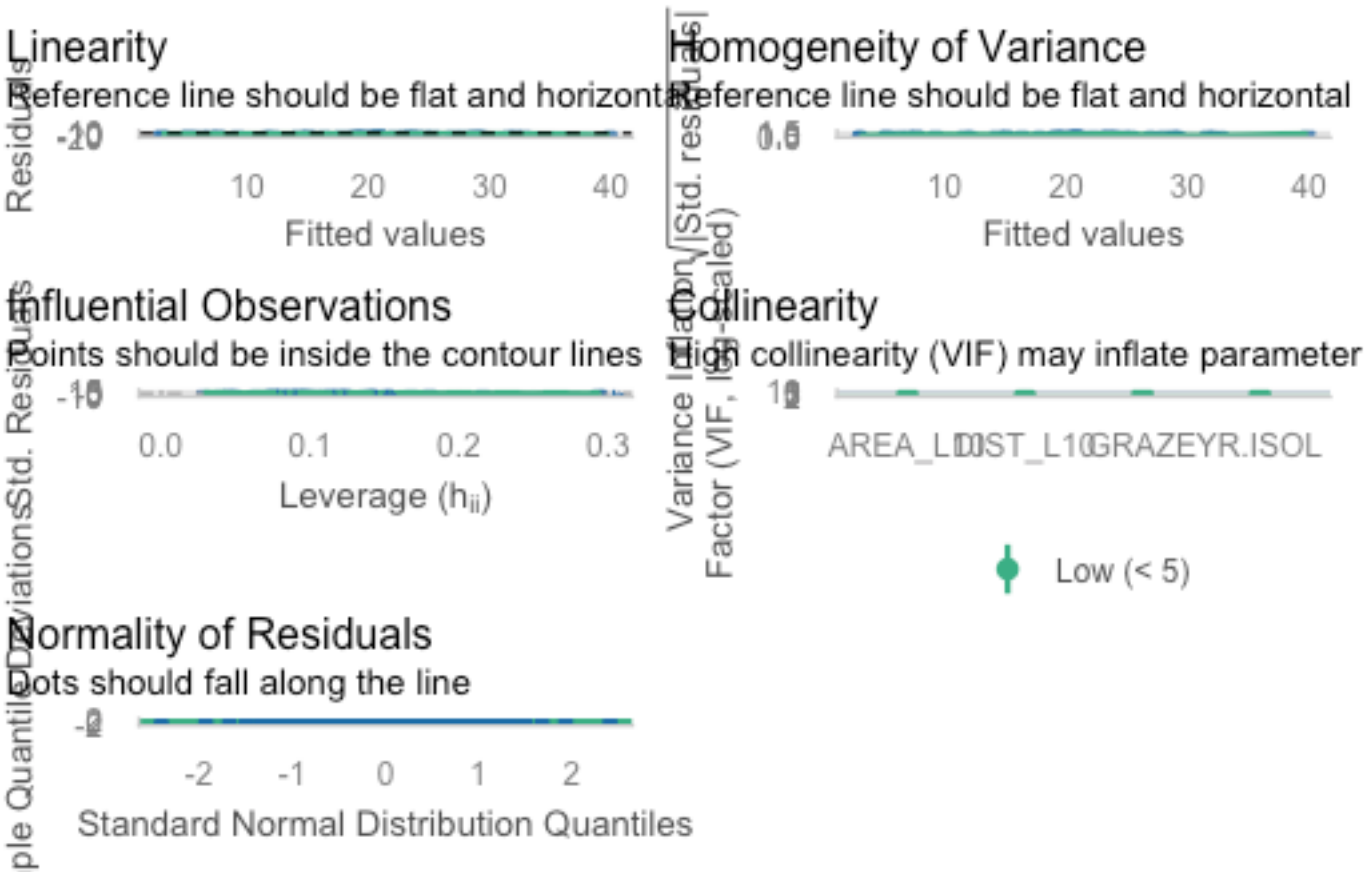
```
Call:
lm(formula = ABUND ~ YR.ISOL + GRAZE + AREA_L10
+ DIST_L10, data = loyn_train)

Residuals:
     Min       1Q   Median       3Q      Max
-18.1683  -3.1961   0.3374   3.4834  14.2021

Coefficients:
              Estimate Std. Error t value Pr(>|
t|)
(Intercept) -135.17508  102.72850  -1.316
0.196
YR.ISOL        0.08323    0.05118   1.626
0.112
```

```
0.1820
ALT            0.01657    0.02810   0.589
0.5590
AREA_L10       8.09629    1.78591   4.533
5.63e-05 ***
LDIST_L10      2.05115    3.23927   0.633
0.5304
DIST_L10      -6.18596    4.83189  -1.280
0.2082
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
'.' 0.1 ' ' 1

Residual standard error: 6.497 on 38 degrees of
freedom
Multiple R-squared:  0.6752,    Adjusted R-
squared:  0.6239
F-statistic: 13.17 on 6 and 38 DF,  p-value:
5.277e-08
```

```
GRAZE         -1.50496    0.99082  -1.519
0.137
AREA_L10       8.61888    1.61392   5.340
3.98e-06 ***
DIST_L10      -4.87726    2.90770  -1.677
0.101
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
'.' 0.1 ' ' 1

Residual standard error: 6.382 on 40 degrees of
freedom
Multiple R-squared:  0.6701,    Adjusted R-
squared:  0.6371
F-statistic: 20.31 on 4 and 40 DF,  p-value:
3.365e-09
```

Assumptions - Round 2

```
check_model(step_fit, check = c("linearity", "qq", "homogeneity", "outliers", "vif"))
```


Linearity
Reference line should be flat and horizontal
Residuals
Fitted values
Homogeneity of Variance
Reference line should be flat and horizontal
Fitted values
Influential Observations
Points should be inside the contour lines
Leverage
Collinearity
High collinearity (VIF) may inflate parameter
AREA_L
GRAZEYR.ISOL
Low (< 5)
Normality of Residuals
Dots should fall along the line
Standard Normal Distribution Quantiles

Model validation

It looks like the model is good, so let's bring in the test set to see how it performs!

Prepare the test data

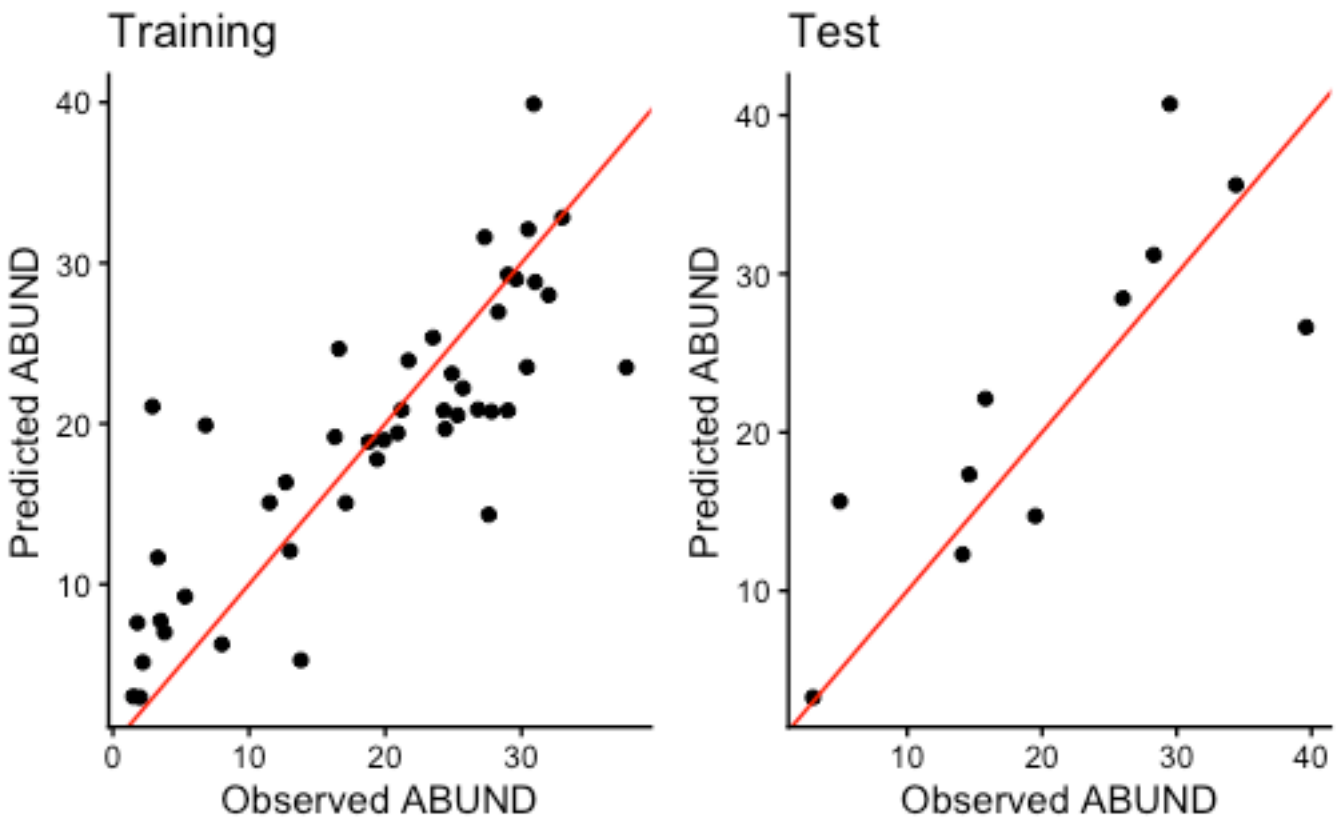
Since the test data has not been transformed, we need to do that first.

We then predict onto the training and test dataset using the reduced stepwise model.

```
loyn_test ← loyn_test %>%
    mutate(
        AREA_L10 = log10(AREA),
        LDIST_L10 = log10(LDIST),
        DIST_L10 = log10(DIST)
    ) %>%
    select(-AREA, -LDIST, -DIST)

loyn_train$pred ← predict(step_fit, newdata = loyn_train)
loyn_test$pred ← predict(step_fit, newdata = loyn_test)
```

Plotting observed vs predicted

```
p1 ← ggplot(loyn_train, aes(ABUND, pred)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "Observed ABUND", y = "Predicted ABUND",
       title = "Training") +
  theme_classic()

p2 ← ggplot(loyn_test, aes(ABUND, pred)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "Observed ABUND", y = "Predicted ABUND",
       title = "Test") +
  theme_classic()

p1 + p2
```


Training
Predicted ABUND
Observed ABUND
0
10
20
30
40
Test
Predicted ABUND
Observed ABUND
10
20
30
40

Calculating metrics - error

Generally the training dataset will have lower error and higher linearity than the test dataset. If this difference is very large – it suggests the model is not applicable to the test data and overfitting.

With error – lower is better. The following all measure error **in the same units as the response variable** (number of birds in each forest patch).

Mean error

```
mean(loyn_train$ABUND - loyn_train$pred) |> round(2)
```

```
[1] 0
```

```
mean(loyn_test$ABUND - loyn_test$pred) |> round(2)
```

```
[1] -1.65
```

We expect the ME for the training dataset to be near 0 – a well-fitted linear regression model will have positive and negative residuals balance each other out.

Mean absolute error

```
mean(abs(loyn_train$ABUND - loyn_train$pred)) |> round(2)
```

```
[1] 4.43
```

```
mean(abs(loyn_test$ABUND - loyn_test$pred)) |> round(2)
```

```
[1] 5.21
```

Root mean squared error (`caret` package)

```
RMSE(loyn_train$ABUND, loyn_train$pred) |> round(2)
```

```
[1] 6.02
```

```
RMSE(loyn_test$ABUND, loyn_test$pred) |> round(2)
```

```
[1] 6.72
```

Calculating metrics – linearity

With linearity – higher is better.

Both training and test datasets perform similarly, which is a good sign the model is not overfitting.

Pearson's correlation coefficient r

```
cor(loyn_train$ABUND, loyn_train$pred) |> round(2)
```

```
[1] 0.82
```

```
cor(loyn_test$ABUND, loyn_test$pred) |> round(2)
```

```
[1] 0.82
```

R^2

Can either square the correlation coefficient or use the `R2()` function from the `caret` package.

```r
R2(loyn_train$ABUND, loyn_train$pred) |> round(2)
```

```
[1] 0.67
```

```r
R2(loyn_test$ABUND, loyn_test$pred) |> round(2)
```

```
[1] 0.68
```

Lin's concordance correlation coefficient (CCC) (`epiR` package)

```r
epi.ccc(loyn_train$ABUND, loyn_train$pred)$rho.c$est |> round(2)
```

```
[1] 0.8
```

```r
epi.ccc(loyn_test$ABUND, loyn_test$pred)$rho.c$est |> round(2)
```

```
[1] 0.81
```

Conclusions

```
# put all data into a tible and kable it
tibble(
    Dataset = c("Training", "Test"),
    ME = c(
        mean(loyn_train$ABUND - loyn_train$pred),
        mean(loyn_test$ABUND - loyn_test$pred)
    ),
    MAE = c(
        mean(abs(loyn_train$ABUND - loyn_train$pred)),
        mean(abs(loyn_test$ABUND - loyn_test$pred))
    ),
    RMSE = c(
        RMSE(loyn_train$ABUND, loyn_train$pred),
        RMSE(loyn_test$ABUND, loyn_test$pred)
    ),
    cor = c(
        cor(loyn_train$ABUND, loyn_train$pred),
        cor(loyn_test$ABUND, loyn_test$pred)
    ),
```

```
    R2 = c(
        R2(loyn_train$ABUND, loyn_train$pred),
        R2(loyn_test$ABUND, loyn_test$pred)
    ),
    LCCC = c(
        epi.ccc(loyn_train$ABUND, loyn_train$pred)$rho.c$est,
        epi.ccc(loyn_test$ABUND, loyn_test$pred)$rho.c$est
    )
) %>%
    knitr::kable(digits = 2)
```

Dataset	ME	MAE	RMSE	cor	R2	LCCC
Training	0.00	4.43	6.02	0.82	0.67	0.80
Test	−1.65	5.21	6.72	0.82	0.68	0.81

- The model fit for the training dataset is marginally better (slight overfitting, but not a concern)
- Small differences are also expected due to the small sample size or the chosen `set.seed()`
- The model predicts bird abundance in forest patches in SE Victoria well (r = 0.82, **LCCC** = 0.81, **MAE** = 5.21 birds/patch, **RMSE** = 6.72 birds/patch).

Thanks!

Questions? Comments?

Slides made with Quarto