Applied Data Science

L6-7. Supervised Learning. Regression

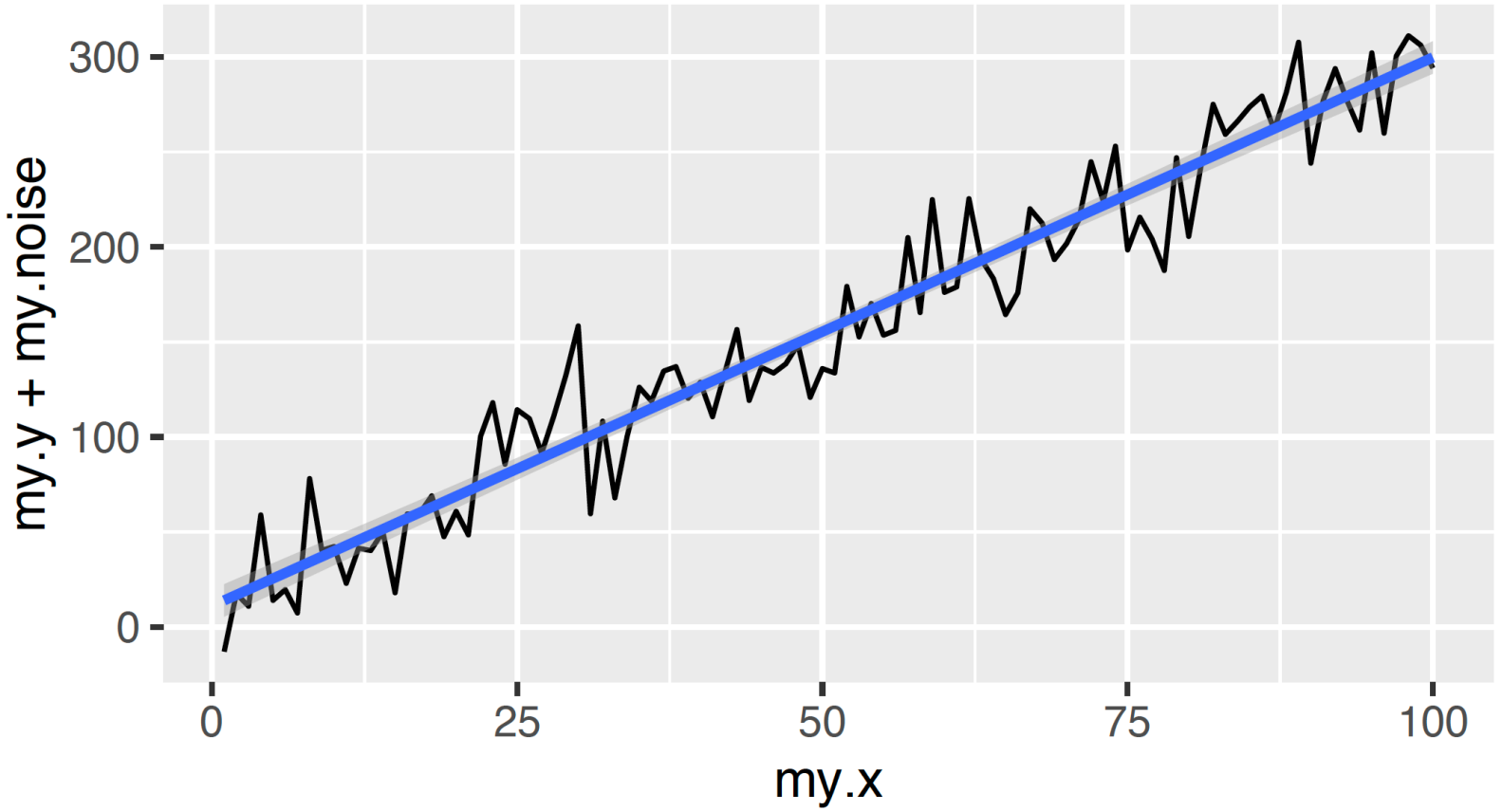
Irina Mohorianu

Head of Bioinformatics/ Scientific Computing @CSCI

Outline [Regression 1]

Linear regression. Definition

linear regression = a very simple approach for supervised learning.
We assume a linear dependence of Y on $X_1, X_2, ..., X_p$.



Linear regression. Definition

We assume the model

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where β_0 and β_1 are two unknown constants that represent the *intercept* and the *slope*; β_0 and β_1 are also known as *coefficients* or *parameters* and ϵ is the error term.

Given the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ for the model coefficients, we predict the output, $\hat{y}$ using

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x,$$

where $\hat{y}$ indicates the prediction of Y on the basis of $X = x$. The *hat* symbol denotes an estimated value.

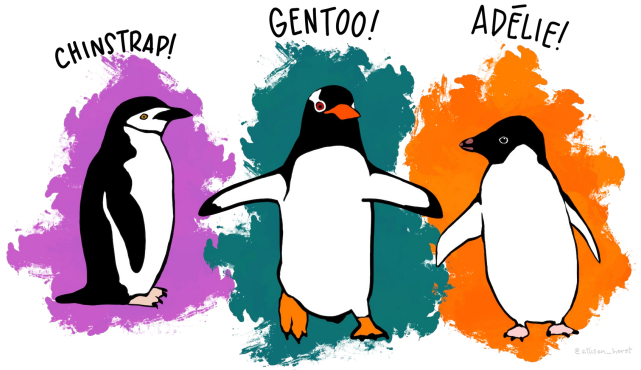
Linear regression. Definition

```
penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male

```
penguins.describe()
```

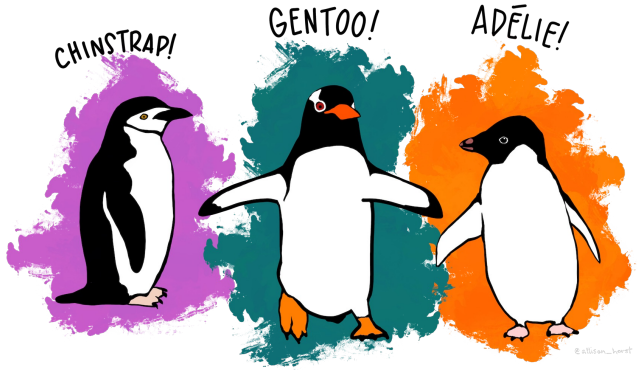
	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	333.000000	333.000000	333.000000	333.000000
mean	43.992793	17.164865	200.966967	4207.057057
std	5.468668	1.969235	14.015765	805.215802
min	32.100000	13.100000	172.000000	2700.000000
25%	39.500000	15.600000	190.000000	3550.000000
50%	44.500000	17.300000	197.000000	4050.000000
75%	48.600000	18.700000	213.000000	4775.000000
max	59.600000	21.500000	231.000000	6300.000000



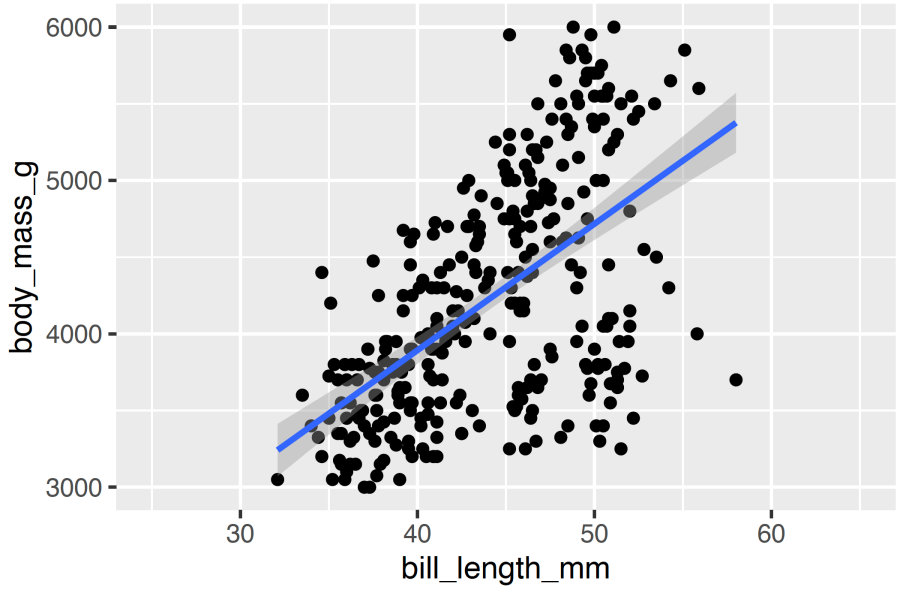
Linear regression. Why is regression useful?

In this lecture, we focus on some key aspects underlying linear regression models, loss functions and Bias/Variance decomposition.

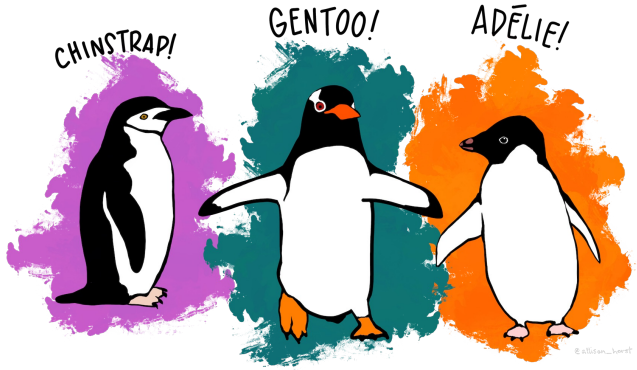
1. Is there a [linear] relationship between flipper length (feature) and body mass (response variable)?
2. How strong is the relationship between flipper length (feature) and body mass (response variable)?
3. Which feature (bill length, bill depth, flipper length) explains the body mass?
4. Can we [accurately] estimate the effect of each feature on the body mass?
5. How accurately can we predict future weights of penguins?
6. Is the relationship linear between the feature and the output variable?
7. Is there synergy among the features?

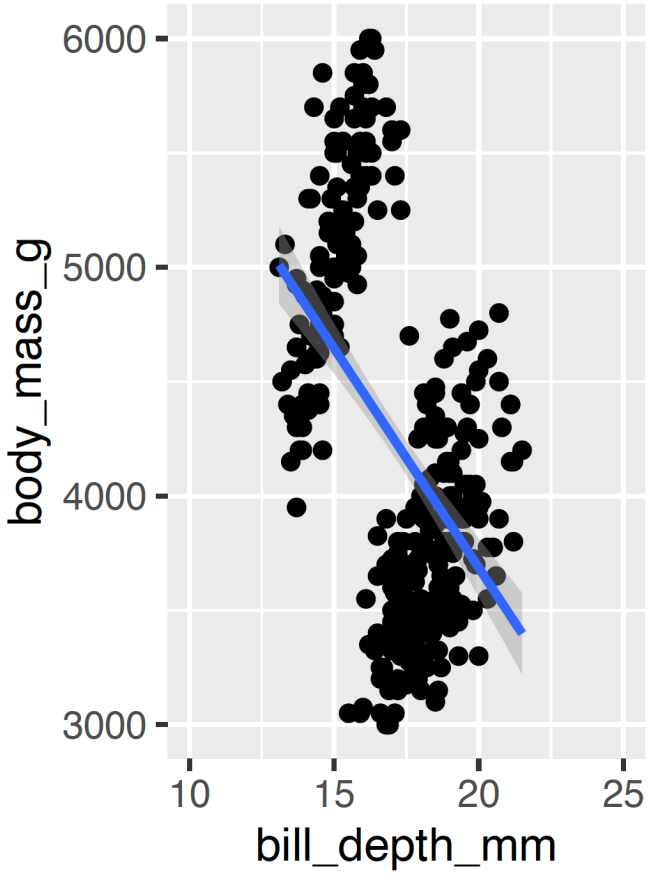


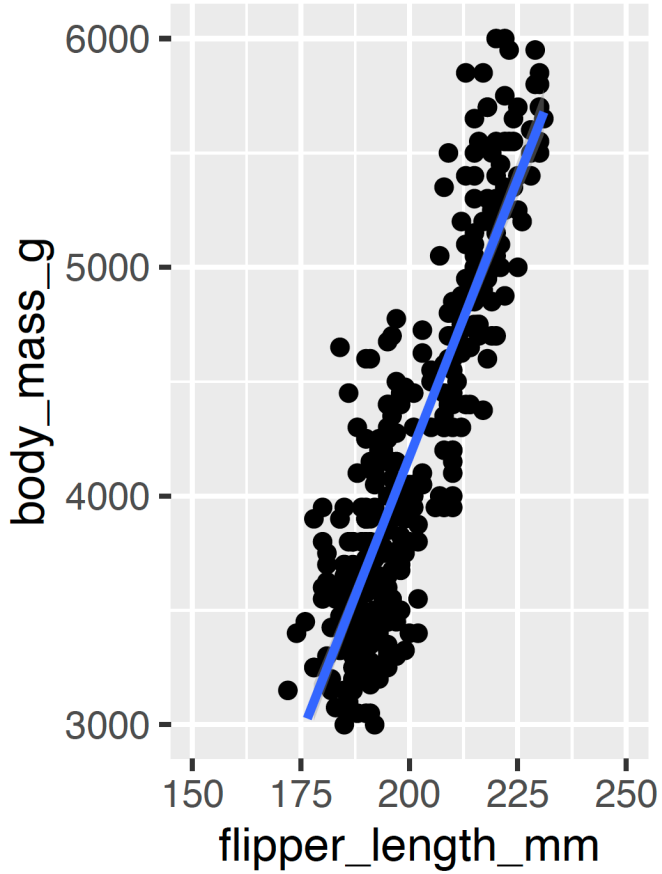
Linear regression. Dataset structure



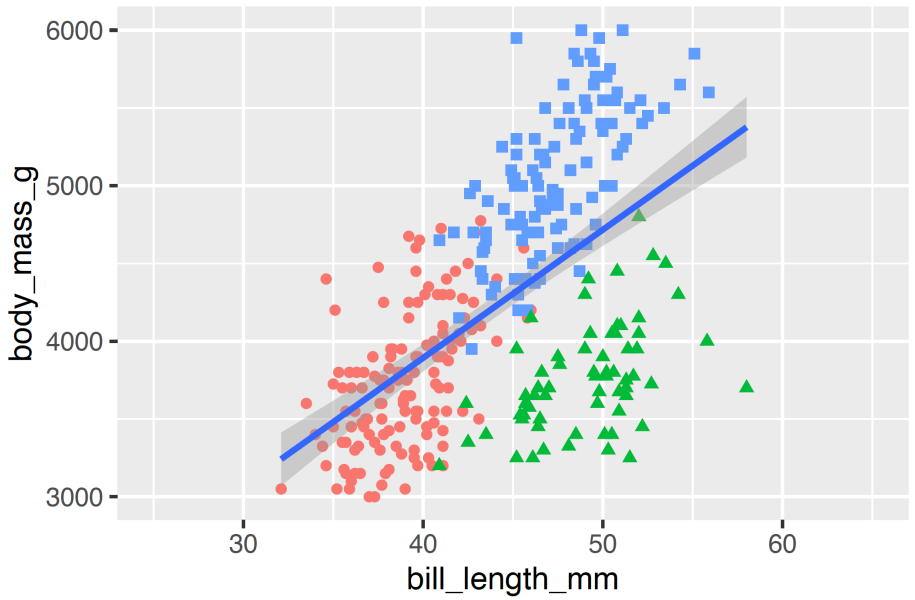
Is this linear relation meaningful?

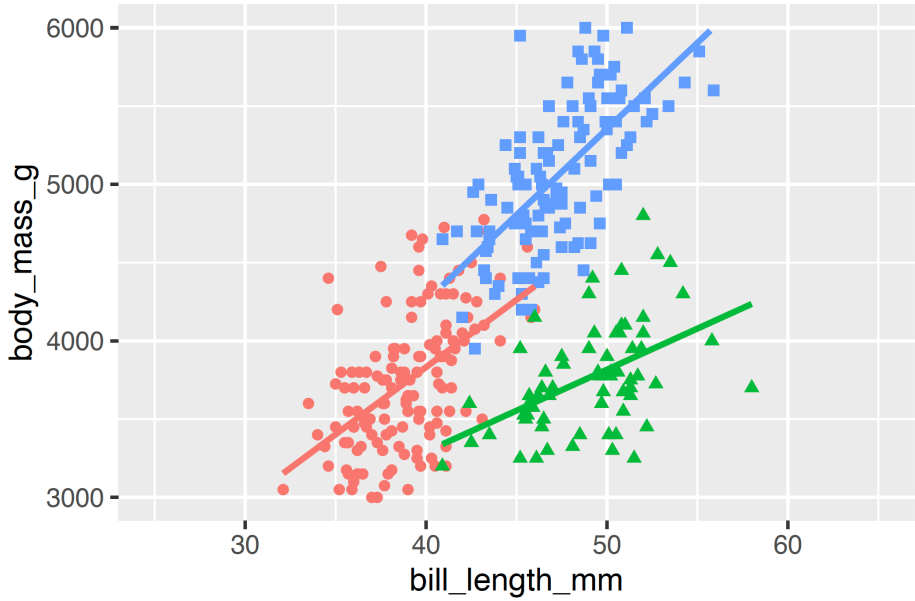




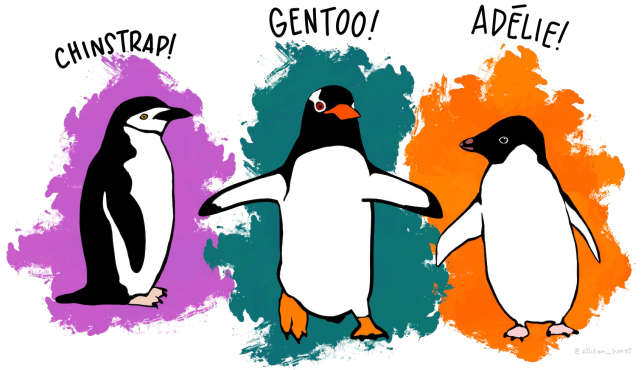


Linear regression. Dataset structure

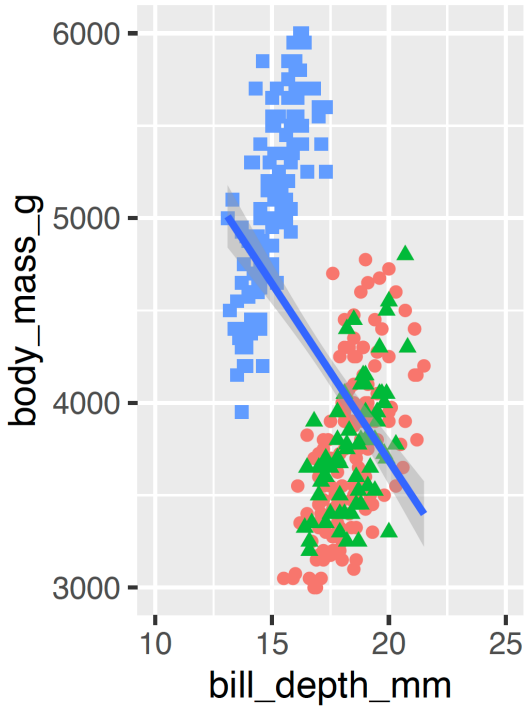


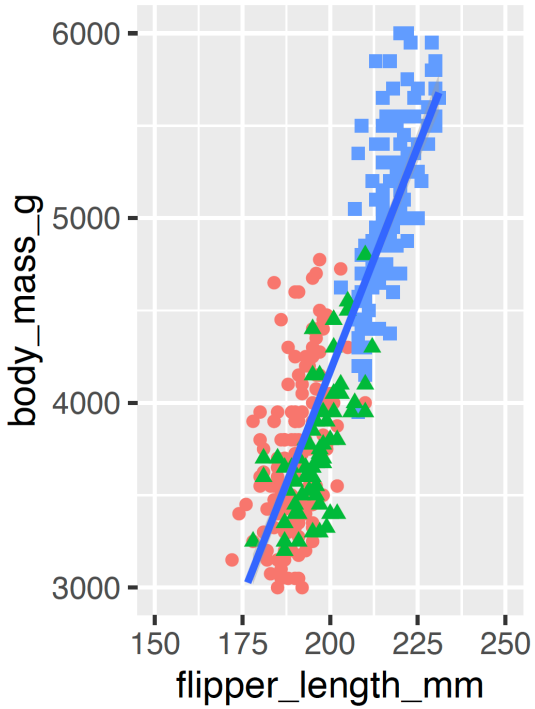


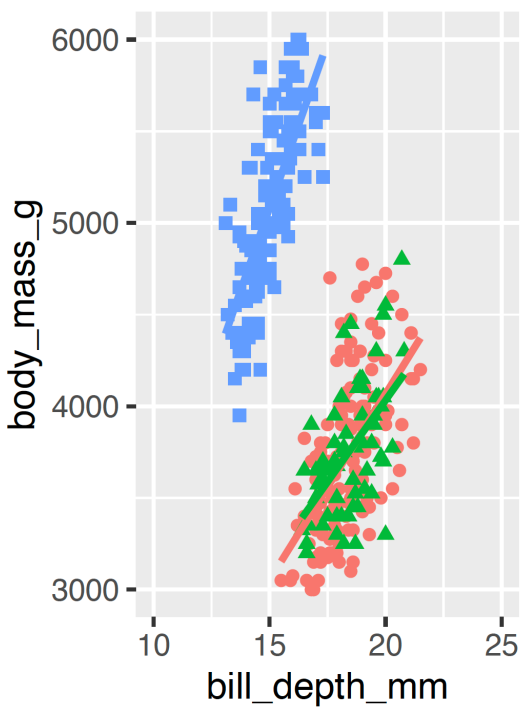
By emphasising the 3 types of penguins, we note there is an underlying structure in the data.
The second panel shows that the coefficients per class are close, but not matched.

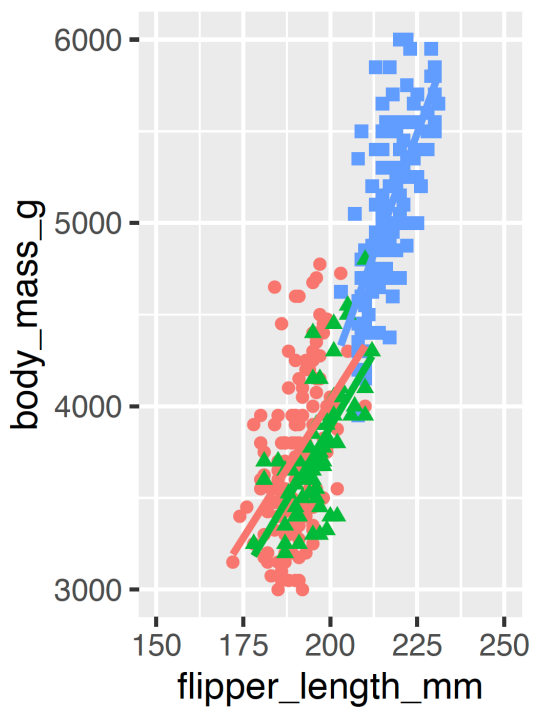


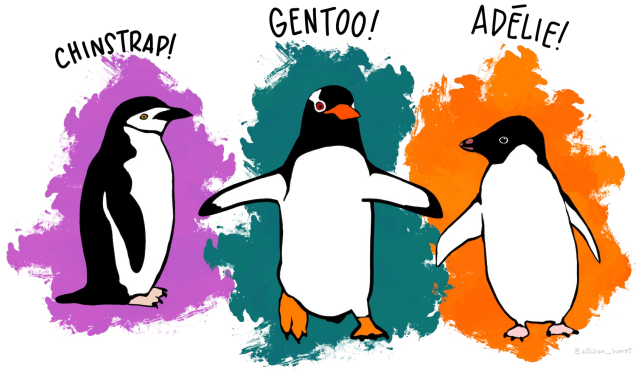
Linear regression. Dataset structure











Linear regression. One feature model

More formally:

$$body.mass \approx \beta_0 + \beta_1 \times bill.length + \epsilon$$

$$body.mass \approx \beta_0 + \beta_1 \times bill.depth + \epsilon$$

$$body.mass \approx \beta_0 + \beta_1 \times flipper.length + \epsilon$$

or with all predictors into one model:

$$body.mass \approx \beta_0 + \beta_1 \times bill.length + \beta_2 \times bill.depth + \beta_3 \times flipper.length + \epsilon$$

Linear regression. Residuals

Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$, be the prediction for Y based on the ith value of X.
Then $\epsilon_i = y_i - \hat{y}_i$; ϵ_i is the ith residual.

We define the *residual sum of squares (RSS)* as

$$RSS = \epsilon_1^2 + \epsilon_2^2 + ... + \epsilon_n^2$$

which is equivalent to

$$RSS = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + ... + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2$$

Linear regression. Estimates

The least squares approach chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimise the RSS. The minimising values are:

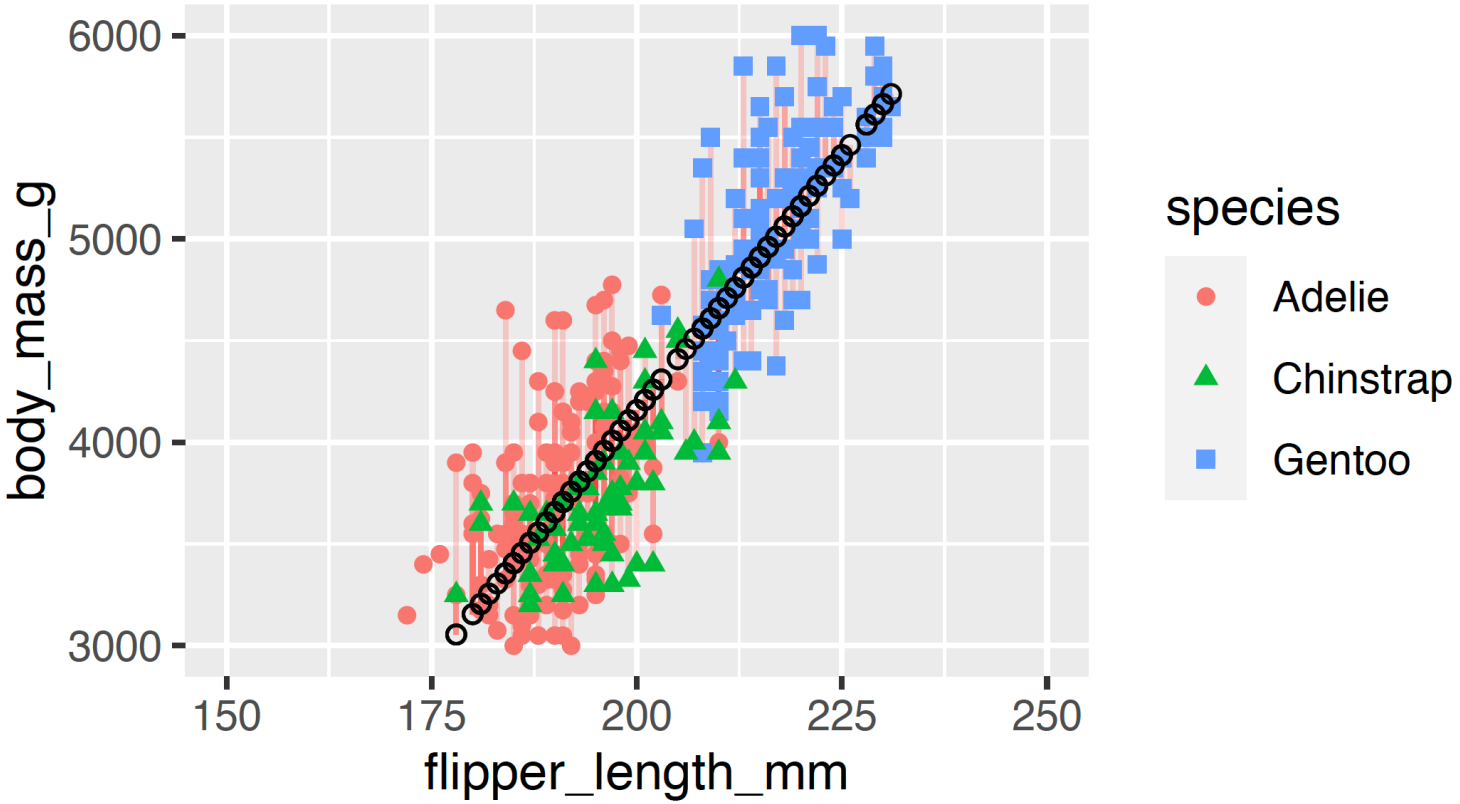
$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

and

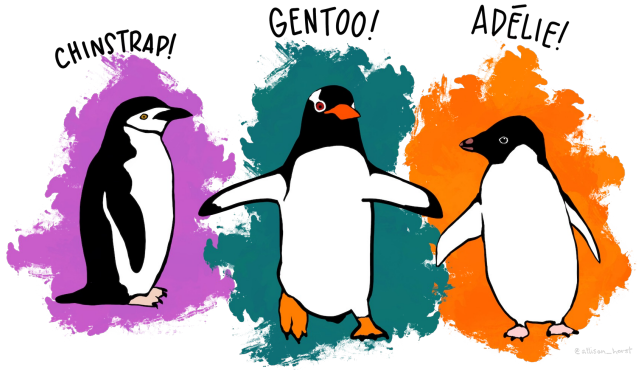
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means.

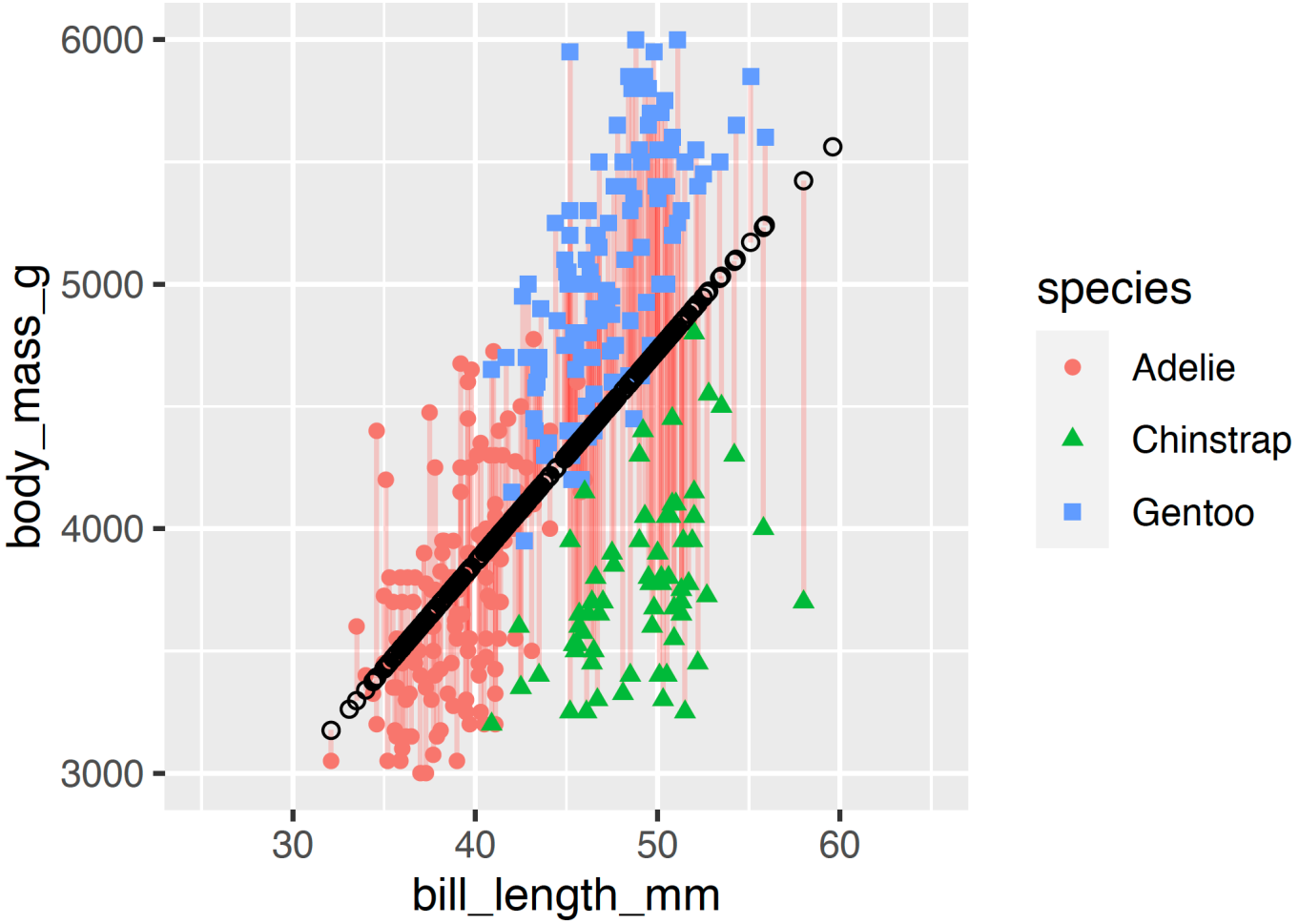
Linear regression. Flipper length

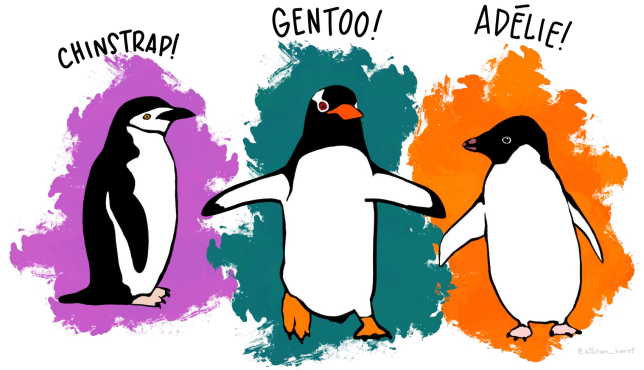


The least squares fit for the regression of body mass onto flipper length. For this predictor, the linear fit captures the essence of the relationship; however the errors are large for large values.

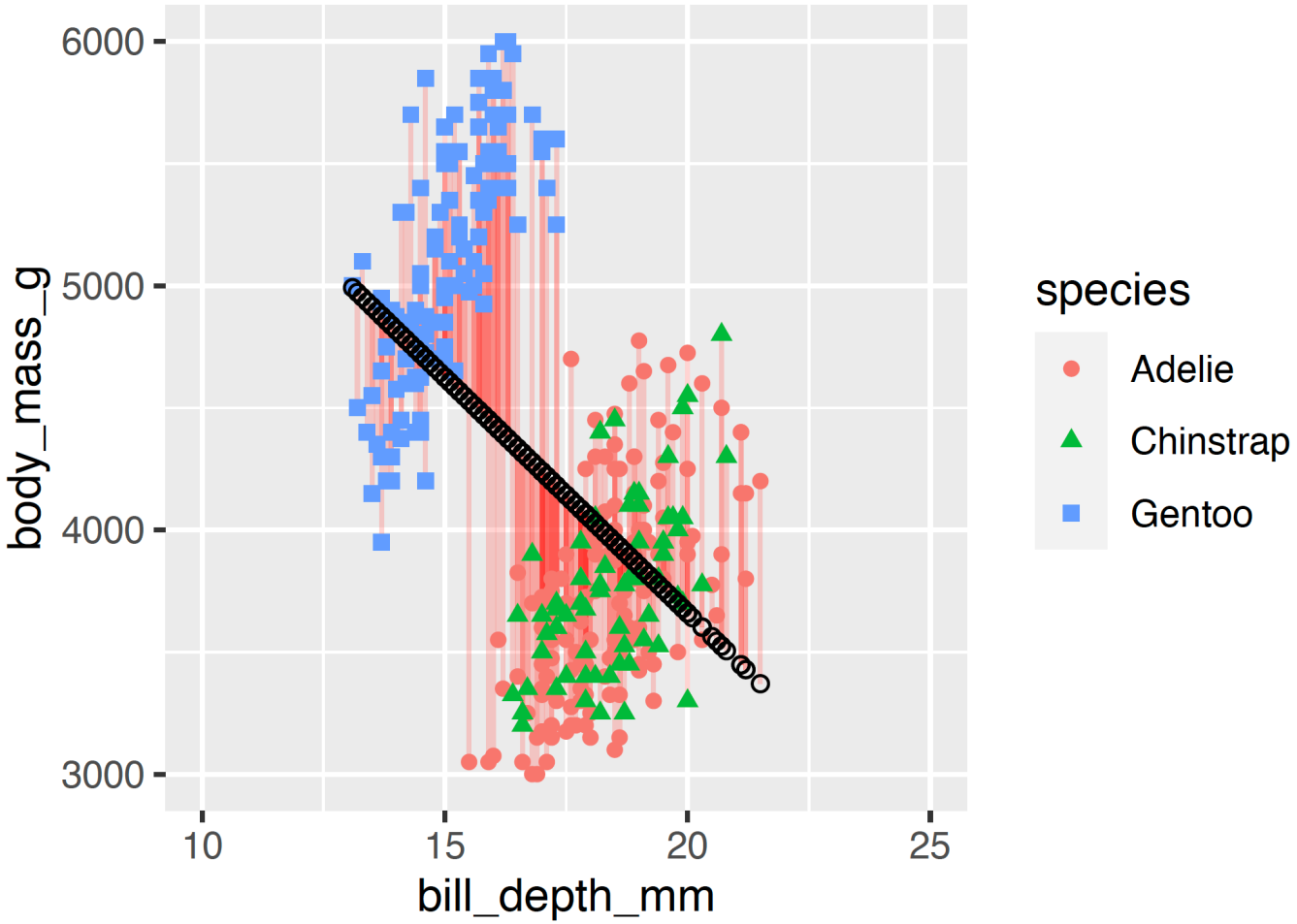


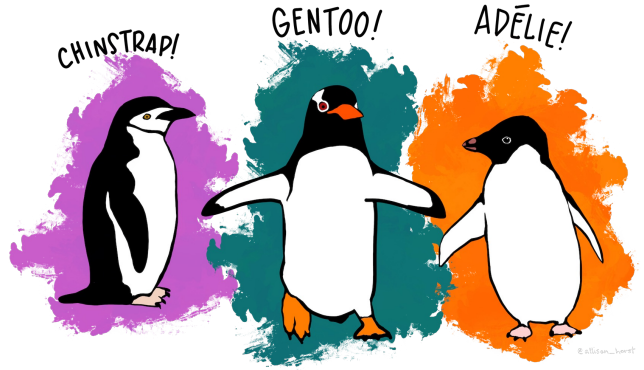
Linear regression. Bill length





Linear regression. Bill depth





Linear regression. Standard errors

The standard error of an estimator reflects how it varies under repeated subsampling.

$$SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

and

$$SE(\hat{\beta}_0)^2 = \sigma^2\{\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\}$$

where $\sigma^2 = Var(\epsilon)$.

The standard errors can be used to compute confidence intervals. A 95% CI is defined as the range of values such that with 95% probability the range will contain the true unknown value of the parameter.

$$\hat{\beta}_1 \pm 2 \cdot SE(\hat{\beta}_1).$$

Linear regression. Hypothesis testing

Standard errors can be used to perform *hypothesis testing* on the coefficients. The most common hypothesis test involves testing the *null hypothesis* of:

H_0 There is no relationship between X and Y

H_1 There is some relationship between X and Y

Or, more formally:

H_0 $\beta_1 = 0$

H_1 $\beta_1 \neq 0$

since if $\beta_1 = 0$ the model becomes $Y = \beta_0 + \epsilon$ and X is not associated with Y.

Linear regression. P values

To test the H_0 we compute a t-statistic:

$$t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)}$$

This will be a t-distribution with $n - 2$ degrees of freedom.

Using R, we can compute the probability of observing any value $\geq |t|$. We call this probability **p-value**.

Linear regression

```python
import statsmodels.api as sm
import numpy as np

X = penguins[['bill_length_mm']]
X = sm.add_constant(X)

y = penguins['body_mass_g']

# Fit linear regression model
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.347
Model:                            OLS   Adj. R-squared:                  0.345
Method:                 Least Squares   F-statistic:                     176.2
Date:                Wed, 18 Oct 2023   Prob (F-statistic):           1.54e-32
Time:                        12:04:31   Log-Likelihood:                -2629.1
No. Observations:                 333   AIC:                             5262.
Df Residuals:                     331   BIC:                             5270.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          388.8452    289.817      1.342      0.181    -181.271     958.961
bill_length_mm  86.7918      6.538     13.276      0.000      73.931      99.652
==============================================================================
Omnibus:                        6.141   Durbin-Watson:                   0.845
Prob(Omnibus):                  0.046   Jarque-Bera (JB):                4.899
Skew:                          -0.197   Prob(JB):                       0.0864
Kurtosis:                       2.555   Cond. No.                         360.
==============================================================================
```

Linear regression

```python
X = penguins[['bill_depth_mm']]
X = sm.add_constant(X)

y = penguins['body_mass_g']

model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.223
Model:                            OLS   Adj. R-squared:                  0.220
Method:                 Least Squares   F-statistic:                     94.89
Date:                Wed, 18 Oct 2023   Prob (F-statistic):           7.02e-20
Time:                        12:07:12   Log-Likelihood:                -2658.2
No. Observations:                 333   AIC:                             5320.
Df Residuals:                     331   BIC:                             5328.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          7519.9808    342.325     21.967      0.000    6846.574    8193.388
bill_depth_mm  -193.0061     19.814     -9.741      0.000    -231.983    -154.029
==============================================================================
Omnibus:                        8.001   Durbin-Watson:                   2.279
Prob(Omnibus):                  0.018   Jarque-Bera (JB):                6.116
Skew:                           0.225   Prob(JB):                       0.0470
Kurtosis:                       2.511   Cond. No.                         152.
==============================================================================
```

Linear regression

```
X = penguins[['flipper_length_mm']]
X = sm.add_constant(X)

y = penguins['body_mass_g']

model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.762
Model:                            OLS   Adj. R-squared:                  0.761
Method:                 Least Squares   F-statistic:                     1060.
Date:                Wed, 18 Oct 2023   Prob (F-statistic):          3.13e-105
Time:                        12:08:27   Log-Likelihood:                -2461.1
No. Observations:                 333   AIC:                             4926.
Df Residuals:                     331   BIC:                             4934.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -5872.0927    310.285    -18.925      0.000   -6482.472   -5261.713
flipper_length_mm    50.1533      1.540     32.562      0.000      47.123      53.183
==============================================================================
Omnibus:                        5.922   Durbin-Watson:                   2.102
Prob(Omnibus):                  0.052   Jarque-Bera (JB):                5.876
Skew:                           0.325   Prob(JB):                       0.0530
Kurtosis:                       3.025   Cond. No.                     2.90e+03
==============================================================================
```

Linear regression. Assessing accuracy

Residual standard error:

$$RSE = \sqrt{\frac{1}{n-2} RSS}$$

where the *residual sum of squares* is $RSS = \sum_{i=1}^{n} (y_i - \hat{y}_1)^2$

R^2 fraction of variance explained is

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

where $TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2$ is the total sum of squares.

Linear regression. Assessing accuracy

An R^2 statistic that is close to 1 indicates that a large proportion of variability is explained by the regression. Values close to 0 occur when the linear model is wrong or the error σ^2 is high.

The link between correlation:

$$Cor(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

and R^2 is $R^2 = Cor(X, Y)^2$.

Linear regression. Main points

- we discussed the coefficient estimation for single parameter regression
- we assessed the discriminative power for single paramter regression
- our aim is to identify "optimal" models using some form of incremental selection of predictors

Linear regression. Other loss functions
L1 loss function

Mean Absolute Error (also called **L1 loss**) is a simple yet robust loss function used for regression models.

Suitable for models dealing with variable that are not strictly Gaussian e.g. due to the presence of outliers. MAE is an ideal option since it does not take into account the direction of the outliers (unrealistically high positive or negative values).

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

Reasons why it is used in practice follow the lines of median/ MAD approaches discussed earlier.

Linear regression
Mean Bias Error

Mean Bias Error focuses on the average bias between models (i.e. the original and the predicted values). The Bias is related to overestimating or underestimating a parameter.

The Mean Bias Error measures the difference between the target and the predicted value (**not the absolute difference**); positive and the negative errors can cancel each other out, which is why it is one of the lesser-used loss functions.

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

Linear regression.
L2 loss

Mean Squared Error (also called **L2 loss**) was used for the previous examples.

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

Linear regression.
Loss functions

Comparing the L_1 vs L_2 errors we note:

L_1 loss is more robust to outliers.

if the difference between the predicted and the observed value is high, L_2 loss magnifies the effect.

L_1 loss is less stable than L_2 loss.

small noise-induced change can lead to significant changes in the regression line. Such an effect across multiple iterations may lead to a significant change in the slope between iterations.

L_2 (MSE) ensures the regression line moves lightly on small adjustments due to noise.

Linear regression
Huber loss

Huber Loss combines the robustness of L1 to outliers with the stability of L2 to noise-induced variations; i.e. for large errors, it is linear; for small errors, it is quadratic.

$$L_\delta == \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{if } |y - f(x)| < \delta. \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & \text{otherwise.} \end{cases} \tag{1}$$

Linear regression
Comparison of loss functions

```python
def calculate_huber_loss(residual, delta):
    if abs(residual) <= delta:
        return 0.5 * residual**2
    else:
        return delta * abs(residual) - 0.5 * delta**2

def get_loss_values_df(residuals):
    mae = np.mean(np.abs(residuals))
    mbe = np.mean(residuals)
    mse = np.mean(residuals**2)

    # Calculate delta using MADN approach
    delta = 1.4826 * np.median(np.abs(residuals - np.median(residuals)))
    huber = np.mean([calculate_huber_loss(r, delta) for r in residuals])

    return pd.DataFrame({
        'mae': [mae],
        'mbe': [mbe],
        'mse': [mse],
        'huber': [huber]
    })
```

```python
X = penguins[['flipper_length_mm']]
y = penguins['body_mass_g']
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
predicted_values = model.predict(X)
residuals = y - predicted_values

# Get loss values for different loss functions
loss_values = get_loss_values_df(residuals)
loss_values
```

	mae	mbe	mse	huber
0	312.084746	-1.287495e-11	153789.677867	63801.449193

- MAE: on average, the predictions are off by 312 units.
- MBE: the errors cancel out i.e the model doesn't have a systematic bias in its predictions; rephrased, it does not consistently overpredict or underpredict the target variable.
- MSE: larger errors are amplified; highest value across metrics; even a few significant deviations (outliers) can result in a high value.
- Huber loss bridges MAE and MSE, penalizing outliers on a linear function. Its value between MAE and MSE suggests there are outliers in the data.

Linear regression.
Comparison of loss functions

Fit linear regression models using different loss functions (MAE, MBE, Huber loss), calculating coefficients and residuals for each.

```python
import numpy as np
from scipy.optimize import minimize

X = penguins['flipper_length_mm'].values
y = penguins['body_mass_g'].values

def mae_loss(params):
    intercept, slope = params
    predictions = intercept + slope * X
    return np.mean(np.abs(y - predictions))

def calculate_huber_loss(residual, delta=1.35):
    if abs(residual) <= delta:
        return 0.5 * residual**2
    else:
        return delta * abs(residual) - 0.5 * delta**2

def huber_loss_function(params):
    intercept, slope = params
    predictions = intercept + slope * X
    residuals = y - predictions
    delta = 1.4826 * np.median(np.abs(residuals - np.median(residuals)))
    return np.mean([calculate_huber_loss(r, delta) for r in residuals])

def calculate_residuals(params):
    intercept, slope = params
    predictions = intercept + slope * X
    return y - predictions
```

```python
# Initial guess for intercept and slope
initial_guess = [0, 0]

# Fit linear regression model using MAE loss
result_mae = minimize(mae_loss, initial_guess)
coefficients_mae = result_mae.x
residuals_mae = calculate_residuals(coefficients_mae)

# Fit linear regression model using Huber loss
result_huber = minimize(huber_loss_function, initial_guess)
coefficients_huber = result_huber.x
residuals_huber = calculate_residuals(coefficients_huber)
```

Linear regression.
MAE loss

```
# Get loss values for model fitted with MAE
loss_values_MAE = get_loss_values_df(residuals_mae)
loss_values_MAE
```

	mae	mbe	mse	huber
0	468.259282	52.45026	326753.880377	143132.160546

- Optimized to minimize the average absolute error.
- The MBE is positive, i.e. the model tends to underpredict the target variable, though the value is relatively small.
- MSE and Huber loss values suggest some presence of outliers

Linear regression.
Huber loss

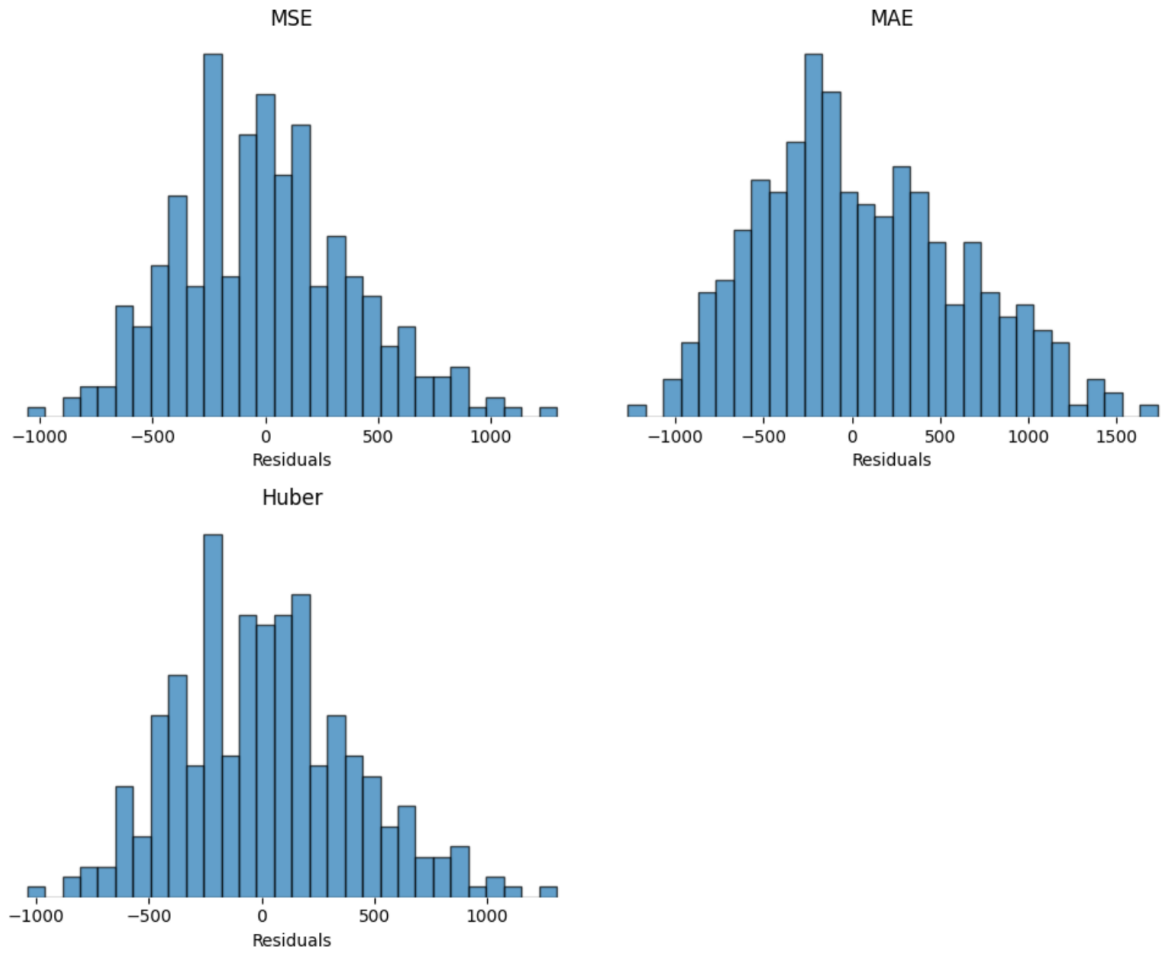
```
# Get loss values for model fitted with Huber loss
loss_values_huber = get_loss_values_df(residuals_huber)
loss_values_huber
```

	mae	mbe	mse	huber
0	311.342372	17.711967	154107.992294	63599.419906

- The Huber loss balances MAE (L1) and MSE (L2). It inherits the robustness of MAE against outliers, while also benefiting from the stability of MSE on small residuals.
- The results of the Huber model are as expected a balance between the results of the MAE and MSE models.

Linear regression.
Comparison loss functions. Residuals

```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize=(10, 8))
axes[1, 1].axis('off')

# Function to remove borders/spines
def remove_spines(ax):
    for spine in ax.spines.values():
        spine.set_visible(False)
    ax.tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)

# Plot histograms of residuals
def plot_hist(ax, residuals, title):
    ax.hist(residuals, bins=30, edgecolor='black', alpha=0.7)
    ax.set_title(title)
    ax.set_xlabel("Residuals")
    remove_spines(ax)

plot_hist(axes[0, 0], residuals, "MSE")
plot_hist(axes[0, 1], residuals_mae, "MAE")
plot_hist(axes[1, 0], residuals_huber, "Huber")

plt.tight_layout()
plt.show()
```



Linear regression.
Comparison of loss functions

MSE loss:

- Approximately normal distribution centered around 0, with values from -1000 to 1000, indicating consistent predictions.

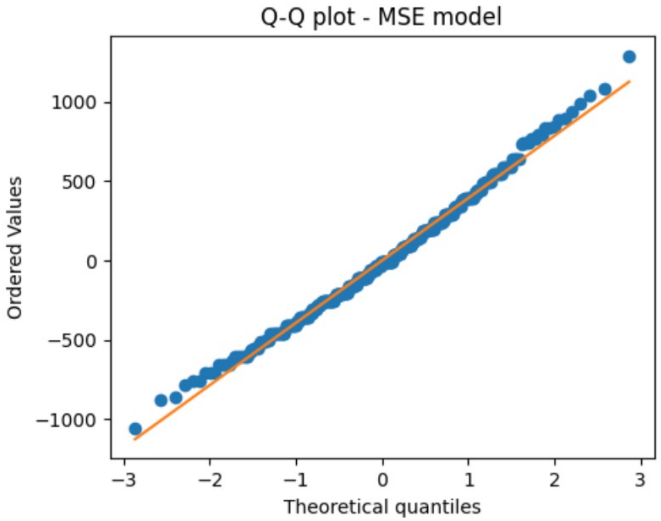
MAE loss:

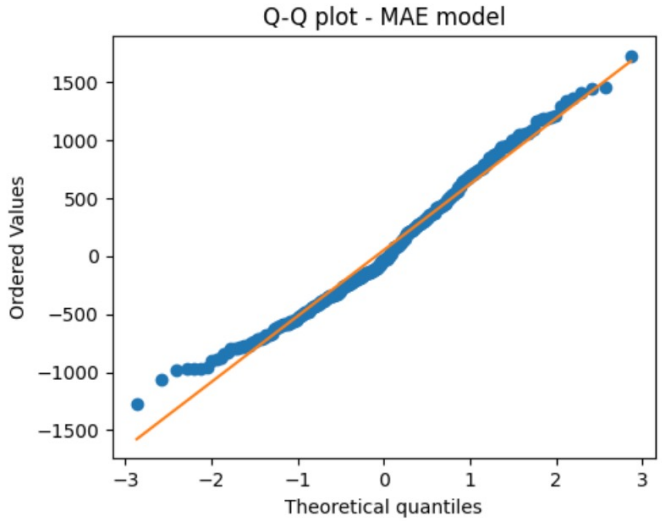
- Approximately normal with mean around 0. we note an extended tail with values ranging up to 1500, although these occur with low frequency (presence of outliers).

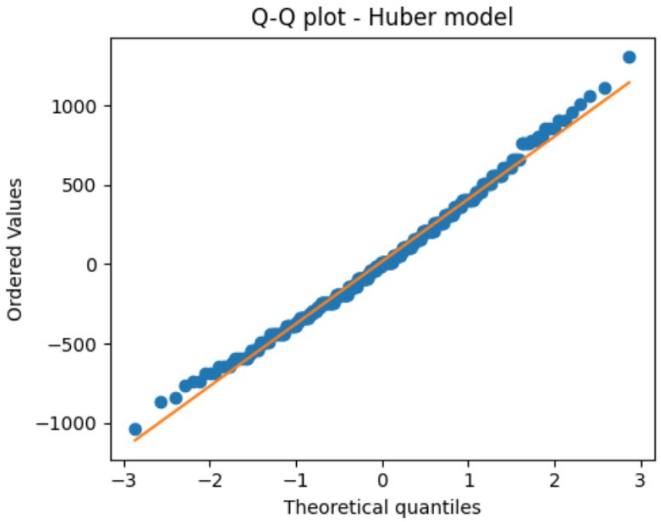
Huber loss:

- Approximately normal, centered around 0, values range from -1000 to 1200.

Linear regression. Homoscedasticity of residuals







The Q-Q plots for MSE, MAE, and Huber support that the residuals are approximately normally distributed; these models adhere to the homoscedasticity assumption.

Linear regression. Bias/Variance tradeoff

Sources of error in ML: bias and variance

Bias: the model's error rate on the training set (rephrased, the difference between the average prediction and the correct value we are predicting).

A model with high bias is oversimplified (insufficient information acquired from the training data).

Variance: the model's error rate on the validation (or test) set, in addition to the bias

A model with high variance captures the signal and the noise in the training data and fails to generalise well on (unseen) test data.

Linear regression.
Bias/variance decomposition

On a model M built on an output variable y, and X predictors

$$y = f(X) + \epsilon$$

Where ϵ is the error term, expected to be normally distributed with mean 0.

The expected error on Y is

$$Err(Y) = E[(Y - f(\hat{X}))^2]$$

The $Err(Y)$ can be further decomposed as:

$$Err(Y) = E[f(\hat{X}) - f(X)]^2 + E[(f(\hat{X}) - E[f(\hat{X})])^2] + \sigma_\epsilon^2$$

$Err(Y)$ is the sum of squared bias, variance and irreducible error.

Linear regression

"Essentially all models are wrong, but some are useful".

— George Box

"The only way to find out what will happen when a complex system is disturbed is to disturb the system, not merely to observe it passively."

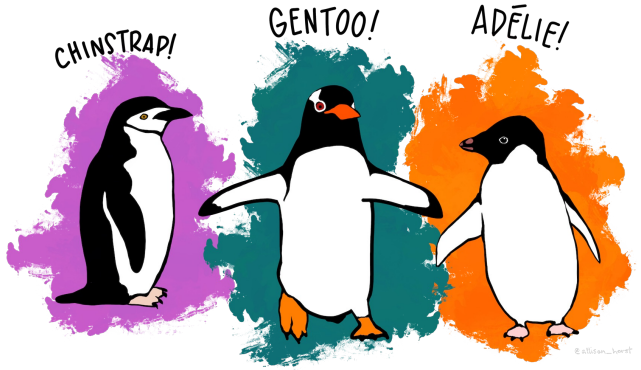
— Fred Mosteller and John Tukey, paraphrasing George Box

Outline [Regression 2]

Linear regression.
Multiple regression

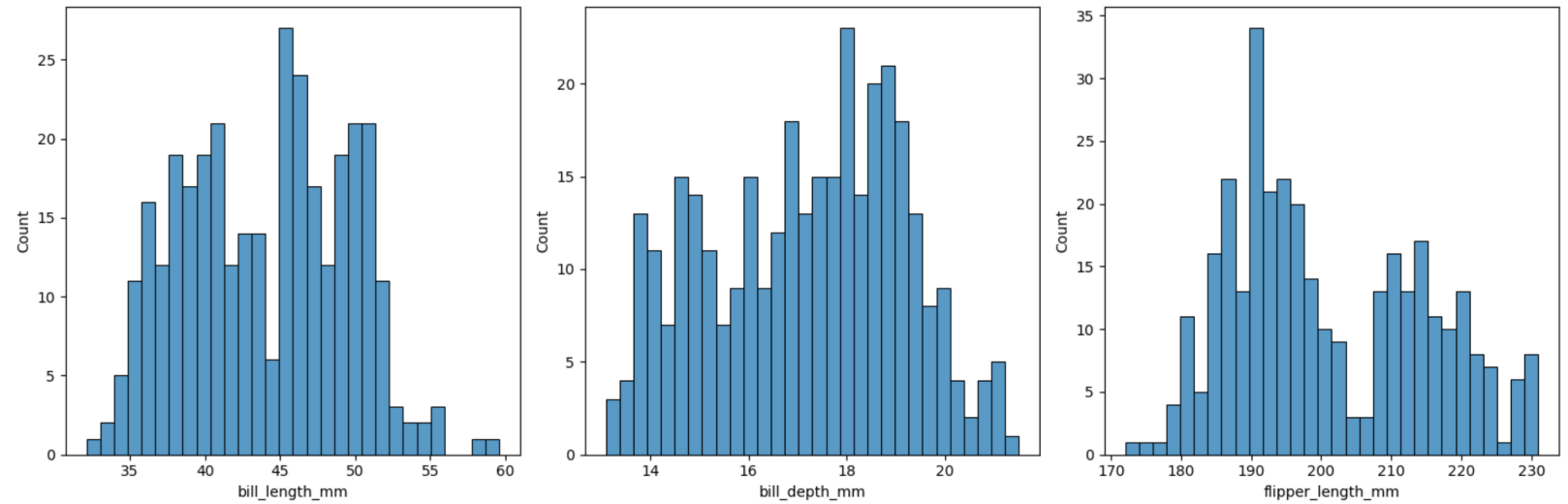
```
X = penguins[['flipper_length_mm', 'bill_length_mm', 'bill_depth_mm']]
y = penguins['body_mass_g']
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
print(model.summary())
```

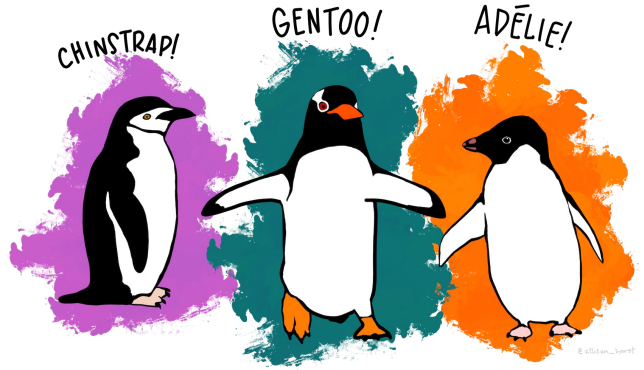
```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.764
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     354.9
Date:                Wed, 18 Oct 2023   Prob (F-statistic):          9.26e-103
Time:                        12:51:01   Log-Likelihood:                -2459.8
No. Observations:                 333   AIC:                             4928.
Df Residuals:                     329   BIC:                             4943.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -6445.4760    566.130    -11.385      0.000   -7559.167   -5331.785
flipper_length_mm    50.7621      2.497     20.327      0.000      45.850      55.675
bill_length_mm        3.2929      5.366      0.614      0.540      -7.263      13.849
bill_depth_mm        17.8364     13.826      1.290      0.198      -9.362      45.035
==============================================================================
Omnibus:                        5.596   Durbin-Watson:                   1.968
Prob(Omnibus):                  0.061   Jarque-Bera (JB):                5.469
Skew:                           0.312   Prob(JB):                       0.0649
Kurtosis:                       3.068   Cond. No.                     5.44e+03
==============================================================================
```



Linear regression. Pre-processing tests

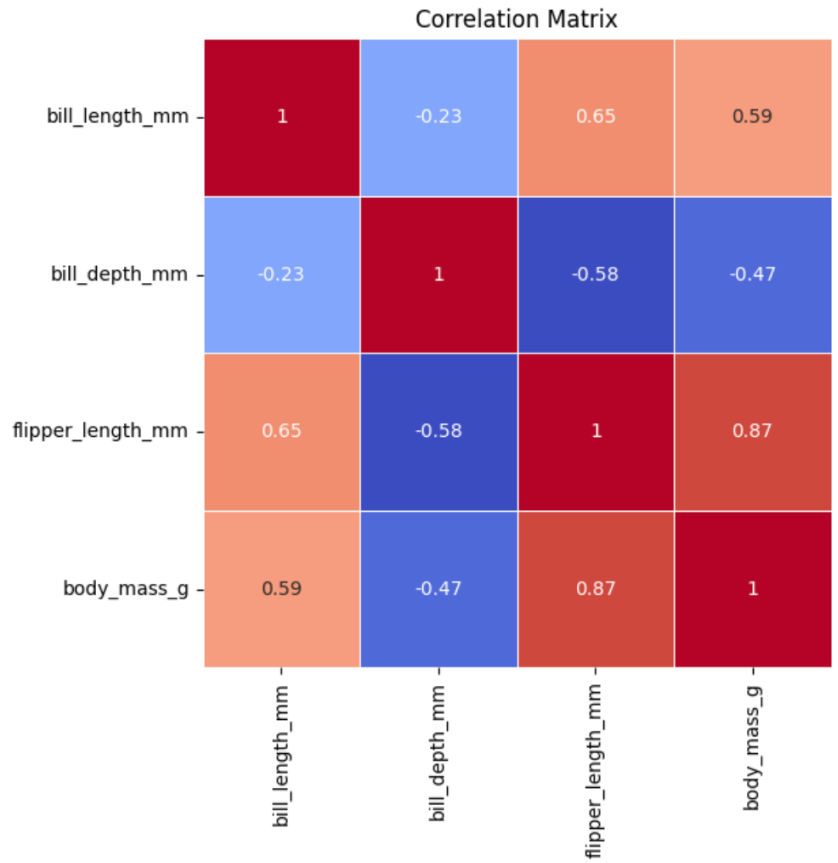
```
import seaborn as sns

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

sns.histplot(data=penguins, x='bill_length_mm', ax=axes[0], bins=30)
sns.histplot(data=penguins, x='bill_depth_mm', ax=axes[1], bins=30)
sns.histplot(data=penguins, x='flipper_length_mm', ax=axes[2], bins=30)

plt.tight_layout()
plt.show()
```

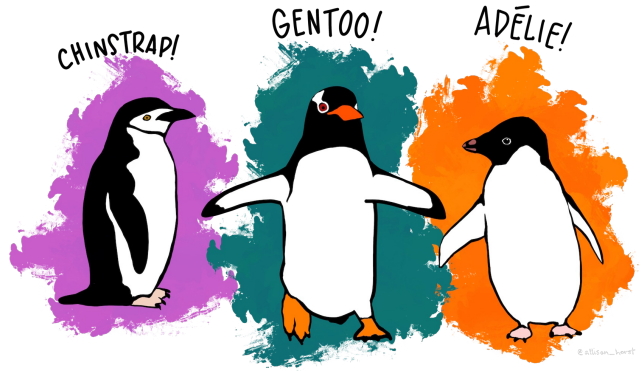




Linear regression
Assessment of colinearity

```
cor_matrix = penguins.iloc[:, 2:6].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(cor_matrix, annot=True, cmap='coolwarm', cbar=False, square=True, linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```





Linear regression.
Questions related to multiple predictors

1. Is at least one of the predictors $X_1, X_2, ..., X_p$ useful in predicting the response?
2. Do all the predictors help to explain Y, or is only a subset of the predictors useful?
3. How well does the model fit the data?
4. Given a set of predictor values, what response value should we predict, and how accurate is our prediction?

Linear regression
Important variables

- The brute force approach is called "all subsets" or best subsets regression i.e. we compute the least squares fit for all possible subsets and then choose between them based on some criteria that balances training error with model size.
- However we often can't examine all possible models, since they are 2^p of them; for example when p $=$ 40 there are over a trillion models! Instead we need an automated approach that searches through a subset of them.

Linear regression.
Forward selection

- Begin with the *null model* - a model that contains an intercept but no predictors
- Fit p simple linear regressions and add to the null model the variable that results in the lowest RSS.
- Add to that model the variable that results in the lowest RSS amongst all two-variable models.
- Continue until some stopping rule is satisfied, for example when all remaining variables have a p-value above some threshold

Linear regression
Forward selection

```
X_FL = sm.add_constant(penguins['flipper_length_mm'])
model_FL = sm.OLS(penguins['body_mass_g'], X_FL).fit()

X_BD = sm.add_constant(penguins['bill_depth_mm'])
model_BD = sm.OLS(penguins['body_mass_g'], X_BD).fit()

X_BL = sm.add_constant(penguins['bill_length_mm'])
model_BL = sm.OLS(penguins['body_mass_g'], X_BL).fit()
```

```
model_FL
Residual standard error: 310.285 on 331.0 degrees of freedom
Residual Sum of Squares (RSS): 51211962.730
Multiple R-squared: 0.762, Adjusted R-squared: 0.761
F-statistic: 1060.295 on 1 and 331.0 DF, p-value: < 2.2e-16

model_BD
Residual standard error: 342.325 on 331.0 degrees of freedom
Residual Sum of Squares (RSS): 167300074.220
Multiple R-squared: 0.223, Adjusted R-squared: 0.220
F-statistic: 94.887 on 1 and 331.0 DF, p-value: < 2.2e-16

model_BL
Residual standard error: 289.817 on 331.0 degrees of freedom
Residual Sum of Squares (RSS): 140467132.888
Multiple R-squared: 0.347, Adjusted R-squared: 0.345
F-statistic: 176.243 on 1 and 331.0 DF, p-value: < 2.2e-16
```

Linear regression
Forward selection

```
X_FL_BD = sm.add_constant(penguins[['flipper_length_mm', 'bill_depth_mm']])
model_FL_BD = sm.OLS(penguins['body_mass_g'], X_FL_BD).fit()

X_FL_BL = sm.add_constant(penguins[['flipper_length_mm', 'bill_length_mm']])
model_FL_BL = sm.OLS(penguins['body_mass_g'], X_FL_BL).fit()

summarise_lm(model_FL_BD, "model_FL_BD")
summarise_lm(model_FL_BL, "model_FL_BL")
```

```
model_FL_BD
Residual standard error: 545.348 on 330.0 degrees of freedom
Residual Sum of Squares (RSS): 50873075.493
Multiple R-squared: 0.764, Adjusted R-squared: 0.762
F-statistic: 533.166 on 1 and 330.0 DF, p-value: < 2.2e-16

model_FL_BL
Residual standard error: 312.604 on 330.0 degrees of freedom
Residual Sum of Squares (RSS): 51071962.945
Multiple R-squared: 0.763, Adjusted R-squared: 0.761
F-statistic: 530.447 on 1 and 330.0 DF, p-value: < 2.2e-16
```

Linear regression
Forward selection

```
X_FL_BD_BL = sm.add_constant(penguins[['flipper_length_mm', 'bill_depth_mm', 'bill_length_mm']])
model_FL_BD_BL = sm.OLS(penguins['body_mass_g'], X_FL_BD_BL).fit()

summarise_lm(model_FL_BD_BL, "model_FL_BD_BL")
```

```
model_FL_BD_BL
Residual standard error: 566.130 on 329.0 degrees of freedom
Residual Sum of Squares (RSS): 50814911.804
Multiple R-squared: 0.764, Adjusted R-squared: 0.762
F-statistic: 354.898 on 1 and 329.0 DF, p-value: < 2.2e-16
```

model.FL.BD.BL has a similar RSS compared to model.FL.BD, and comparable R^2 i.e. we are not explaining more signal.

We also note that the difference in R^2 between model.FL.BD and model.FL is minimal.

Following the lines of keep-it-simple, the preferred model is model.FL

Linear regression
Backward selection

- Start with all variables in the model; fit the model
- Remove the variable with the largest p-value i.e. the variable that is the least statistically significant
- The new $p - 1$-variable model is fit, and the variable with the largest p-value is removed.
- Continue until a stopping rule is reached e.g. when all remaining variables have a significant p-value above some threshold

Linear regression
Backward selection

```
X_FL_BD_BL = sm.add_constant(penguins[['flipper_length_mm', 'bill_depth_mm', 'bill_length_mm']])
model_FL_BD_BL = sm.OLS(penguins['body_mass_g'], X_FL_BD_BL).fit()
print(model_FL_BD_BL.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.764
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     354.9
Date:                Wed, 18 Oct 2023   Prob (F-statistic):          9.26e-103
Time:                        13:19:12   Log-Likelihood:                -2459.8
No. Observations:                 333   AIC:                             4928.
Df Residuals:                     329   BIC:                             4943.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -6445.4760    566.130    -11.385      0.000   -7559.167   -5331.785
flipper_length_mm    50.7621      2.497     20.327      0.000      45.850      55.675
bill_depth_mm        17.8364     13.826      1.290      0.198      -9.362      45.035
bill_length_mm        3.2929      5.366      0.614      0.540      -7.263      13.849
==============================================================================
Omnibus:                        5.596   Durbin-Watson:                   1.968
Prob(Omnibus):                  0.061   Jarque-Bera (JB):                5.469
Skew:                           0.312   Prob(JB):                       0.0649
Kurtosis:                       3.068   Cond. No.                     5.44e+03
==============================================================================
```

The bill_length_mm has the largest p-value. i.e. it will be excluded

Linear regression
Backward selection

```
X_FL_BD = sm.add_constant(penguins[['flipper_length_mm', 'bill_depth_mm']])
model_FL_BD = sm.OLS(penguins['body_mass_g'], X_FL_BD).fit()
print(model_FL_BD.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.764
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     533.2
Date:                Wed, 18 Oct 2023   Prob (F-statistic):          4.28e-104
Time:                        13:21:44   Log-Likelihood:                -2460.0
No. Observations:                 333   AIC:                             4926.
Df Residuals:                     330   BIC:                             4937.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -6537.5978    545.348    -11.988      0.000   -7610.395   -5464.800
flipper_length_mm    51.7670      1.884     27.481      0.000      48.061      55.473
bill_depth_mm        19.8780     13.407      1.483      0.139      -6.496      46.252
==============================================================================
Omnibus:                        5.766   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.056   Jarque-Bera (JB):                5.643
Skew:                           0.317   Prob(JB):                       0.0595
Kurtosis:                       3.070   Cond. No.                     5.13e+03
==============================================================================
```

The bill_depth_mm has the largest p-value. i.e. it will be excluded

Linear regression. Backward selection

```
X_FL = sm.add_constant(penguins[['flipper_length_mm']])
model_FL = sm.OLS(penguins['body_mass_g'], X_FL).fit()
print(model_FL.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.762
Model:                            OLS   Adj. R-squared:                  0.761
Method:                 Least Squares   F-statistic:                     1060.
Date:                Wed, 18 Oct 2023   Prob (F-statistic):          3.13e-105
Time:                        13:23:32   Log-Likelihood:                -2461.1
No. Observations:                 333   AIC:                             4926.
Df Residuals:                     331   BIC:                             4934.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -5872.0927    310.285    -18.925      0.000   -6482.472   -5261.713
flipper_length_mm    50.1533      1.540     32.562      0.000      47.123      53.183
==============================================================================
Omnibus:                        5.922   Durbin-Watson:                   2.102
Prob(Omnibus):                  0.052   Jarque-Bera (JB):                5.876
Skew:                           0.325   Prob(JB):                       0.0530
Kurtosis:                       3.025   Cond. No.                     2.90e+03
==============================================================================
```

The remaining feature is significant – this is the final model

Linear regression. Mixed selection

- this is a combination of forward and backward selection
- start with no variables in the model; add the variable that provides the best fit
- as more predictors are added, the p-values of the predictors already included will start to fluctuate; exclude the ones that rise above an *a priori* defined threshold
- continue performing the forward and backward steps until all variables in the model have sufficiently low p-values
- **Backward selection cannot be performed if $p > n$**
- **Forward selection can always be used; however it is a greedy approach (remedied by the mixed selection)**

Linear regression.
Models with categorical variables

Since the island feature has more than 2 categories, we use one-hot encoding to transform each category into a new binary column, making it suitable for regression modeling.

```
# Perform one-hot encoding on the island feature
island_encoded_df = pd.get_dummies(penguins['island'], drop_first=True)

# Replace island variable with the dummy variables introduced
penguins_enc = pd.concat([penguins.drop('island', axis=1), island_encoded_df], axis=1)
```

Linear regression
Models with categorical variables

The species output variable has multiple [3] classes. We choose one class (e.g. “Adelie”) and fit a binary logistic regression model. For a holistic analysis, one would typically fit models for each species and compare the results.

By adopting the One-Versus-All strategy, we can better discern the relationships and influences of predictors on each specific species.

```
# Create binary target variable for Adelie species
penguins_enc['adelie_only'] = (penguins_enc['species'] == 'Adelie').astype(int)
penguins_enc['sex'] = (penguins_enc['sex'] == 'Male').astype(int)

# Fit logistic regression model
X = penguins_enc[['Dream', 'Torgersen', 'sex']]
X = sm.add_constant(X)
y = penguins_enc['adelie_only']

model_adelie = sm.Logit(y, X).fit()
```

Linear regression
Models with categorical variables

```
print(model_adelie.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:            adelie_only   No. Observations:                  333
Model:                          Logit   Df Residuals:                      329
Method:                           MLE   Df Model:                            3
Date:                Wed, 18 Oct 2023   Pseudo R-squ.:                  0.2131
Time:                        13:48:08   Log-Likelihood:                -179.63
converged:                      False   LL-Null:                       -228.29
Covariance Type:            nonrobust   LLR p-value:                 5.868e-21
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.9908      0.218     -4.538      0.000      -1.419      -0.563
Dream          0.7827      0.253      3.093      0.002       0.287       1.279
Torgersen     23.4933   1.12e+04      0.002      0.998   -2.19e+04     2.2e+04
sex           -0.0082      0.253     -0.033      0.974      -0.504       0.488
==============================================================================
```

Non-parametric models. Decision trees Brief contrast

Decision trees (DT) are non-parametric i.e. no assumptions about the underlying distribution of the data are made.

The data space is partioned into regions; the partitions are determined based on the data itself.

A major advantage of decision trees is that they are interpretable.

We will partition the data into a training set and a testing set. The training set will be used to build the decision tree model, while the testing set will be used to evaluate its performance.

Non-parametric models. Decision trees Brief contrast

```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets using stratified sampling based on the species column
penguins_train, penguins_test = train_test_split(penguins, test_size=0.2, stratify=penguins['species'], random_state=42)
```

Why do we like seed 42? https://medium.com/@leticia.b/the-story-of-seed-42-874953452b94

Non-parametric models. Decision trees Brief contrast

To ensure the robustness of our model, we chose to use 10-fold cross-validation with 10 repeats.

```
island_encoded_df = pd.get_dummies(penguins_train[['island', 'sex']], drop_first=True)
penguins_train = pd.concat([penguins_train.drop(['island', 'sex'], axis=1), island_encoded_df], axis=1)

# Separate features and target variable
X = penguins_train.drop('species', axis=1)
y = penguins_train['species']

preprocessing = Pipeline([
    ('scaling', StandardScaler()),
    ('remove_nzv', VarianceThreshold())
])

# Decision tree model
model = DecisionTreeClassifier(random_state=42)
pipeline = Pipeline(steps=[('preprocess', preprocessing), ('model', model)])
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state=42)

# Evaluate the model using cross-validation
scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

Non-parametric models. Decision trees Brief contrast

```
scores
```

```
array([1.        , 1.        , 0.96296296, 0.96296296, 0.96296296,
       0.96296296, 0.96153846, 1.        , 0.96153846, 0.96153846,
       1.        , 0.92592593, 1.        , 1.        , 0.96296296,
       0.96296296, 0.92307692, 0.96153846, 1.        , 1.        ,
       1.        , 0.96296296, 1.        , 1.        , 0.92592593,
       0.96296296, 0.92307692, 0.92307692, 1.        , 0.96153846,
       1.        , 0.96296296, 0.96296296, 0.96296296, 0.96296296,
       1.        , 0.96153846, 0.96153846, 0.96153846, 1.        ,
       1.        , 0.96296296, 1.        , 0.96296296, 1.        ,
       1.        , 0.96153846, 0.96153846, 1.        , 0.84615385,
       1.        , 0.92592593, 0.92592593, 0.96296296, 1.        ,
       1.        , 1.        , 0.96153846, 1.        , 0.96153846,
       0.92592593, 0.96296296, 1.        , 1.        , 1.        ,
       0.92592593, 0.96153846, 0.96153846, 0.96153846, 1.        ,
       0.88888889, 1.        , 1.        , 0.96296296, 0.96296296,
       0.92592593, 0.96153846, 1.        , 1.        , 1.        ,
       0.96296296, 1.        , 0.92592593, 0.88888889, 1.        ,
       1.        , 1.        , 0.88461538, 1.        , 1.        ,
       0.96296296, 1.        , 0.96296296, 1.        , 0.96296296,
       0.88888889, 0.92307692, 0.92307692, 0.96153846, 1.        ])
```

The next step is to optimise the complexity parameter (cp).

Cp imposes a penalty to trees having too many splits.

The higher the cp the smaller the tree is.

Complexity (cp) controls the size of the decision tree and determines how much the overall performance metric (e.g. accuracy, Gini index, or cross-entropy) must decrease in order for a split to be attempted.

Non-parametric models. Decision trees Brief contrast

cp is a measure of the minimal improvement that must be achieved by a new split in the decision tree.

- A small cp: the tree will grow larger since small improvements in the performance metric will encourage additional splits.
- A large cp: will result in a small tree i.e. a large improvement in the performance metric will be needed to justify a new split.

```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'model__ccp_alpha': np.linspace(0, 0.05, 20)
}

# Set up GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid, scoring='accuracy', cv=cv, n_jobs=-1)

grid_search.fit(X, y)

best_ccp_alpha = grid_search.best_params_['model__ccp_alpha']
print(f"Best ccp_alpha: {best_ccp_alpha}")

# Evaluate the best model
best_score = grid_search.best_score_
print(f"Best cross-validation score (accuracy): {best_score:.3f}")
```

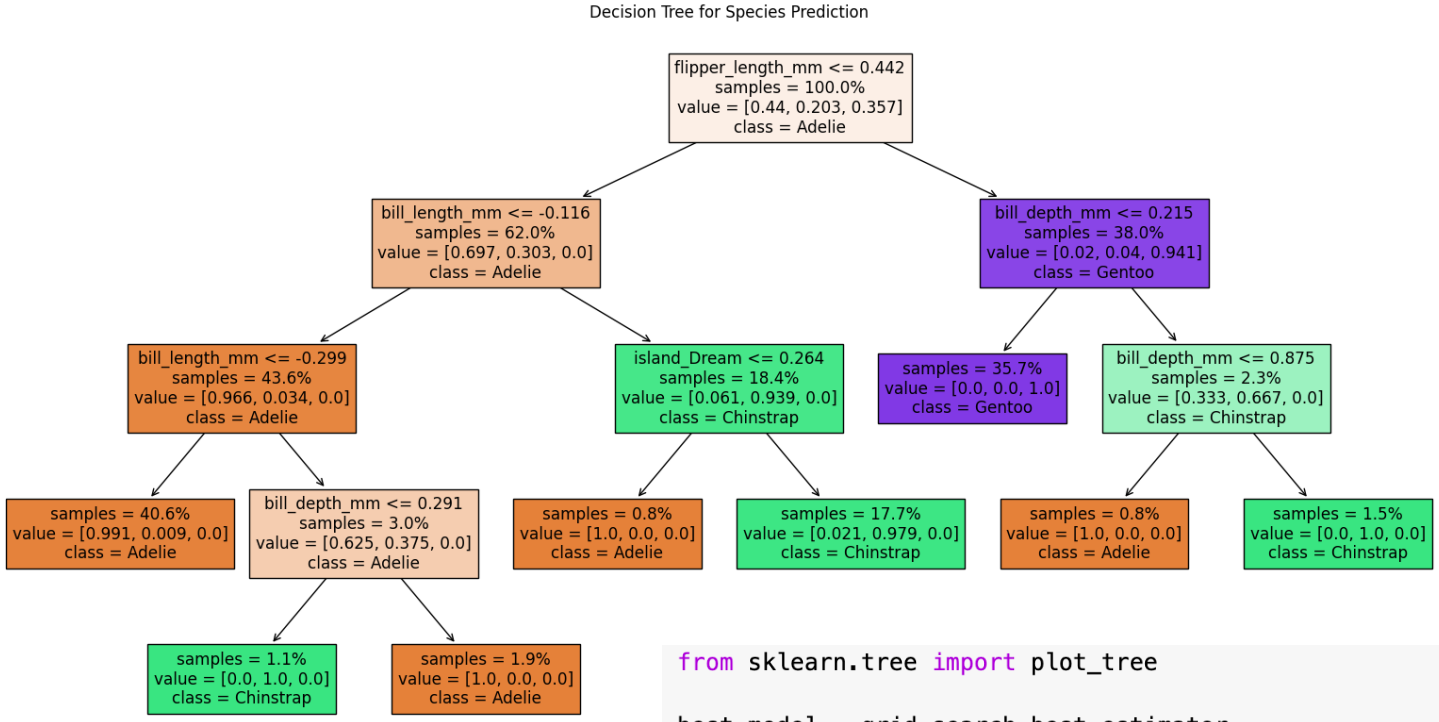
```
Best ccp_alpha: 0.005263157894736842
Best cross-validation score (accuracy): 0.970
```

Non-parametric models. Decision trees Brief contrast

```python
# Get average cross-validation scores for each ccp_alpha value
mean_scores = grid_search.cv_results_['mean_test_score']
ccp_alphas = grid_search.cv_results_['param_model__ccp_alpha'].data

# Display the results
for ccp_alpha, score in zip(ccp_alphas, mean_scores):
    print(f"ccp_alpha: {ccp_alpha:.5f} - Average Cross-Validation Score: {score:.3f}")
```

```
ccp_alpha: 0.00000 - Average Cross-Validation Score: 0.970
ccp_alpha: 0.00208 - Average Cross-Validation Score: 0.970
ccp_alpha: 0.00417 - Average Cross-Validation Score: 0.970
ccp_alpha: 0.00625 - Average Cross-Validation Score: 0.969
ccp_alpha: 0.00833 - Average Cross-Validation Score: 0.957
ccp_alpha: 0.01042 - Average Cross-Validation Score: 0.957
ccp_alpha: 0.01250 - Average Cross-Validation Score: 0.958
ccp_alpha: 0.01458 - Average Cross-Validation Score: 0.958
ccp_alpha: 0.01667 - Average Cross-Validation Score: 0.959
ccp_alpha: 0.01875 - Average Cross-Validation Score: 0.959
ccp_alpha: 0.02083 - Average Cross-Validation Score: 0.958
ccp_alpha: 0.02292 - Average Cross-Validation Score: 0.957
ccp_alpha: 0.02500 - Average Cross-Validation Score: 0.956
ccp_alpha: 0.02708 - Average Cross-Validation Score: 0.954
ccp_alpha: 0.02917 - Average Cross-Validation Score: 0.953
ccp_alpha: 0.03125 - Average Cross-Validation Score: 0.943
ccp_alpha: 0.03333 - Average Cross-Validation Score: 0.943
ccp_alpha: 0.03542 - Average Cross-Validation Score: 0.943
ccp_alpha: 0.03750 - Average Cross-Validation Score: 0.940
ccp_alpha: 0.03958 - Average Cross-Validation Score: 0.940
ccp_alpha: 0.04167 - Average Cross-Validation Score: 0.940
ccp_alpha: 0.04375 - Average Cross-Validation Score: 0.940
ccp_alpha: 0.04583 - Average Cross-Validation Score: 0.940
ccp_alpha: 0.04792 - Average Cross-Validation Score: 0.940
ccp_alpha: 0.05000 - Average Cross-Validation Score: 0.940
```

Non-parametric models. Decision trees Brief contrast




```python
from sklearn.tree import plot_tree

best_model = grid_search.best_estimator_

# Get the decision tree from the best model (pipeline)
decision_tree = best_model.named_steps['model']

plt.figure(figsize=(20,10))
plot_tree(decision_tree, filled=True, feature_names=X.columns, class_names=decision_tree.classes_, proportion=True, impurity=False)
plt.title("Decision Tree for Species Prediction")
plt.show()
```

Non-parametric models. Decision trees Brief contrast

```
# Predict labels on the test set and evaluate performance.
predicted_labels = best_model.predict(penguins_test)
```

It is essential to evaluate our model on unseen data.

While accuracy is a common metric, it is crucial to also consider other metrics such as specificity and sensitivity to get a better understanding of the model performance.

Always remember: **Never look at only accuracy.**

Linear regression

```python
from sklearn.metrics import confusion_matrix, classification_report
# Get the confusion matrix
cm = confusion_matrix(test_true, predicted_labels)
print("Confusion Matrix:")
print(cm)

# Get classification report for sensitivity (recall) and other metrics
report = classification_report(test_true, predicted_labels, output_dict=True)
report
```

```
Confusion Matrix:
[[28  1  0]
 [ 0 14  0]
 [ 2  0 22]]
{'Adelie': {'precision': 0.9333333333333333,
  'recall': 0.9655172413793104,
  'f1-score': 0.9491525423728815,
  'support': 29},
 'Chinstrap': {'precision': 0.9333333333333333,
  'recall': 1.0,
  'f1-score': 0.9655172413793104,
  'support': 14},
 'Gentoo': {'precision': 1.0,
  'recall': 0.9166666666666666,
  'f1-score': 0.9565217391304348,
  'support': 24},
 'accuracy': 0.9552238805970149,
```

Performance analysis: The model achieved very high accuracy ~95%.

This performance is comparable to what we observed on the validation set i.e. the model has high stability and generalises well.

Adelie Penguins

- Sensitivity: 96.55% of actual Adelie penguins were correctly identified.
- Specificity: 93.33% of non-Adelie instances were correctly classified.
- Misclassifications: Two Adelie penguins were predicted as Gentoo.

Chinstrap Penguins

- Sensitivity: 100% of actual Chinstrap penguins were correctly identified.
- Specificity: 93.33% of non-Chinstrap instances were correctly classified.

Gentoo Penguins

- Sensitivity: 100% of actual Adelie penguins were correctly identified.
- Specificity: 91.66% of non-Adelie instances were correctly classified.
- Misclassifications: One Gentoo penguins was predicted as Chinstrap.

Linear regression. Regularisation

Often, simpler models are preferred: keep-it-simple principle.

Regularization: adjusting an algorithm to prefer a simpler model i.e. avoid overfitting.

Regularization: modifying the loss function to penalize large weights.

$$\hat{Y} = W \cdot X$$

The "size" [L_2 norm] of weights:

$$||w|| = \sqrt{\sum_{i=1}^{n} w_i^2}$$

The new goal for minimisation is:

$$L(w) + \lambda||w||$$

where $L(w)$ is the chosen loss function.

By minimising $\lambda||w||$ we favour models with small (close to 0) weights.

λ is a hyperparameter adjusting the tradeoff between low training loss and low weights.

Linear regression.
Ridge regularisation

$$L(w) + \lambda ||w||^2$$

Squaring the weights simplifies the calculations.

The ridge regression is based on the L_2 norm.

It penalises heavily large weights unless they are needed in the model.

The $||w||^2$ is convex i.e. easy to optimise.

Linear regression. Lasso regularisation

A regulariser based on the L_1 norm is the Lasso

$$||w||_1 = \sum_{i=1}^{n} |w_i|$$

The L_1 and L_2 regularisations can be combined:

$$L(w) + \lambda R(w)$$

$$R(w) = \lambda_1 ||w||_1 + \lambda_2 ||w||^2$$

$R(w)$ is the regularisation term or penalty.

This regularisation is called Elastic Net.

It has the advantage that it can adjust the hyperparameters to control which of the two penalties is more important.

Wise advice from others …

"Essentially all models are wrong, but some are useful".

— George Box

"The only way to find out what will happen when a complex system is disturbed is to disturb the system, not merely to observe it passively."

— Fred Mosteller and John Tukey, paraphrasing George Box

Next Lecture …

Applied Data Science

L8. Supervised learning. Classification