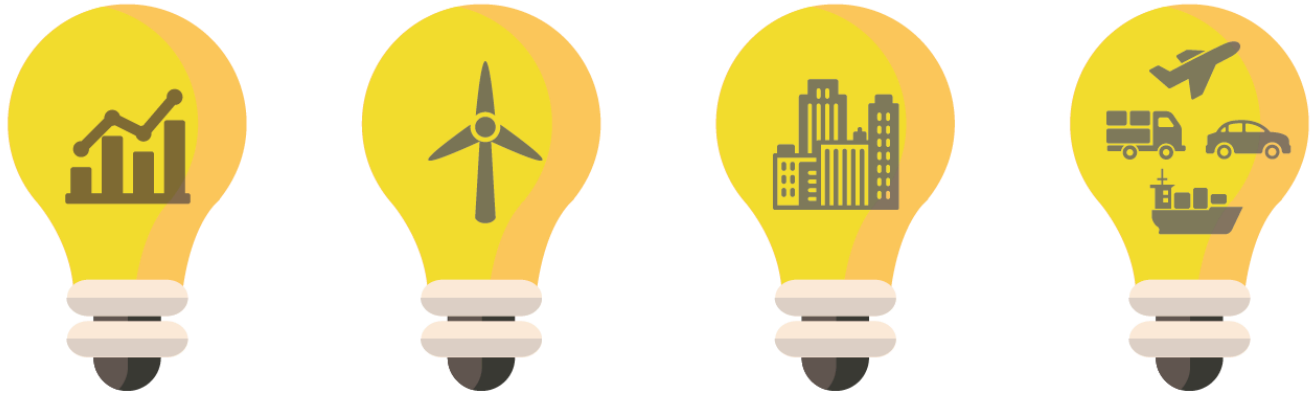
BENV0091 Lecture 4: Supervised Learning 2

Patrick de Mars



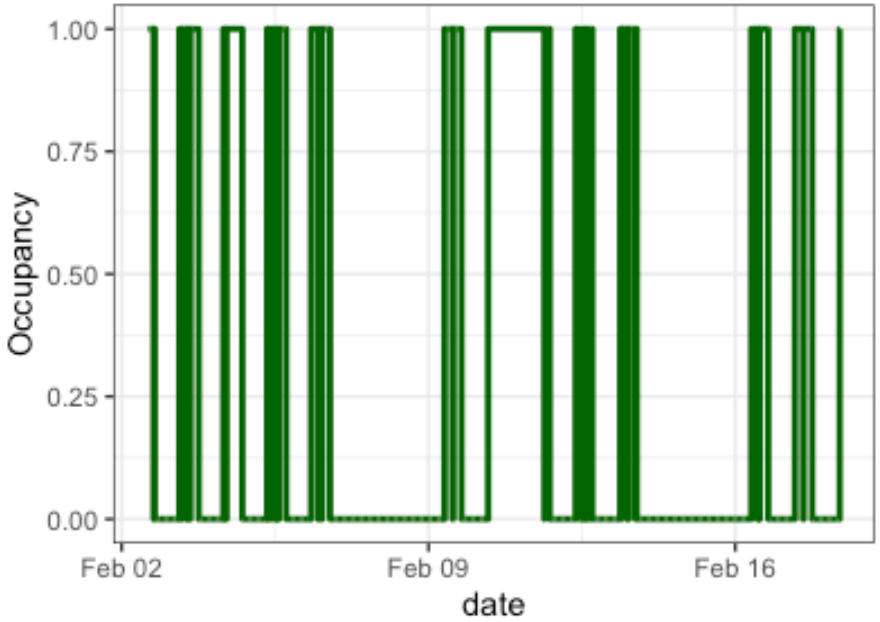
Lecture Overview

1. Supervised learning: classification
2. R Programming: the sf package
3. Git and Github
4. Group projects Q&A
5. Visualisation challenge results!

Supervised Learning: Classification

Occupancy Prediction

- Our task will be to predict building occupancy using measurements of:
  - Humidity
  - Temperature
  - CO2
- Task: use the `retrieve_data()` function from `helpers.R` to read and clean the occupancy data
- Task: build a clean data frame `df_clean` with only the columns needed to build the occupancy model
- Task: set seed and split train/test (80/20)




```r
source('helpers.R')
df <- retrieve_data('data/occupancy')
```

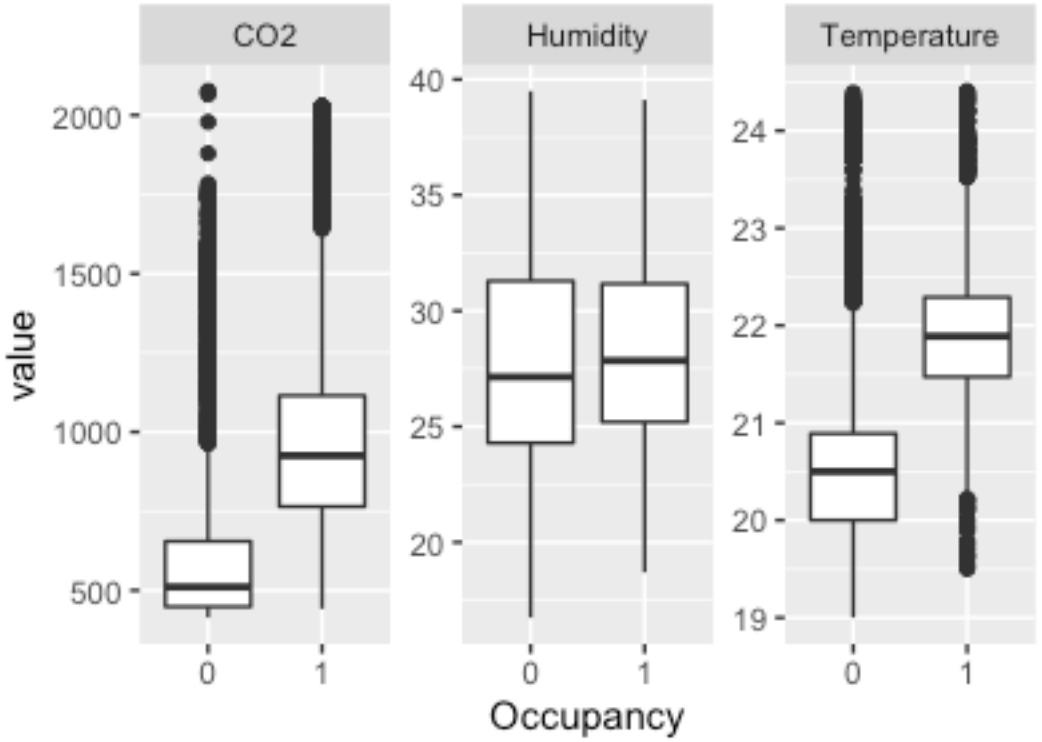
model_matrix()

	`(Intercept)`	Humidity	CO2	Temperature
	<dbl>	<dbl>	<dbl>	<dbl>
1	1	26.3	760.	23.7
2	1	26.2	770.	23.7
3	1	26.1	775.	23.7

Exploratory Data Analysis

Remember that facet_wrap(~var) splits plots by var across panels

- It's useful to have some expectation of what the key predictors are likely to be
- Task: produce the boxplots on the right using the training data
- How do distributions of each of the independent variables differ when the building is occupied vs. unoccupied?



Logistic Regression

glm() can be used to fit other kinds of linear regression models such as Poisson regression and linear regression

- For two classes, logistic regression uses maximum likelihood estimation to fit coefficients for:

$$\ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \ \ldots + \beta_N X_N$$

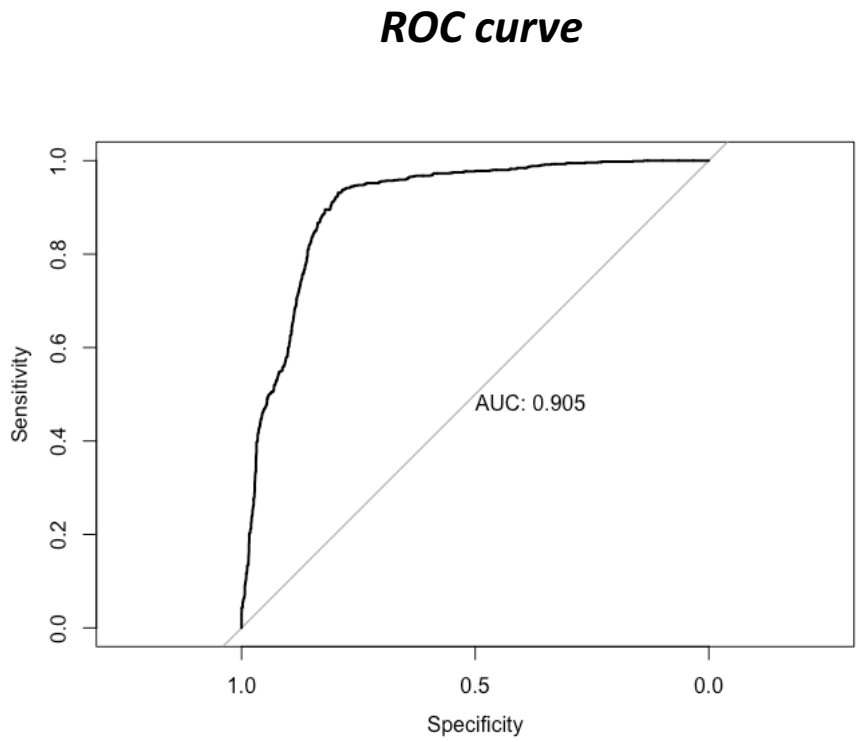
- Task: fit a logistic regression model to the training data
- When calling `predict()` using new data, logistic regression outputs the LHS of the above equation by default
- If you want to retrieve the class probability, you must set `type = "response"`

Use glm(formula, data, family = 'binomial') to fit a logistic regression model

Use predict(model, newdata, type = 'response') to retrieve class probability

Accuracy Metrics (Classification)

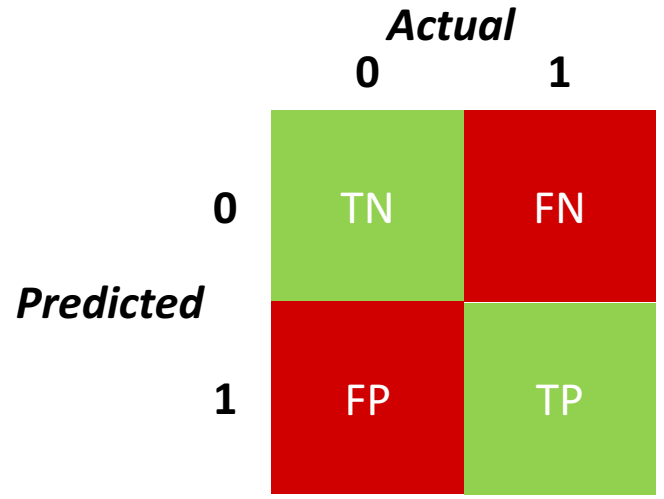
- Task: use mutate() and predict() to add a new predicted class probability column to the `test` data frame
- Accuracy metrics for classification include:
  - Accuracy (proportion of correct classifications)
  - Sensitivity/specificity
  - Area Under the Receiver Operating Characteristic Curve (ROC AUC)
  - Log loss/cross entropy loss
- Task: use the pROC function to plot the ROC curve for your predictions



Use plot(roc(actual, prob, print.auc = TRUE) to plot an ROC curve based on actual classes and the predicted probability

Confusion Matrix

- A **confusion matrix** gives a more detailed view of model predictions, serving a similar purpose to a residual plot for regression
- Task: add a column `correct` to the test data frame indicating whether the classification was correct
- Task: calculate the percentage of correct classifications
- Task: produce a confusion matrix
  - What is the sensitivity? (True positive rate)
  - What is the specificity? (True negative rate)



table(var1) counts the appearance of unique values in var1

table(var1, var2) produces a matrix counting the frequency of coincident values of var1 and var2

Downsampling

- You should have found that the **sensitivity was a lot lower than the specificity**
- Although the overall accuracy was good, the model is not actually very good at identifying when the house is occupied (not sensitive)
- This is due to **class imbalance**
- Task: quantify the class imbalance in the training data using `count()` or `table()`
- Downsampling is often used to reduce the number of observations in the majority class

Original

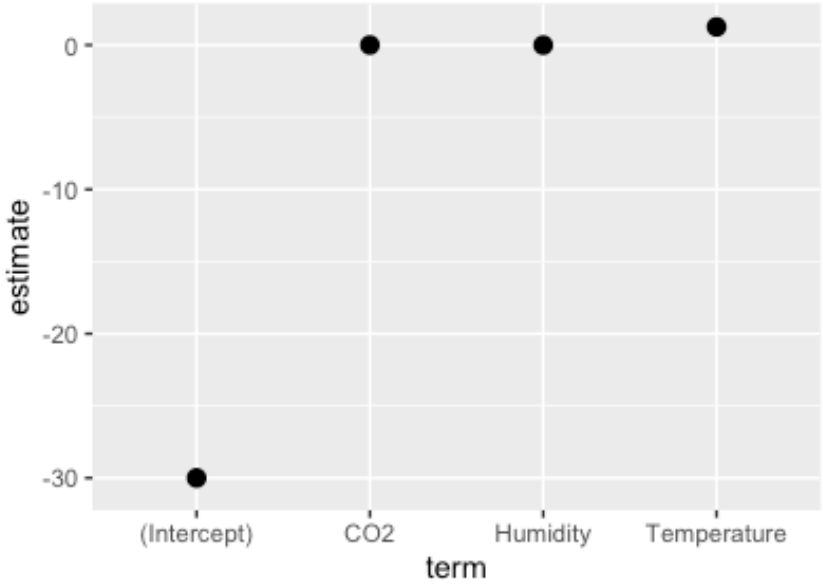
Class	Feature
A	1.2
A	1.1
A	1.5
B	1.7
A	0.7
B	0.1
A	2.2

Downsampled

Class	Feature
A	1.5
B	1.7
B	0.1
A	2.2

Coefficients of Regression Model

- Task: use `broom::tidy()` to tidy the logistic regression model
- Which coefficients are significant?
- Task: reproduce the plot on the right hand side from the tidy model, showing the estimated coefficients for each independent variable
- The plot is not very useful for determining the relative impacts of the different factors
- It is often good practice to **standardise** data in advance of training



Standardisation

$$\hat{x} = \frac{x - \mu}{\sigma}$$

(minus the mean, divide by the standard deviation)

Data Leakage

- Data leakage occurs when the training data informs the testing data, which can artificially improve performance on the test set
- Causes of data leakage:
  - Fitting the model on testing data
  - Pre-processing data before splitting train/test:
    - **Up-sampling**: training observations can end up in the test data
    - **Standardisation:** strictly the test data should not be used to calculate the standardisation factors
    - **Feature selection**: the test data is used to inform the best features

Original

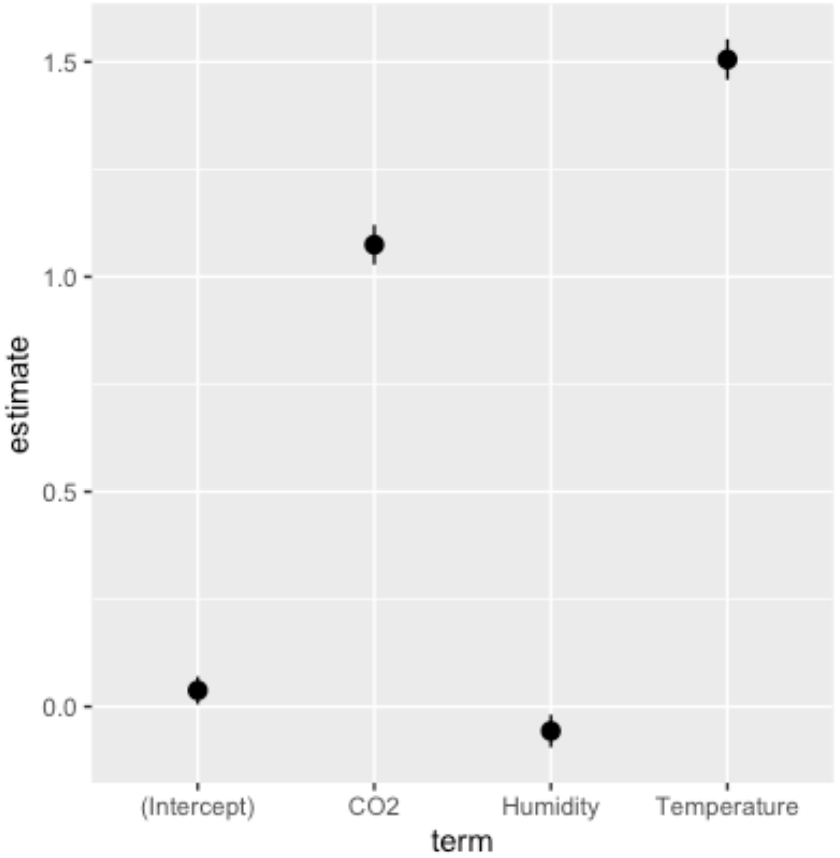
Class	Feature
A	1.2
A	1.1
A	1.5
B	1.7
A	0.7
B	0.1
A	2.2

Upsampled

Class	Feature
A	1.2
A	1.1
A	1.5
B	1.7
A	0.7
B	0.1
A	2.2
B	0.1
B	0.1
B	1.7

A More Robust and Intepretable Model

- Task: retrain the logistic regression model with the downsampling and standardisation techniques:
  - Use the downsampling function from `helpers.R` to downsample the training data
  - Write your own code to standardise each independent variable in the training data
  - Retrain the logistic regression model on the downsampled data
  - Rescale the test data **in the same way as the training data**
  - Make new predictions on the scaled test data
  - Recompute the accuracy
  - Recompute the confusion matrix
  - Reproduce the plot of coefficients (right)



R Programming: sf

The sf package and helper functions

- While code written in the tidyverse ecosystem is generally very readable; lots of R code is not!
- Here we will over review some key concepts in R programming by reviewing helper functions written for the sf package

- These functions employ a number of functions from the `sf` package
- Notice where the custom functions are called within other functions

```r
get_geo_df_limits <- function(df){
  ## Identifies the min/max of the x and y coordinates
  ## for features in a spatial dataframe

  # Extracting x/y coordinates
  df_coords <- st_coordinates(df)

  # Separating x/y coordinates
  x_vals <- df_coords[, 1]
  y_vals <- df_coords[, 2]

  # Identifying the limits of x/y
  min_x <- min(x_vals)
  max_x <- max(x_vals)
  min_y <- min(y_vals)
  max_y <- max(y_vals)

  return(c(min_x, max_x, min_y, max_y))
}
```

```r
extract_epsg_from_df <- function(df) {
  ## Extracts the epsg code from a spatial dataframe
  epsg <- st_crs(df)$epsg

  return(epsg)
}

transform_bbox_corner_crs <- function(min_x_in, max_x_in,
                                      min_y_in, max_y_in,
                                      crs_in, crs_out) {

  bottom_left <- st_point(c(min_x_in, min_y_in))
  top_right <- st_point(c(max_x_in, max_y_in))

  bbox_corners_in <- st_sfc(bottom_left, top_right, crs=crs_in)
  bbox_corners_out <- st_transform(bbox_corners_in, crs=crs_out)

  df_bbox_out <- st_coordinates(bbox_corners_out)

  min_x_out <- df_bbox_out[1, 1]
  max_x_out <- df_bbox_out[2, 1]
  min_y_out <- df_bbox_out[1, 2]
  max_y_out <- df_bbox_out[2, 2]

  return(c(min_x_out, max_x_out, min_y_out, max_y_out))
}
```

```r
get_geo_df_plot_lims <- function(df, crs_out=NA){
  # Identify limits
  geo_df_limits <- get_geo_df_limits(df)

  min_x_out <- geo_df_limits[1]
  max_x_out <- geo_df_limits[2]
  min_y_out <- geo_df_limits[3]
  max_y_out <- geo_df_limits[4]

  # Optionally convert to new crs
  if (!is.na(crs_out)) {
    crs_in <- extract_epsg_from_df(df) %>%
      strtoi() %>%
      st_crs

    out_coords <- transform_bbox_corner_crs(min_x_in, max_x_in,
                                            min_y_in, max_y_in,
                                            crs_in, crs_out)

  #   c(min_x_out, max_x_out, min_y_out, max_y_out) %<-% out_coords
    min_x_out <- out_coords[1]
    max_x_out <- out_coords[2]
    min_y_out <- out_coords[3]
    max_y_out <- out_coords[4]
  }

  # Convert to ggplot2 coordinate limits
  coord_lims <- coord_sf(xlim=c(min_x_out, max_x_out),
                         ylim=c(min_y_out, max_y_out))

  return(coord_lims)
}
```

```r
get_geo_df_limits <- function(df){
  ## Identifies the min/max of the x and y coordinates
  ## for features in a spatial dataframe

  # Extracting x/y coordinates
  df_coords <- st_coordinates(df)

  # Separating x/y coordinates
  x_vals <- df_coords[, 1]
  y_vals <- df_coords[, 2]

  # Identifying the limits of x/y
  min_x <- min(x_vals)
  max_x <- max(x_vals)
  min_y <- min(y_vals)
  max_y <- max(y_vals)

  return(c(min_x, max_x, min_y, max_y))
}
```

Extracting columns of a data frame

Returning multiple values

```r
extract_epsg_from_df <- function(df) {
  ## Extracts the epsg code from a spatial dataframe
  epsg <- st_crs(df)$epsg

  return(epsg)
}

transform_bbox_corner_crs <- function(min_x_in, max_x_in,
                                      min_y_in, max_y_in,
                                      crs_in, crs_out) {

  bottom_left <- st_point(c(min_x_in, min_y_in))
  top_right <- st_point(c(max_x_in, max_y_in))

  bbox_corners_in <- st_sfc(bottom_left, top_right, crs=crs_in)
  bbox_corners_out <- st_transform(bbox_corners_in, crs=crs_out)

  df_bbox_out <- st_coordinates(bbox_corners_out)

  min_x_out <- df_bbox_out[1, 1]
  max_x_out <- df_bbox_out[2, 1]
  min_y_out <- df_bbox_out[1, 2]
  max_y_out <- df_bbox_out[2, 2]

  return(c(min_x_out, max_x_out, min_y_out, max_y_out))
}
```

Using $ to extract a value by name

Extracting values from a matrix

```r
get_geo_df_plot_lims <- function(df, crs_out=NA){
  # Identify limits
  geo_df_limits <- get_geo_df_limits(df)

  min_x_out <- geo_df_limits[1]
  max_x_out <- geo_df_limits[2]
  min_y_out <- geo_df_limits[3]
  max_y_out <- geo_df_limits[4]

  # Optionally convert to new crs
  if (!is.na(crs_out)) {
    crs_in <- extract_epsg_from_df(df) %>%
      strtoi() %>%
      st_crs

    out_coords <- transform_bbox_corner_crs(min_x_in, max_x_in,
                                            min_y_in, max_y_in,
                                            crs_in, crs_out)

#   c(min_x_out, max_x_out, min_y_out, max_y_out) %<-% out_coords
    min_x_out <- out_coords[1]
    max_x_out <- out_coords[2]
    min_y_out <- out_coords[3]
    max_y_out <- out_coords[4]
  }

  # Convert to ggplot2 coordinate limits
  coord_lims <- coord_sf(xlim=c(min_x_out, max_x_out),
                         ylim=c(min_y_out, max_y_out))

  return(coord_lims)
}
```

Extracting values from a vector

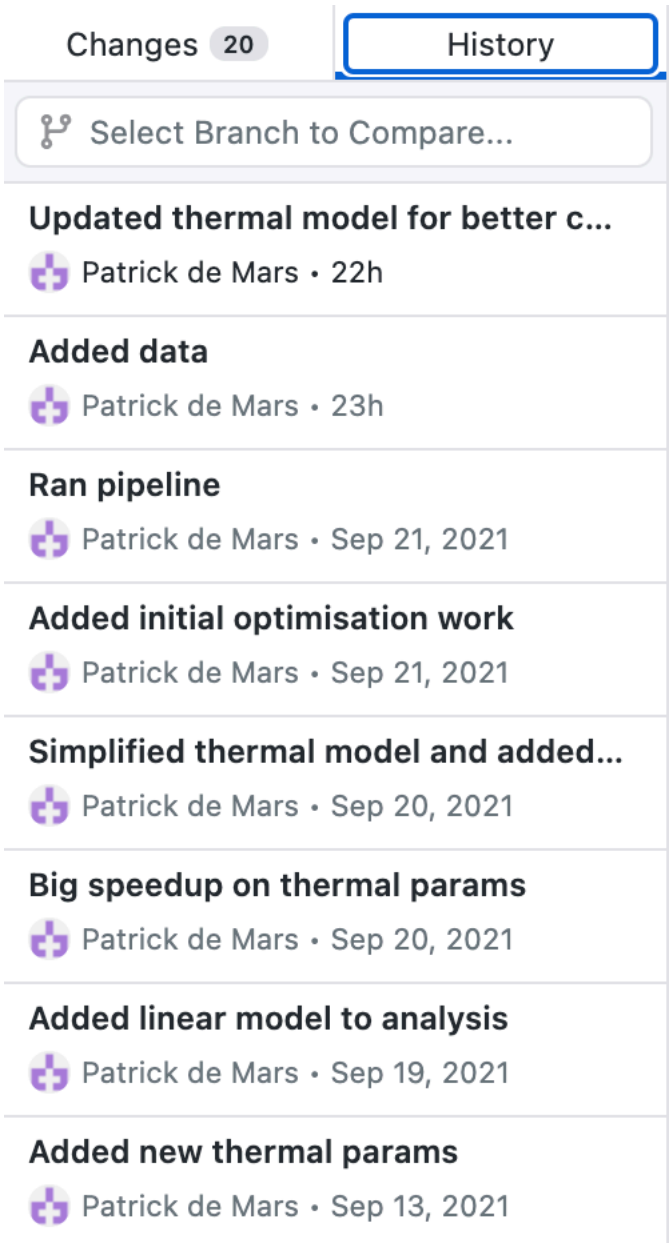
Logical 'not' with `!`

Version Control with Git and Github

Git

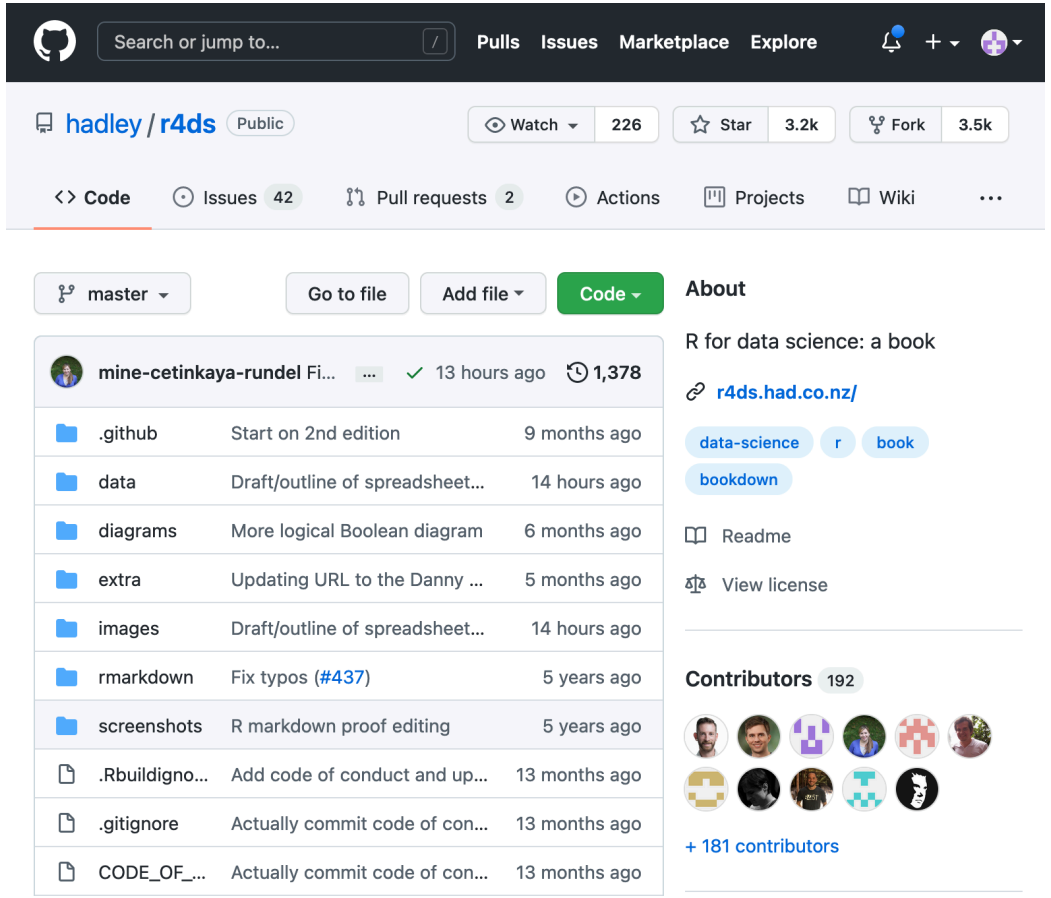
- Git is a **distributed version control system** – it allows you to keep track of changes to your code in a lightweight way
- Git stores a history of changes to your project
- Your Git history is a sequence of **commits** which are organised into **branches**
- Using Git, you are able to:
  - Revert to old *commits*
  - Develop new features/changes in parallel *branches* and safely merge changes between branches
  - Easily share code and Git history with collaborators via Github

A Git history



Github

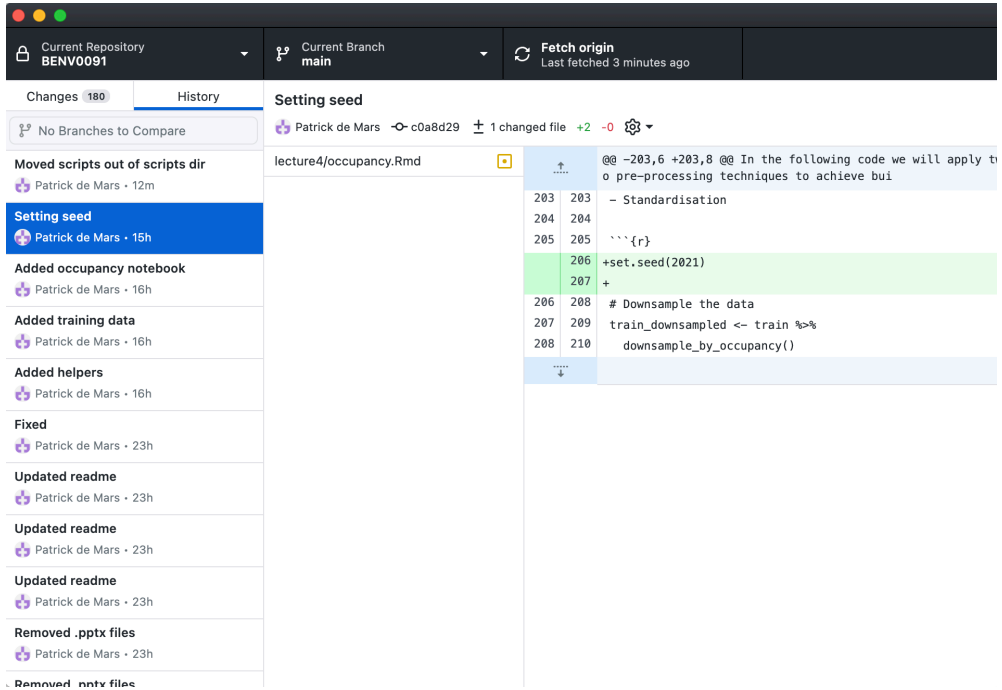
- Github hosts **repositories** remotely on the Internet, which can be accessed from anywhere
- When using Git, you will typically make changes locally in a series of **commits**, then **push** them to the remote repository on Github
- If someone else makes a change to the repository, you can **pull** those changes from Github
- Github has facilitates collaboration through **pull requests**, **issues** and other other functions



Interfacing with Git

- There are several ways to issue Git commands:
  - Command line interface
  - RStudio interface
  - Github Desktop
- We will use the Github Desktop interface to learn the basics, but you may prefer to use one of the other methods

Github Desktop

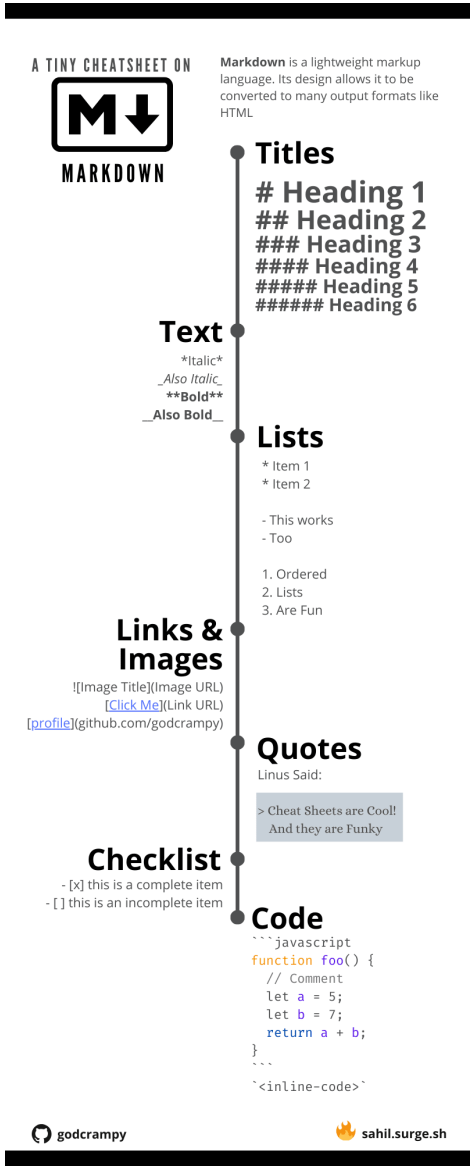


Terminology

- **Repository**: project directory including Git history
- **Local:** your own computer
- **Remote**: elsewhere (e.g. hosted by Github)
- **Clone**: download a repository to your local machine
- **Fork**: create a copy of a repository that is tied to your Github account
- **Commit**: record a group of changes to files (similar to save). Creates a “checkpoint” in your Git history
- **Push**: upload your local repository to a remote one
- **Pull:** update your local repository with updates from a remote repository
- **Branch:** identifying name given to certain commits (e.g. ‘main’, ‘feature_name’)
- **Merge:** combine two branches
- **Pull request:** ask other repository maintainers to approve changes you want to merge into a branch

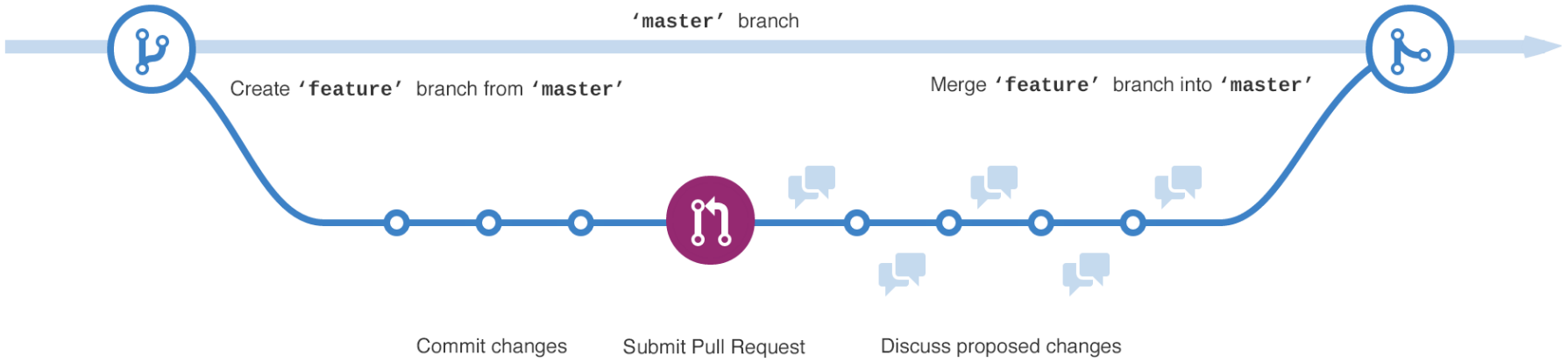
Exercise: Introduction to Git

- Task: fork, clone, commit and push:
  - Open a web browser and go to: https://github.com/pwdemars/BENV0091/
  - **Fork** the repository (top right)
  - Open Github Desktop and **clone** the forked repository (File > Clone Repository > URL)
  - Open the README.md file in RStudio and add your name to it
  - In Github Desktop, **commit** your change
  - In Github Desktop, **push** your change to the remote repository
  - Visit your repository in a web browser view the change to the README.md file (it should be visible on the homepage)



Branches

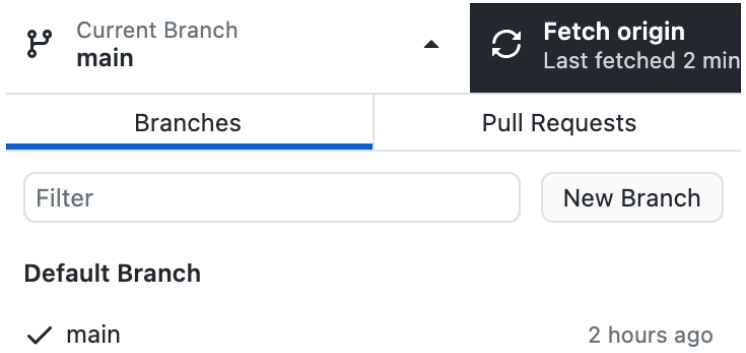
- Projects typically have a main or master branch
- When a new feature is developed, this is often done on a parallel branch and merged into the main branch
- You can experiment in the feature branch without affecting the main branch
- Branches are usually merged through a **pull request** – allowing you to discuss changes with other users, resolve **conflicts** and make further changes



Exercise: Branch and Pull Request

- Task: create a new branch and open a pull request:
  - In Github Desktop, create a new **branch** called `feature`
  - Open occupancy.Rmd in the lecture4 subdirectory and add `author: “your name”` at the top (see right)
  - **Commit** the change
  - **Push** the change (called publish when creating a new branch)
  - From Github Desktop, open a **pull request**
  - **Merge** the pull request (in your browser)
  - Delete the branch

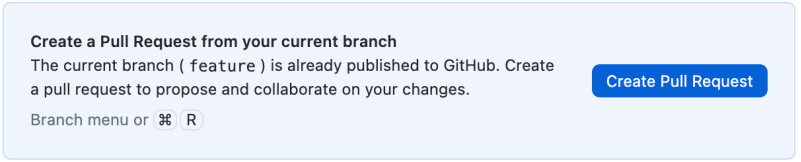
Create a new branch



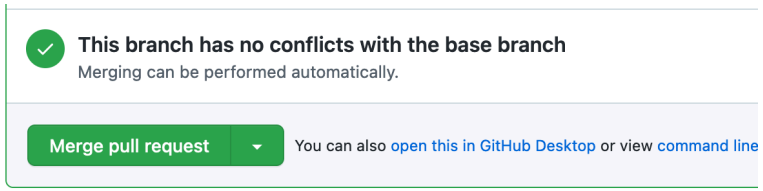
Change occupancy.Rmd

```
---
title: "Occupancy Prediction"
author: "Patrick de Mars"
```

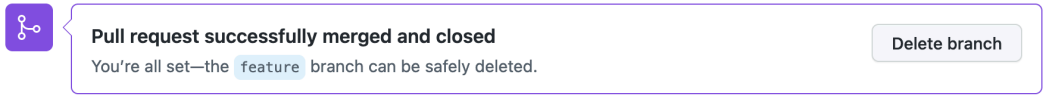
Open a pull request



Merge pull request

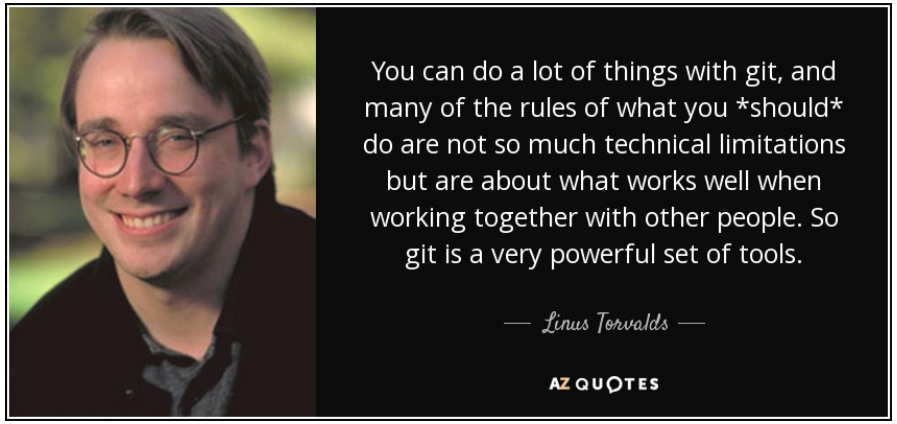


Delete the branch



Further Notes

- To initialise Git on an existing directory, you can:
  - Create a new repository on Github and follow the command line instructions
  - Create a new repository on Github Desktop and specify the path of your existing directory
- You may not want to track or put all of your files on Github. You can explicitly prevent files from being tracked by adding them to a **.gitignore** file
  - Add sensitive data to your .gitignore file!
- Recommended reading: **Github documentation and Getting Started guides**



Group Projects Q&A