# Machine learning in R

*with TidyModels*

SIB days
24/June/2024

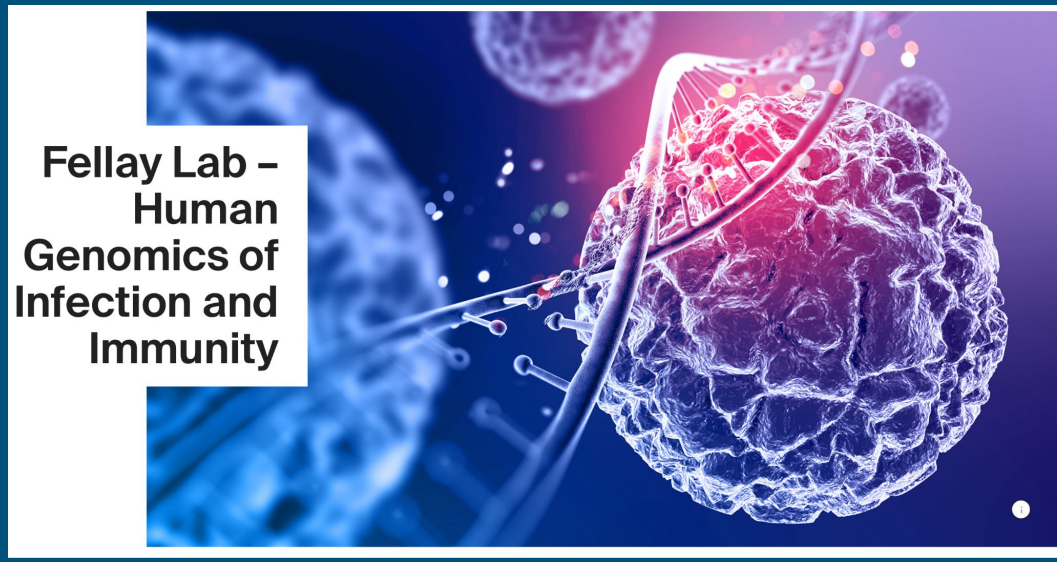
# All materials (code, slides, data) can be found at:

https://github.com/AliSaadatV/ML_in_R_workshop

# Who are we?

Ali

Simon



# Who are you?

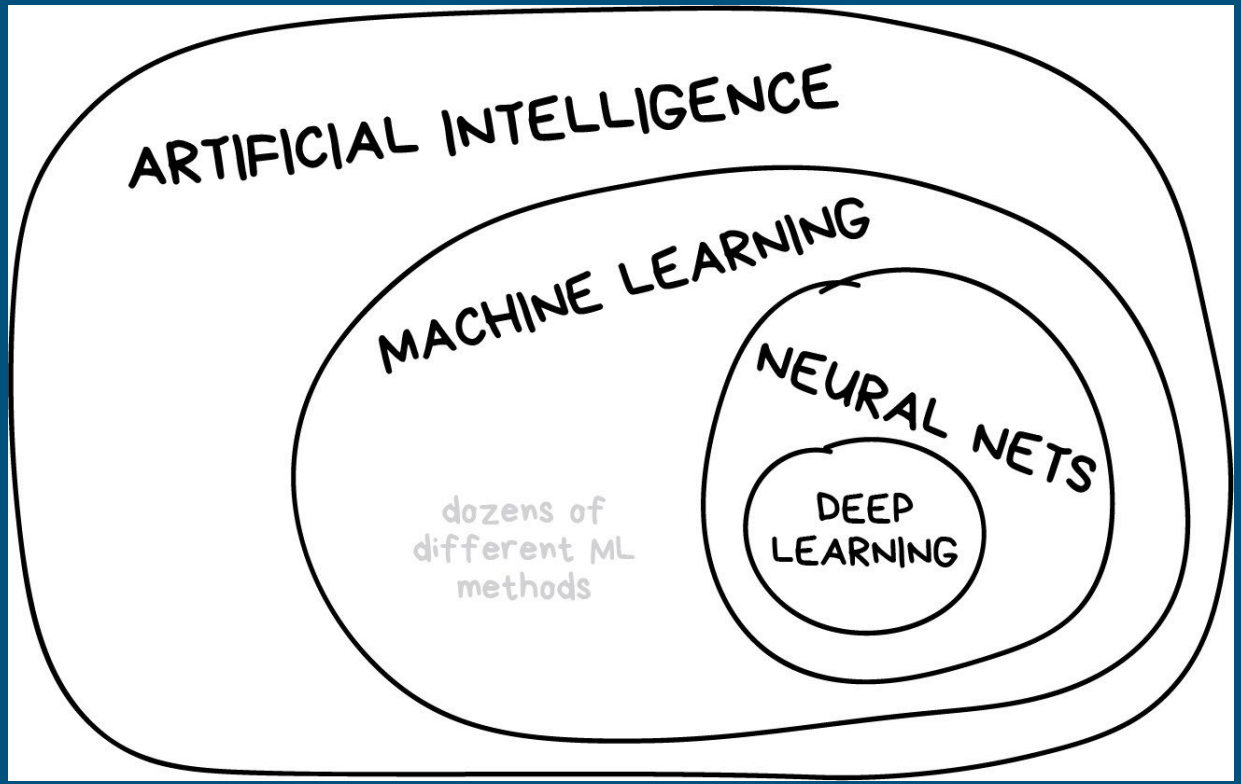
- You have exposure to basic statistical concepts
- You do **not** need intermediate or expert familiarity with modeling or ML
- You know the basics of R programming

# Table of content


1. Introduction
    a. What is machine learning?
    b. What is TidyModels?
    c. What is a pipe?
    d. Our dataset for workshop
2. Preprocessing
    a. Exploratory data analysis
    b. Splitting data
3. Feature engineering
4. Model fitting & Hyperparameter tuning
5. Model selection & Evaluation
6. Bonus: Model interpretation
7. Bonus: all code in one slide!

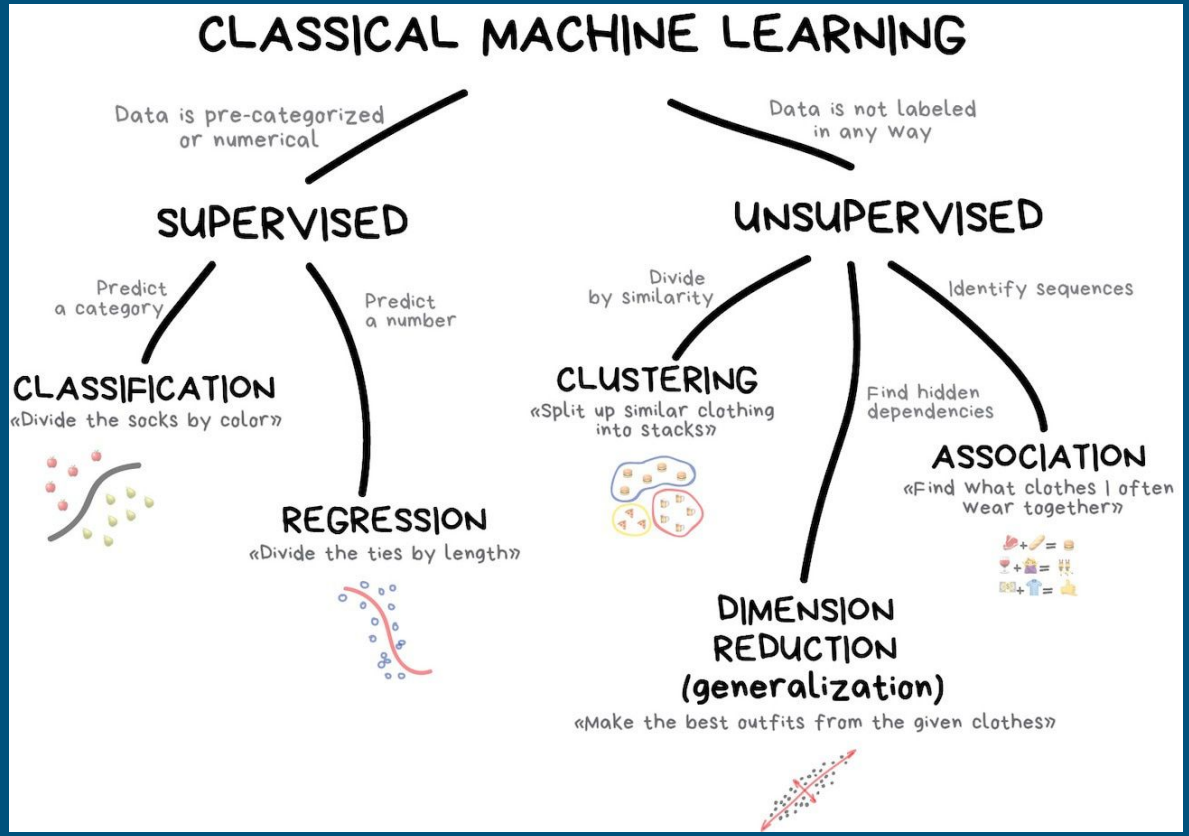
# Introduction

# What is machine learning?



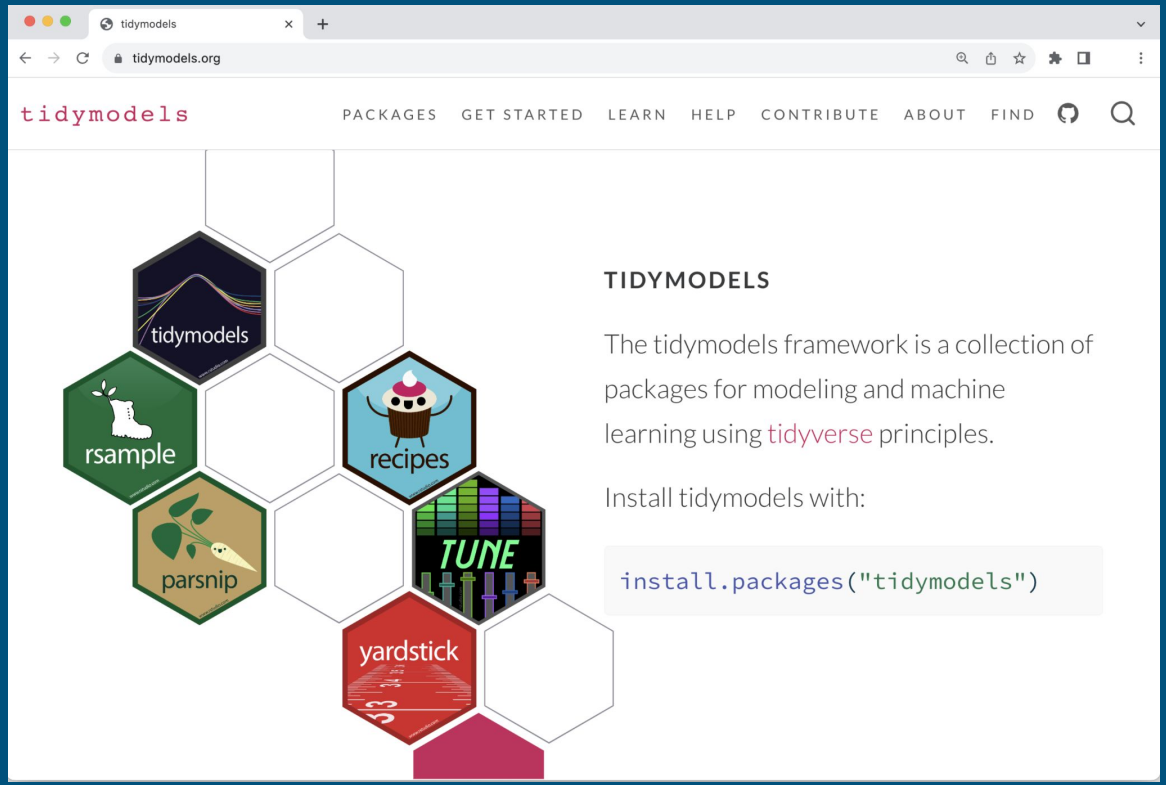

https://vas3k.com/blog/machine_learning/

# What is machine learning?




https://vas3k.com/blog/machine_learning/

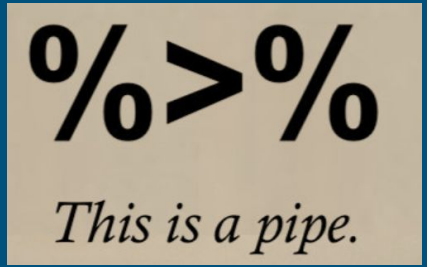
# What is TidyModels?



# What is TidyModels?

```
library(tidymodels)
#> ── Attaching packages ──────────────────────────── tidymodels 1.2.0 ──
#> ✔ broom        1.0.5      ✔ rsample      1.2.1
#> ✔ dials        1.2.1      ✔ tibble       3.2.1
#> ✔ dplyr        1.1.4      ✔ tidyr        1.3.1
#> ✔ infer        1.0.7      ✔ tune         1.2.1
#> ✔ modeldata    1.3.0      ✔ workflows    1.1.4
#> ✔ parsnip      1.2.1      ✔ workflowsets 1.1.0
#> ✔ purrr        1.0.2      ✔ yardstick    1.3.1
#> ✔ recipes      1.0.10
#> ── Conflicts ─────────────────────────────── tidymodels_conflicts() ──
#> ✖ purrr::discard() masks scales::discard()
#> ✖ dplyr::filter()  masks stats::filter()
#> ✖ dplyr::lag()     masks stats::lag()
#> ✖ recipes::step()  masks stats::step()
#> • Use tidymodels_prefer() to resolve common conflicts.
```

# What are pipes?




```r
# Load dplyr
library(dplyr)
```

```r
# Load data
data("iris")

# View data
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

# What are pipes?

```r
# Filter and mutate data
new_iris <- iris %>%
  filter(Sepal.Length > 5 & Petal.Length > 3) %>%
  mutate(Sepal.Area = Sepal.Length * Sepal.Width,
         Petal.Area = Petal.Length * Petal.Width)

# View new data
head(new_iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species Sepal.Area
## 1          7.0         3.2          4.7         1.4 versicolor      22.40
## 2          6.4         3.2          4.5         1.5 versicolor      20.48
## 3          6.9         3.1          4.9         1.5 versicolor      21.39
## 4          5.5         2.3          4.0         1.3 versicolor      12.65
## 5          6.5         2.8          4.6         1.5 versicolor      18.20
## 6          5.7         2.8          4.5         1.3 versicolor      15.96
##   Petal.Area
## 1       6.58
## 2       6.75
## 3       7.35
## 4       5.20
## 5       6.90
## 6       5.85
```

# Today's dataset

> Front Genet. 2019 Jul 9:10:600. doi: 10.3389/fgene.2019.00600. eCollection 2019.

## Five-Feature Model for Developing the Classifier for Synergistic vs. Antagonistic Drug Combinations Built by XGBoost

Xiangjun Ji [1] [2], Weida Tong [3], Zhichao Liu [3], Tieliu Shi [1] [4]

Affiliations + expand

PMID: 31338106 PMCID: PMC6629777 DOI: 10.3389/fgene.2019.00600

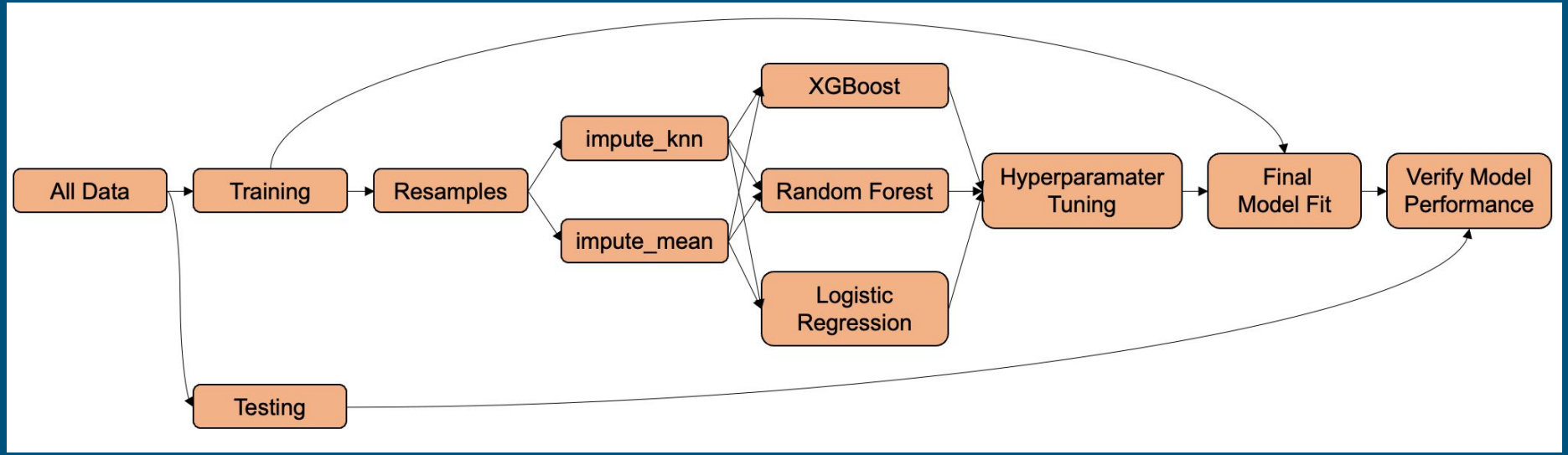
Synergistic: (Drug1+Drug2) > (Drug1) + (Drug2)

Antagonistic: (Drug1+Drug2) < (Drug1) + (Drug2)

# Today's dataset

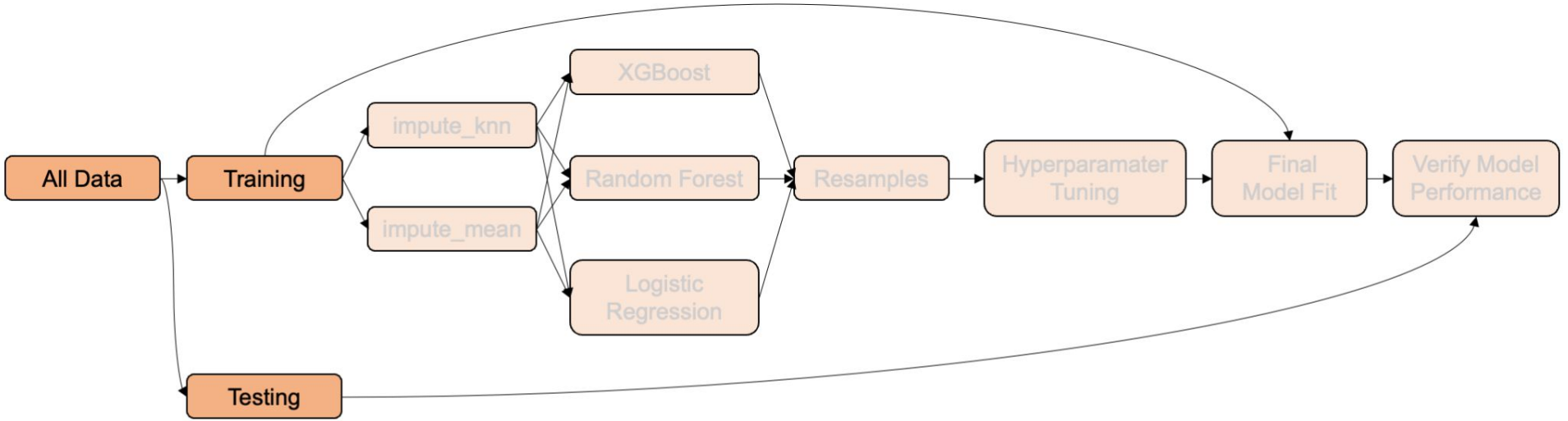
| drug1 | drug2 | Disease Intersection Degree | Adverse Drug Reaction Intersection Degree | The Similarity of Mode of Action | Biological Process Similarity | Separation Score | judge |
|---|---|---|---|---|---|---|---|
| PACLITAXEL | ZOLEDRONIC ACID | | 0.285714286 | 1 | 0.228 | | 1 |
| DOXORUBICIN | ZOLEDRONIC ACID | | 0.226890756 | | 0.251 | | 1 |
| AMBRISENTAN | TADALAFIL | | 0.328205128 | | 0.216 | 0.533333333 | 1 |
| EFAVIRENZ | NFV | | 0.29 | | | | 1 |
| 3TC | EFAVIRENZ | 0 | 0.32122905 | 1 | | | 1 |
| LEVOFLOXACIN | TIGECYCLINE | 0 | 0.260393873 | | | | 1 |
| 5-FLUOROURACIL | TAMOXIFEN | 0.066666667 | 0.177710843 | | 0.441 | 0.399305556 | 1 |
| IMIPRAMINE | MORPHINE | 0.045454545 | 0.236074271 | -1 | 0.671 | 0.076428571 | 1 |
| FORMOTEROL | TIOTROPIUM | 0 | 0.28125 | | 0.541 | | 1 |
| DECITABINE | DEPSIPEPTIDE | | 0.220408163 | | | | 1 |
| 5-FLUOROURACIL | IMIQUIMOD | 0.052631579 | 0.174731183 | | 0.229 | 0.75 | 1 |
| AZT | EFAVIRENZ | 0 | 0.265714286 | 1 | 0.506 | | 1 |
| ALBUTEROL | IPRATROPIUM BROMIDE | 0.25 | 0.305970149 | | 0.555 | | 1 |
| CANDESARTAN | RAMIPRIL | | 0.235042735 | | 0.467 | | 1 |
| GABAPENTIN | OXCARBAZEPINE | 0 | 0.340909091 | 1 | 0.507 | 0.083333333 | 1 |
| ENALAPRIL | LOVASTATIN | 0 | 0.273809524 | | 0.542 | -0.014102564 | 1 |

# End-to-end ML


All Data
Training
Resamples
impute_knn
impute_mean
XGBoost
Random Forest
Logistic Regression
Hyperparamater Tuning
Final Model Fit
Verify Model Performance
Testing

# Questions?

# Preprocessing



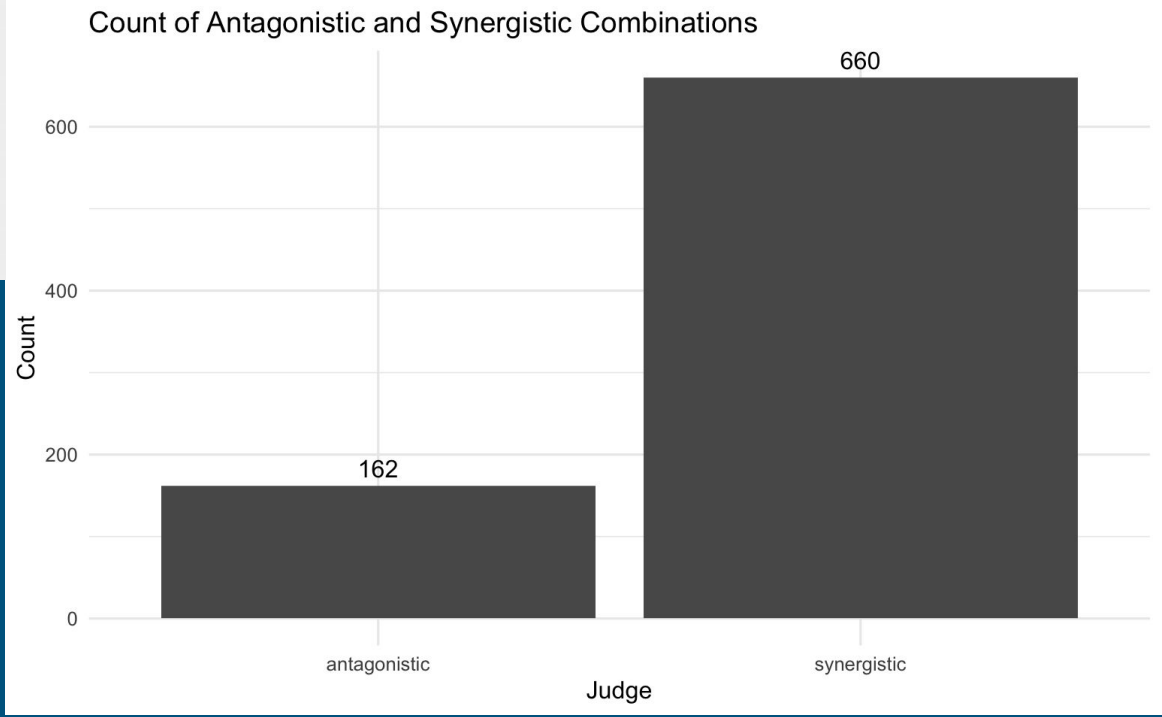
# Exploratory data analysis (EDA)

1. What’s the distribution of output variable?
2. What’s the distribution of numerical features?
3. What’s the correlation between features?

# Exploratory data analysis (EDA)

## What's the distribution of output variable?

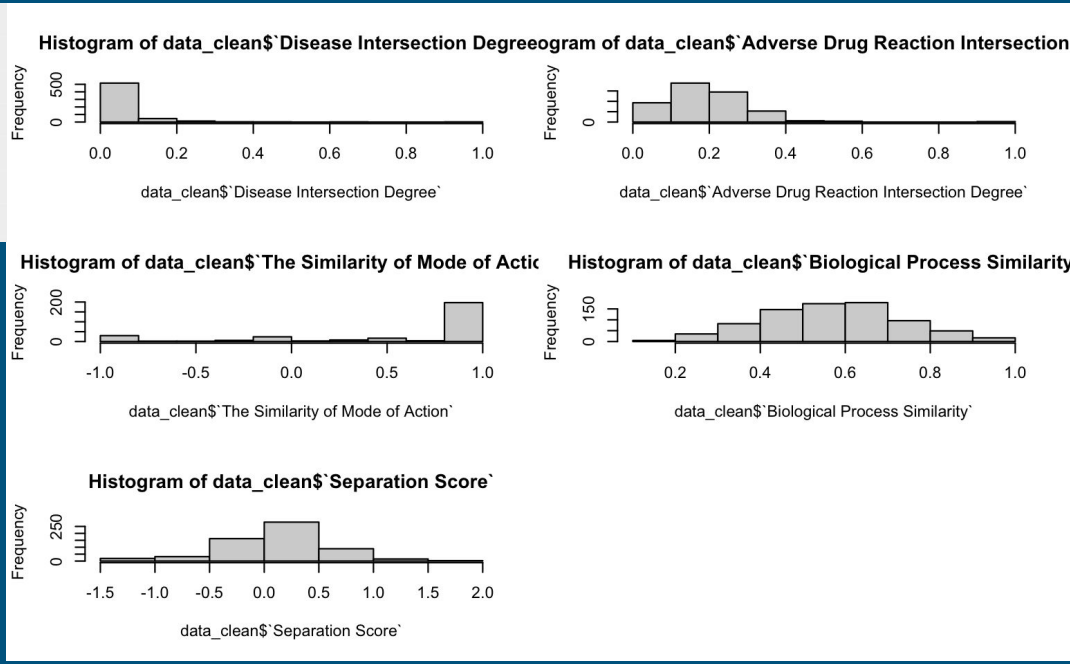
```r
# Rename the factors
named_judge <- factor(data_clean$judge, levels = c(0, 1), labels =
c("antagonistic", "synergistic"))
# Create the bar plot
ggplot(data_clean, aes(x = factor(named_judge))) +
  geom_bar() +
  geom_text(stat = 'count', aes(label = ..count..), vjust = -0.5) +
  labs(title = "Count of Antagonistic and Synergistic Combinations",
       x = "Judge",
       y = "Count") +
  theme_minimal()
```



# Exploratory data analysis (EDA)

## What's the distribution of numerical features?

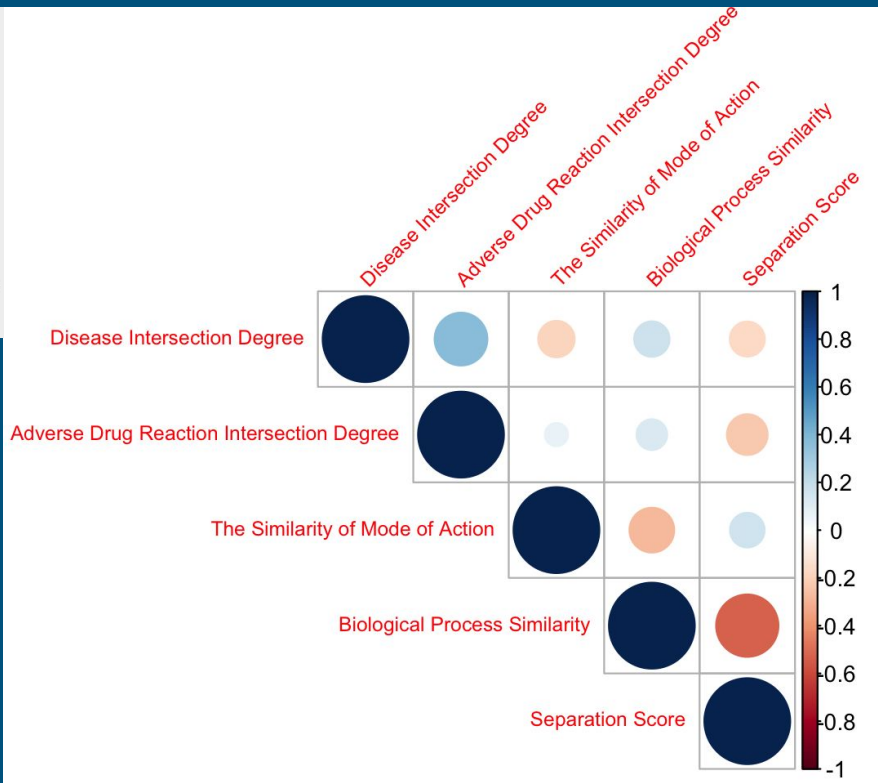
```
# What's the distribution of numerical features?
par(mfrow = c(3,2))
hist(data_clean$`Disease Intersection Degree`)
hist(data_clean$`Adverse Drug Reaction Intersection Degree`)
hist(data_clean$`The Similarity of Mode of Action`)
hist(data_clean$`Biological Process Similarity`)
hist(data_clean$`Separation Score`)
```



# Exploratory data analysis (EDA)

## Correlation between features

```r
# What's the correlation of numerical features?
corrplot_input <- na.omit(subset(data_clean, select = -c(judge, Pubmed,
DCDB)))

# Calculate the correlation matrix
corr_matrix <- cor(corrplot_input)

# Create the correlation plot
corrplot::corrplot(corr_matrix, method = "circle", tl.srt = 45, tl.cex = 0.7,
type = "upper")
```



# Checklist for features

1. Will the variable be available at inference time?
2. Is it ethical / legal to include this variable?
3. Does variable contribute to model explanation?

# Data splitting

- We split the data into **train** and **test** set. Usually 70%-90% goes into train set.
- **Train** set → **estimate** the parameters
- **Test** set → **evaluate** the performance of the model
- Do **NOT** use test set during training.

# Data splitting in TidyModels

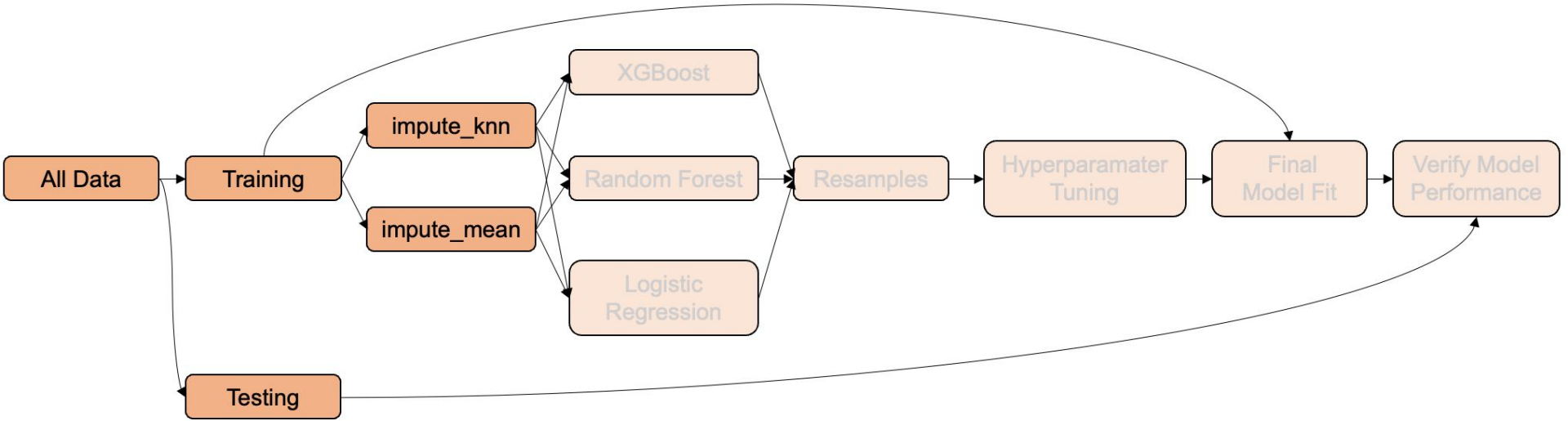
```
splits <- initial_split(data_clean, prop = 0.75, strata = judge)

data_train <- training(splits)
data_test  <- testing(splits)

data_train %>%
  dplyr::count(judge) %>%
  dplyr::mutate(proportion = n/sum(n))

data_test %>%
  dplyr::count(judge) %>%
  dplyr::mutate(proportion = n/sum(n))
```

| judge <chr> | n <int> | proportion <dbl> |
|---|---|---|
| 0 | 121 | 0.1964286 |
| 1 | 495 | 0.8035714 |

| judge <chr> | n <int> | proportion <dbl> |
|---|---|---|
| 0 | 41 | 0.1990291 |
| 1 | 165 | 0.8009709 |

# Questions?

# Feature engineering



# Preprocessing vs Feature engineering

- Preprocessing: the process of cleaning and organizing the raw data

- Feature engineering: what you do to the original predictors to make the model do the least work to perform great

# Examples of feature engineering

- Do qualitative predictors require a numeric encoding?
- Should columns with a single unique value be removed?
- Does the model struggle with missing data?
- Does the model struggle with correlated predictors?
- Should predictors be centered and scaled?

# Two types of feature engineering

Static:

- Log
- Square root
- Inverse

Trained:

- PCA
- Imputation
- Center and scaling

# Feature engineering in TidyModels

- One challenge is that there are many possible options to use
- It’s easy to try various feature engineering combinations with **recipes** package

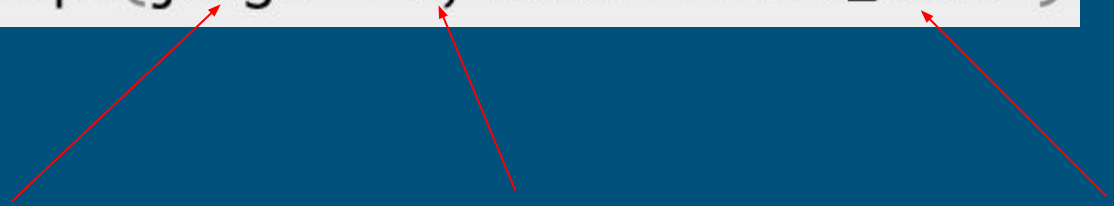
# How to create a recipe?

1) Use ***recipe*** function to indicate output and input variables
2) Use ***step_**** functions to process the variables

# How to create a recipe?

*Simple Example*

```
data_rec_mean <-
  recipe(judge ~ ., data = data_clean)
```



We specify the variable to predict

We specify all other variables should be predictor variables

We specify the dataset to get these variables from

# Today's Recipes

```
data_rec_mean <-
  recipe(judge ~ ., data = data_clean) %>%
  update_role(Pubmed, DCDB, new_role = "ID") %>% # Specify variables that are IDs, thus not predictors
  step_zv(all_predictors()) %>% # remove zero or near zero variance features - none are removed
  step_impute_mean(all_predictors()) %>% # Impute all missing values
  step_YeoJohnson(all_predictors()) %>% # Use Yeo_Johnson to resolve skewness
  step_normalize(all_predictors()) # Standardize: center and scale
```

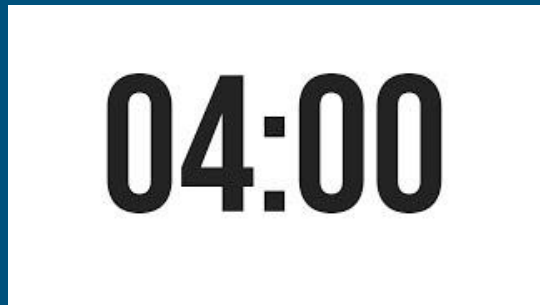
```
data_rec_knn <-
  recipe(judge ~ ., data = data_clean) %>%
  update_role(Pubmed, DCDB, new_role = "ID") %>%
  step_zv(all_predictors()) %>%
  step_impute_knn(all_predictors(), neighbors = 5) %>%
  step_YeoJohnson(all_predictors()) %>%
  step_normalize(all_predictors())
```

We will use two recipes to compare performance. One imputes missing values by the mean, while the other by k nearest neighbours.

# Your turn!

Go to **recipes.tidymodels.org/reference**

- What normalizations are available?
- What other options are available instead of ***all_predictors()***?
- If we had a categorical variable, which ***step_**** function to use?
- How to model interactions?



# Recommended preprocessing

Table A.1: Preprocessing methods for different models.

| model | dummy | zv | impute | decorrelate | normalize | transform |
|---|---|---|---|---|---|---|
| C5_rules() | × | × | × | × | × | × |
| bag_mars() | ✔ | × | ✔ | ○ | × | ○ |
| bag_tree() | × | × | × | ○[1] | × | × |
| bart() | × | × | × | ○[1] | × | × |
| boost_tree() | ×[2] | ○ | ✔[2] | ○[1] | × | × |
| cubist_rules() | × | × | × | × | × | × |
| decision_tree() | × | × | × | ○[1] | × | × |
| discrim_flexible() | ✔ | × | ✔ | ✔ | × | ○ |
| discrim_linear() | ✔ | ✔ | ✔ | ✔ | × | ○ |
| discrim_regularized() | ✔ | ✔ | ✔ | ✔ | × | ○ |
| gen_additive_mod() | ✔ | ✔ | ✔ | ✔ | × | ○ |
| linear_reg() | ✔ | ✔ | ✔ | ✔ | × | ○ |
| logistic_reg() | ✔ | ✔ | ✔ | ✔ | × | ○ |
| mars() | ✔ | × | ✔ | ○ | × | ○ |
| mlp() | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| multinom_reg() | ✔ | ✔ | ✔ | ✔ | ×[2] | ○ |
| naive_Bayes() | × | ✔ | ✔ | ○[1] | × | × |
| nearest_neighbor() | ✔ | ✔ | ✔ | ○ | ✔ | ✔ |
| pls() | ✔ | ✔ | ✔ | × | ✔ | ✔ |
| poisson_reg() | ✔ | ✔ | ✔ | ✔ | × | ○ |
| rand_forest() | × | ○ | ✔[2] | ○[1] | × | × |
| rule_fit() | ✔ | × | ✔ | ○[1] | ✔ | × |
| svm_*() | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

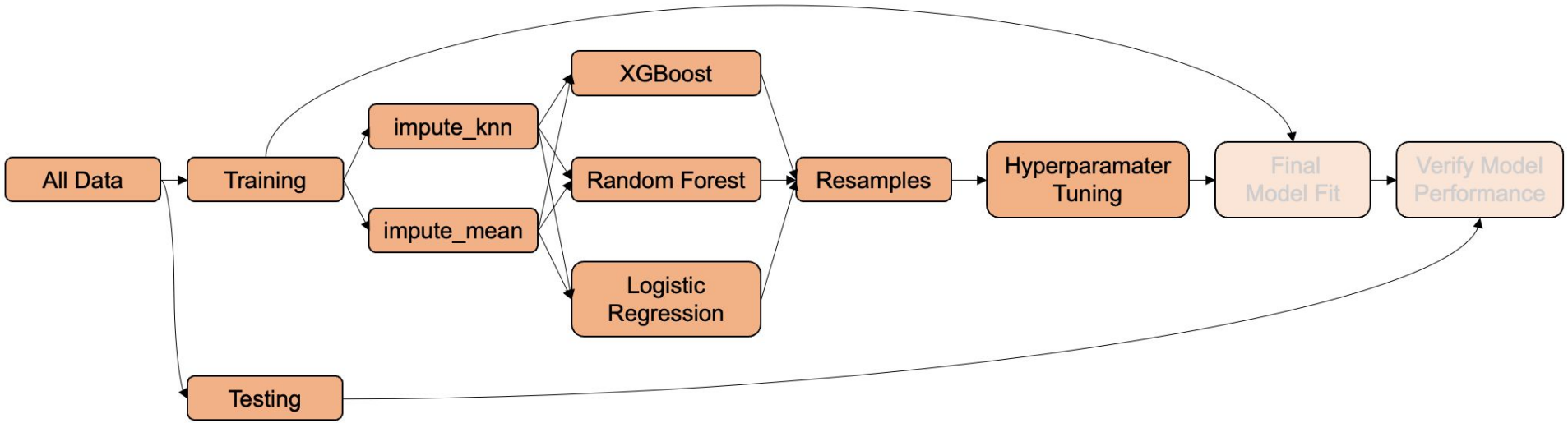
tmwr.org/pre-proc-table

# Hands-on coding!

1. Go to *https://github.com/AliSaadatV/ML_in_R_workshop*
2. Click on *Code* (the green button)
3. Click on *Download ZIP*
4. Unzip the files and open *TidyModelsWorkshop.Rmd*
5. Read and understand each line of code (including comments)
6. Run the code until before *1.4 Build models*

(Coding will continue in the next section, so keep R open if you want to keep coding!)

# Model fitting & Hyperparameter tuning



# How to define a model?

- Choose a model
- Specify an engine
- Set the mode


tidymodels.org/find/parsnip/

| title | model | engine | topic | mode |
|---|---|---|---|---|
| All | All | All | All | All |
| Boosted trees via lightgbm | boost_tree | lightgbm | boost_tree_lightgbm | classification |
| Boosted trees | boost_tree | mboost | boost_tree_mboost | censored regression |
| Boosted trees via Spark | boost_tree | spark | boost_tree_spark | classification |
| Boosted trees via Spark | boost_tree | spark | boost_tree_spark | regression |
| Boosted trees via xgboost | boost_tree | xgboost | boost_tree_xgboost | classification |
| Boosted trees via xgboost | boost_tree | xgboost | boost_tree_xgboost | regression |
| Cubist rule-based regression models | cubist_rules | Cubist | cubist_rules_Cubist | regression |
| Decision trees via C5.0 | decision_tree | C5.0 | decision_tree_C5.0 | classification |
| Decision trees via partykit | decision_tree | partykit | decision_tree_partykit | regression |
| Decision trees via partykit | decision_tree | partykit | decision_tree_partykit | classification |

# Model definition

*Examples*

```
xgboost_spec <-
  boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```
> show_engines("boost_tree")
# A tibble: 5 × 2
  engine   mode
  <chr>    <chr>
1 xgboost  classification
2 xgboost  regression
3 C5.0     classification
4 spark    classification
5 spark    regression
```

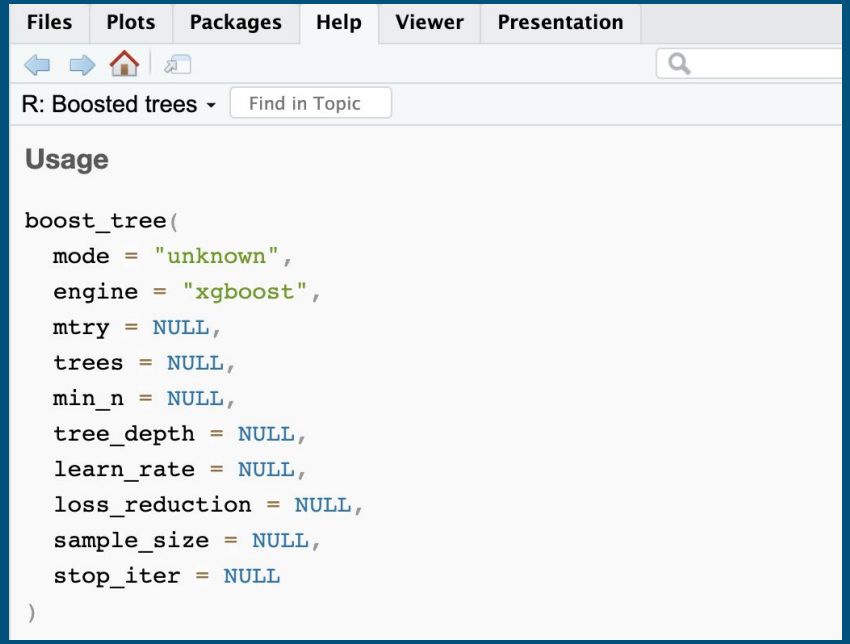
# Model definition + tuning

*Examples*

```
xgboost_spec <-
  boost_tree(mtry = tune(),
             trees = tune(),
             min_n = tune(),
             tree_depth = tune(),
             learn_rate = tune(),
             loss_reduction = tune(),
             sample_size = tune()
             ) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

We are tuning all possible model parameters.

```
?boost_tree
```



# Model definition + tuning

*Examples*

We will be running three different models to assess the performance of each of them:

- xgboost
- Random forest
- Logistic regression

```
xgboost_spec <-
  boost_tree(mtry = tune(),
             trees = tune(),
             min_n = tune(),
             tree_depth = tune(),
             learn_rate = tune(),
             loss_reduction = tune(),
             sample_size = tune()
             ) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

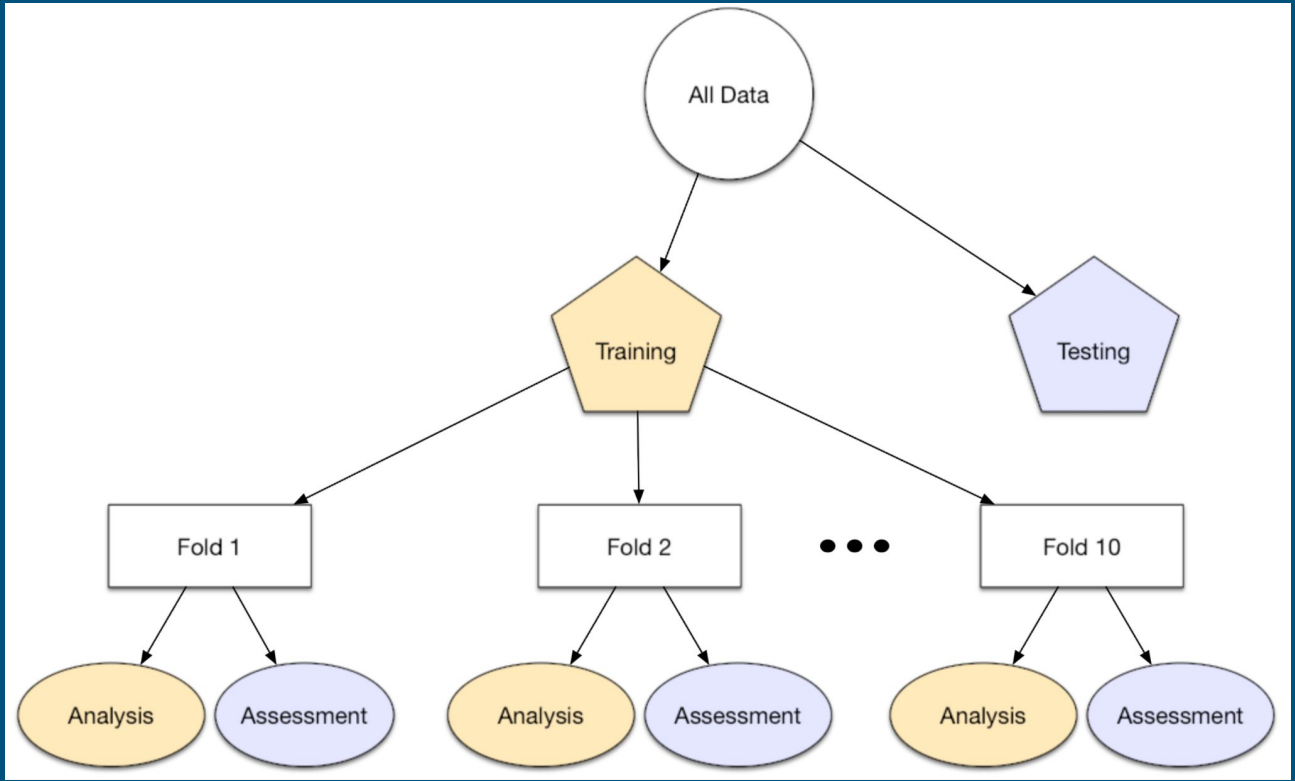
```
randomforest_spec <-
  rand_forest(mtry = tune(),
              trees = tune(),
              min_n = tune()
              ) %>%
  set_engine("randomForest") %>%
  set_mode("classification")
```

```
lr_spec <-
  logistic_reg(penalty = tune(),
               mixture = tune()
               ) %>%
  set_engine("glmnet") %>%
  set_mode("classification")
```

# Creating *workflow_set*

```
### Create list of recipes and models
recipe_list <-
  list(mean=data_rec_mean,
       knn=data_rec_knn)

model_list <-
  list(XGBoost = xgboost_spec,
       RandomForest = randomforest_spec,
       LogisticRegression = lr_spec
       )

### Ceate model set
model_set <- workflow_set(preproc = recipe_list, models = model_list, cross = T)
```

# Creating folds



# Creating folds

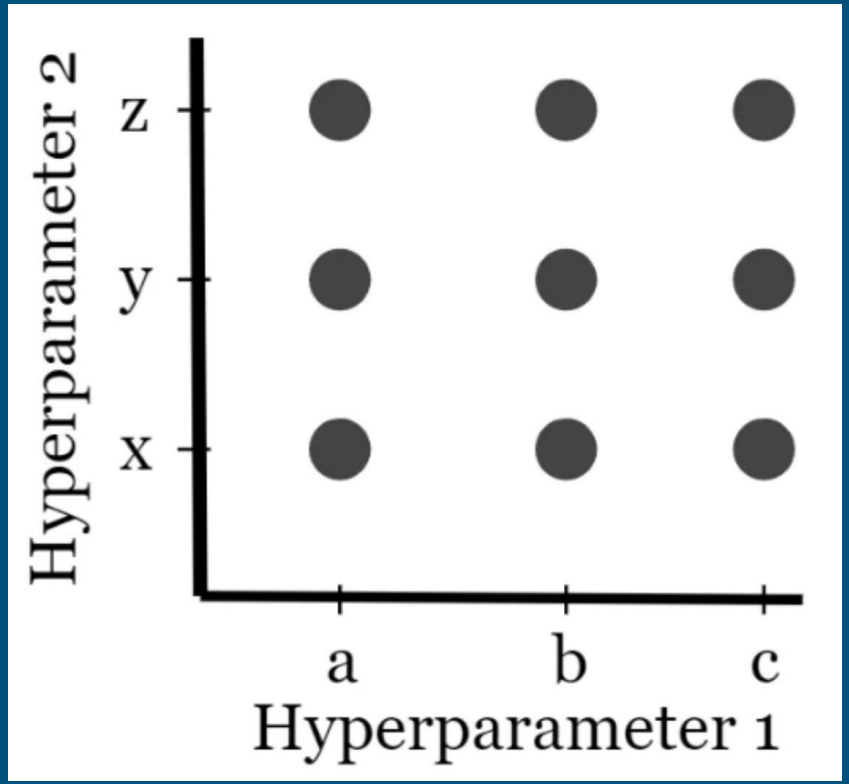
```
data_folds <- vfold_cv(data_train, v = 10, strata = judge)
```

# Fit all models with *workflow_map*

```
grid_ctrl <-
  control_grid(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )

# Parallelise

doParallel::registerDoParallel(cores = 6)
```

```
# Generate models
grid_results_impute <-
  model_set %>%
  workflow_map(
    seed = 123,
    resamples = data_folds,
    grid = 50,
    control = grid_ctrl,
    verbose = TRUE,
    metrics = metric_set(accuracy,
                         roc_auc,
                         f_meas,
                         mcc,
                         sens,
                         spec,
                         bal_accuracy)
  )
```

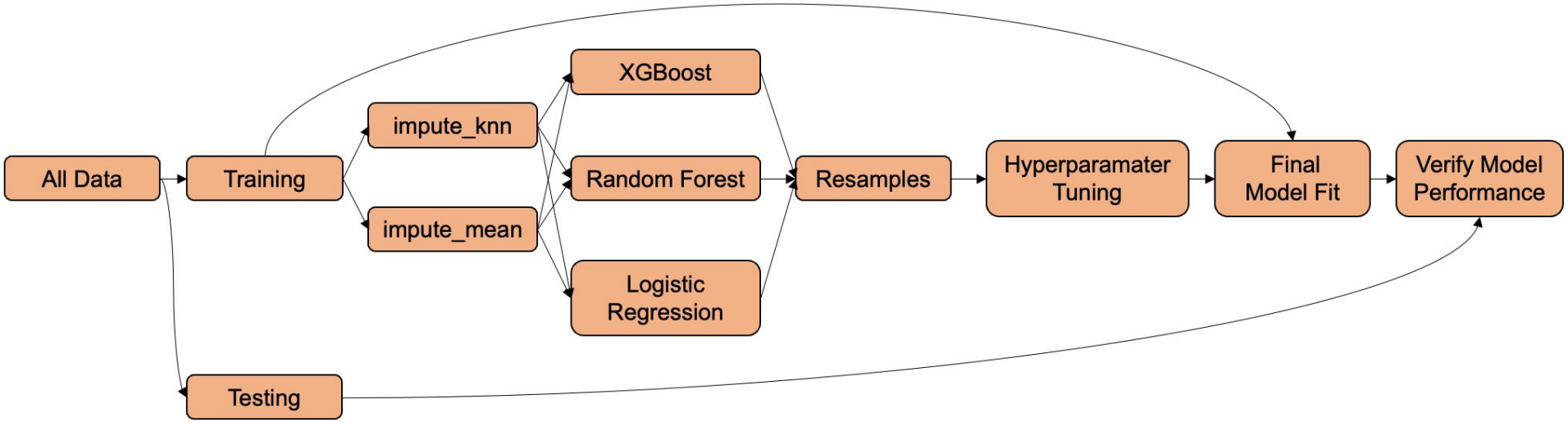
# Grid search



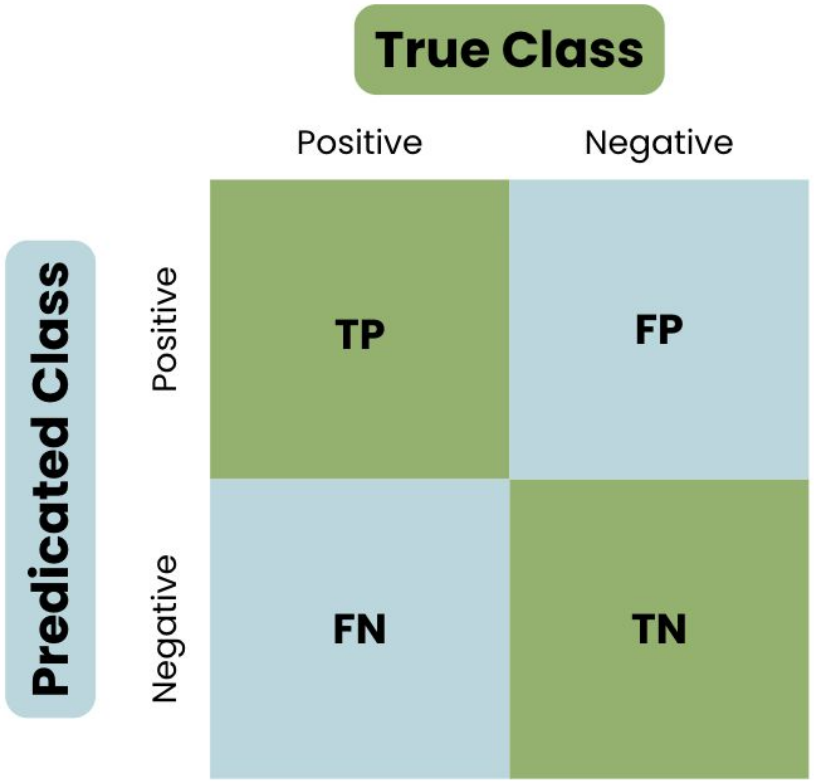

https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35

# Questions?

# Model selection & evaluation



# Metrics




$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

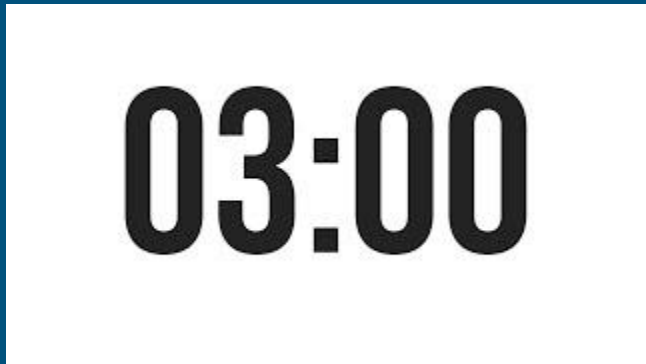
$$F\text{-}score = \frac{2 * Recall * Precision}{Recall + Precision}$$

https://www.datacamp.com/tutorial/what-is-a-confusion-matrix-in-machine-learning

# Your turn!

Go to **yardstick.tidymodels.org/reference**

- What metrics are available for classification/regression?
- Which function is measuring *F1*?



# Compare models with *collect_metrics*

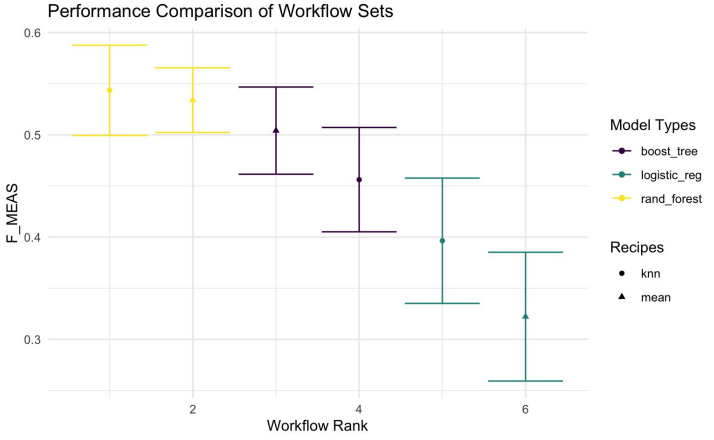
```r
metrics <- collect_metrics(grid_results_impute) %>%
  separate(wflow_id, into = c("Recipe", "Model_Type"), sep = "_", remove = F, extra = "merge") %>% # Add new columns Recipe and Model_Type, defined from wflow_id
  filter(.metric == "f_meas") %>% # Filter only results that measure performance by F1
  group_by(wflow_id) %>% # Group all models from the same combination of recipe + model together (hyperparameters differ)
  filter(mean == max(mean, na.rm = T)) %>% # For each recipe + model combination, take best hyperparameter combo
  group_by(model) %>% # This and following code removes duplicates in case there are duplicate models with the same highest mean
  select(-.config) %>%
  distinct() %>%
  ungroup() %>%
  dplyr::mutate(Workflow_Rank =  row_number(-mean),
         .metric = str_to_upper(.metric)) # This adds an extra column to rank the models by decreasing means, and then capitalises the metric.
```



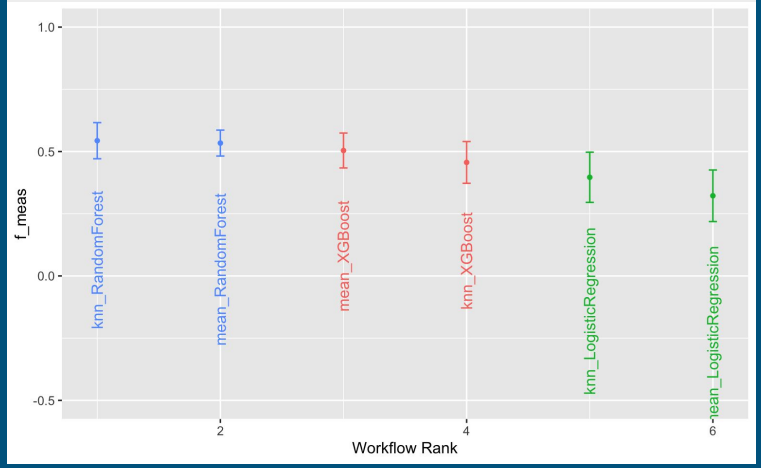
| wflow_id <chr> | Recipe <chr> | Model_Type <chr> | preproc <chr> | model <chr> | .metric <chr> | .estimator <chr> | mean <dbl> | n <int> | std_err <dbl> |
|---|---|---|---|---|---|---|---|---|---|
| mean_XGBoost | mean | XGBoost | recipe | boost_tree | F_MEAS | binary | 0.5041157 | 10 | 0.04263572 |
| mean_RandomForest | mean | RandomForest | recipe | rand_forest | F_MEAS | binary | 0.5339394 | 10 | 0.03167246 |
| mean_LogisticRegression | mean | LogisticRegression | recipe | logistic_reg | F_MEAS | binary | 0.3221749 | 8 | 0.06298290 |
| knn_XGBoost | knn | XGBoost | recipe | boost_tree | F_MEAS | binary | 0.4562030 | 10 | 0.05101662 |
| knn_RandomForest | knn | RandomForest | recipe | rand_forest | F_MEAS | binary | 0.5435447 | 10 | 0.04414735 |
| knn_LogisticRegression | knn | LogisticRegression | recipe | logistic_reg | F_MEAS | binary | 0.3964279 | 8 | 0.06127543 |

# Compare models with *collect_metrics*

```
metrics %>%
  ggplot(aes(x=Workflow_Rank, y = mean, shape = Recipe, color = model)) +
    geom_point() +
    geom_errorbar(aes(ymin = mean-std_err, ymax = mean+std_err)) +
    theme_minimal()+
    scale_colour_viridis_d() +
    labs(title = "Performance Comparison of Workflow Sets", x = "Workflow
Rank", y = str_to_upper(chosen_metric), color = "Model Types", shape =
"Recipes")
```




```
autoplot(
   grid_results_impute,
   rank_metric = chosen_metric,
   metric = chosen_metric,
   select_best = TRUE) +
   geom_text(aes(y = mean - 1/5, label = wflow_id), angle = 90, hjust = 1) +
   lims(y = c(-0.5, 1)) +
   theme(legend.position = "none")
```



# Choose best model with *select_best*

```r
# Name of best model+recipe (identified from autoplot)
best_model_id <- "knn_RandomForest"

# Select best hyperparameters of best model
best_results_parameters <-
   grid_results_impute %>%
   extract_workflow_set_result(best_model_id) %>%
   select_best(metric = "f_meas")

best_results_parameters
```



| mtry <int> | trees <int> | min_n <int> | .config <chr> |
|---|---|---|---|
| 3 | 191 | 4 | Preprocessor1_Model42 |

# Evaluate the best model on test set

```
### Now that we have our optimal model, we can finalise our workflow with the settings saved in best_tree.
final_wf <-
   grid_results_impute %>%
   extract_workflow(best_model_id) %>%
   finalize_workflow(best_results_parameters)
```

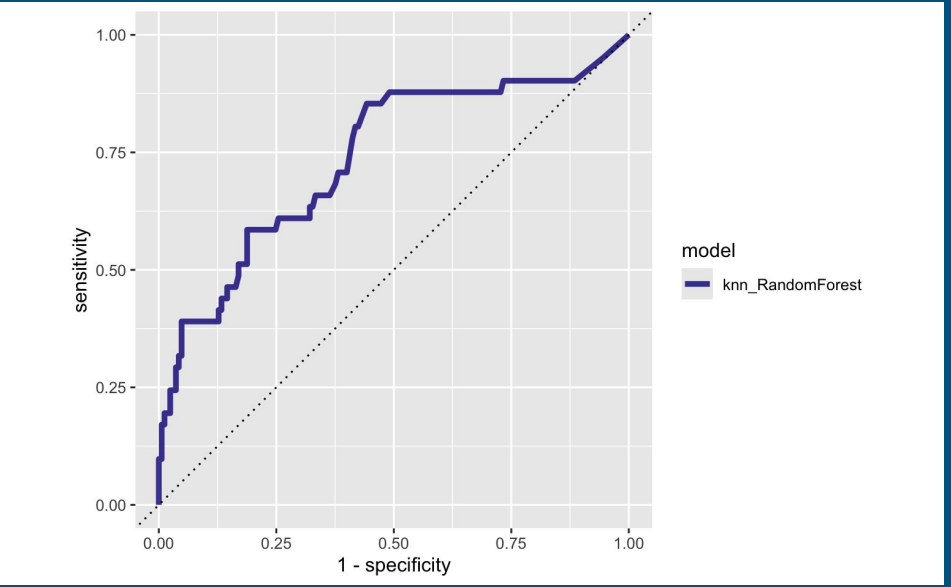
```
── Workflow ────────────────────────────────────
Preprocessor: Recipe
Model: rand_forest()

── Preprocessor ────────────────────────────────
4 Recipe Steps

• step_zv()
• step_impute_knn()
• step_YeoJohnson()
• step_normalize()

── Model ───────────────────────────────────────
Random Forest Model Specification (classification)

Main Arguments:
  mtry = 3
  trees = 191
  min_n = 4

Computational engine: randomForest
```

```
# Now that we have finalised out workflow, we can fit it against our test data
final_fit <-
  final_wf %>%
  last_fit(split = splits, metrics = all_metrics)

# Collect metrics about predictive model, like accuracy, roc-auc, brier class
final_fit %>%
  collect_metrics()
```

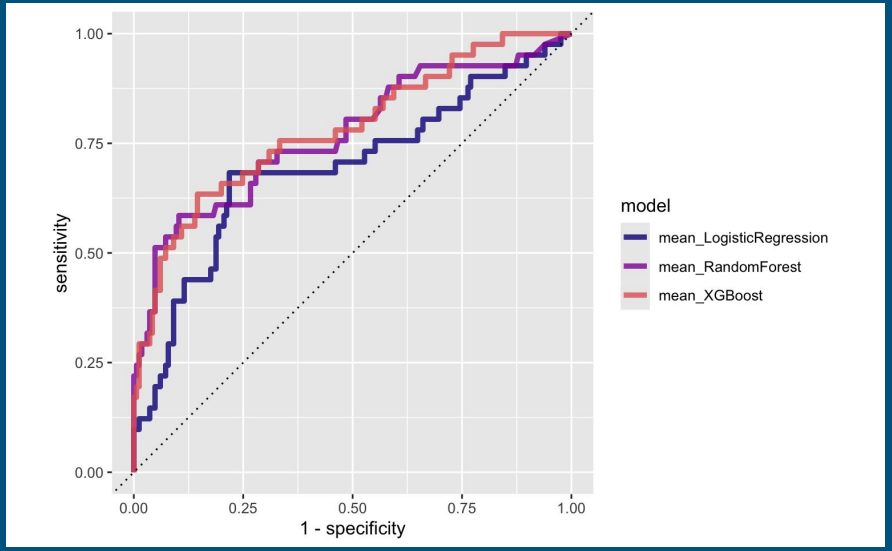
| .metric <chr> | .estimator <chr> | .estimate <dbl> | .config <chr> |
|---|---|---|---|
| accuracy | binary | 0.8203883 | Preprocessor1_Model1 |
| f_meas | binary | 0.4477612 | Preprocessor1_Model1 |
| mcc | binary | 0.3597110 | Preprocessor1_Model1 |
| sens | binary | 0.3658537 | Preprocessor1_Model1 |
| spec | binary | 0.9333333 | Preprocessor1_Model1 |
| bal_accuracy | binary | 0.6495935 | Preprocessor1_Model1 |
| roc_auc | binary | 0.7348115 | Preprocessor1_Model1 |

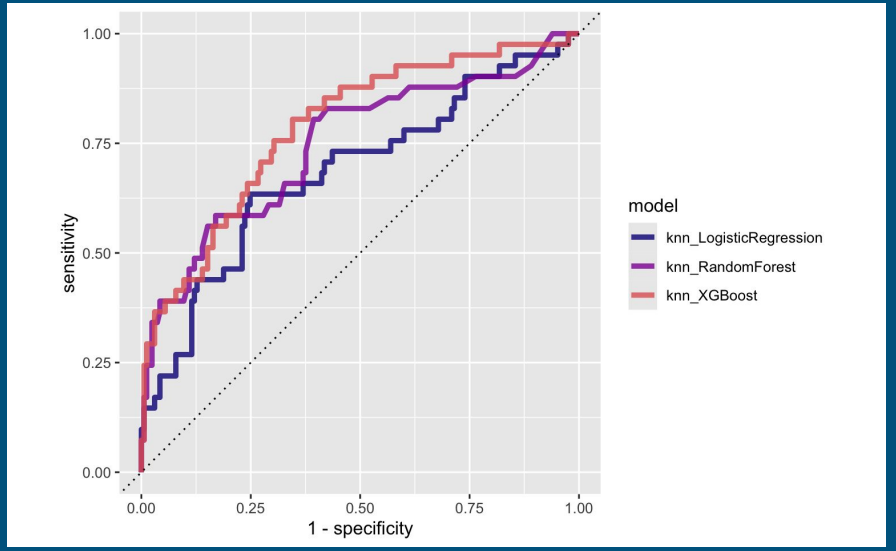
# Generate ROC plots

```r
# Plot ROC-AUC curve for knn_RandomForest
xgb_auc <- final_fit %>%
  collect_predictions() %>%
  roc_curve(judge, .pred_0) %>%
  dplyr::mutate(model = "knn_RandomForest")

xgb_auc %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```



# Generate ROC plots (extended)



# Questions?

# Bonus: Model interpretation

# What is model interpretation?

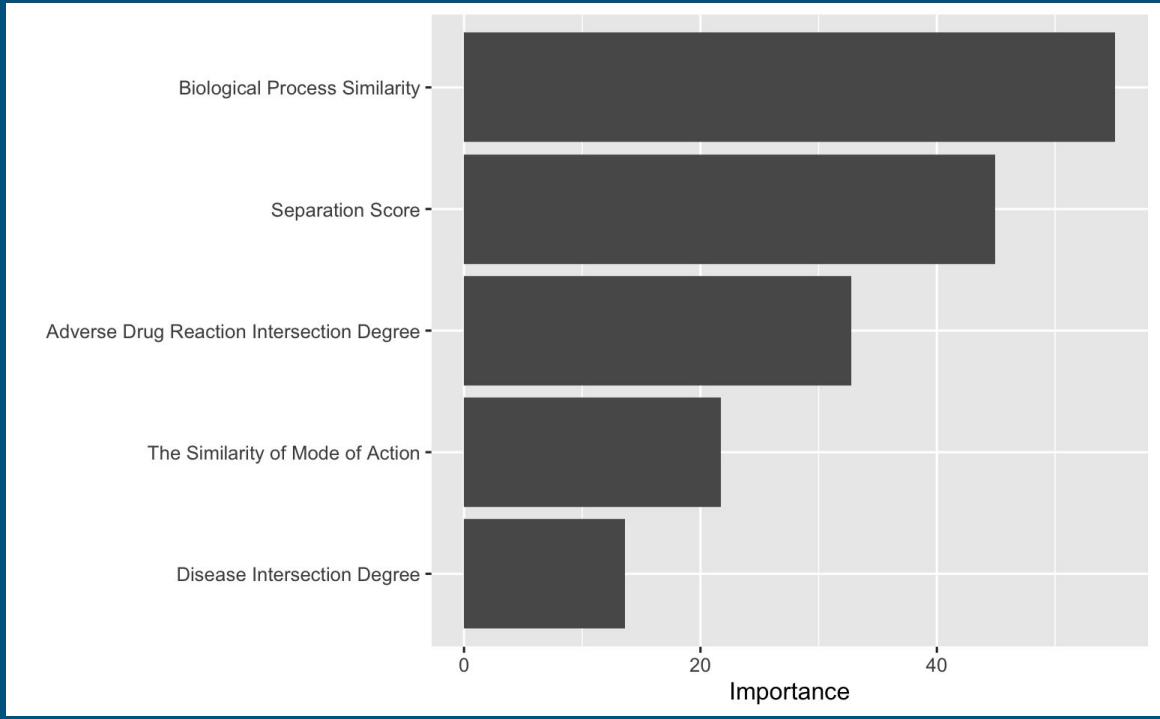
- Why a model makes a prediction?
- In other words, which features are most important in driving the predictions?

# Create explainer with *explain_tidymodels*

**Partial dependence profiles** show how the expected value of a model prediction changes as a function of a feature

# Calculate feature importance

```
final_tree <- extract_workflow(final_fit)
final_tree %>%
  extract_fit_parsnip() %>%
  vip()
```



# Questions?

# Bonus: all code in one slide!

## 1. Split data into train and test

```
# Split data, let's keep 25% for the test.
set.seed(123)
splits <- initial_split(data_clean, prop = 0.75, strata = judge)
data_train <- training(splits)
data_test  <- testing(splits)
```

## 2. Make recipe, model, and workflow

```
# We build a recipe.
data_rec_knn <-
  recipe(judge ~ ., data = data_clean) %>%
  update_role(Pubmed, DCDB, new_role = "ID") %>%
  step_zv(all_predictors()) %>%
  step_impute_knn(all_predictors(), neighbors = 5) %>%
  step_YeoJohnson(all_predictors()) %>%
  step_normalize(all_predictors())

# We build a model.
randomforest_spec <-
  rand_forest(mtry = tune(),
              trees = tune(),
              min_n = tune()
  ) %>%
  set_engine("randomForest") %>%
  set_mode("classification")

# We create the workflow.
workflow <-
  workflow() %>%
  add_model(randomforest_spec) %>%
  add_recipe(data_rec_knn)
```

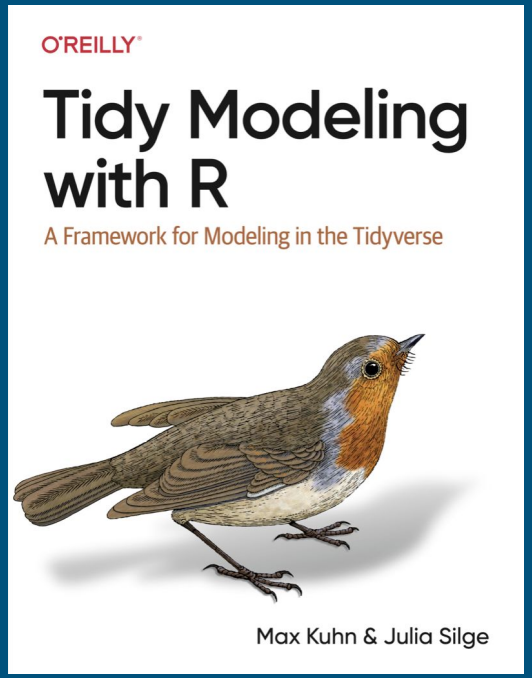
## 3. Train Model

```
# Hyperparameter tuning setting
grid_ctrl <-
  control_grid(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )

# 10-fold CV
set.seed(123)
data_folds <- vfold_cv(data_train, v = 10, strata = judge)

# Train model on train set
grid_results_impute <-
  workflow %>%
  tune_grid(resamples = data_folds, # cross-fold validation groupings
            grid = 50,
            control = grid_ctrl,
            metrics = all_metrics
  )
```

## 4. Finalise model and test against test data!

```
# Select best hyperparameters of best model
best_results_parameters <- select_best(grid_results_impute, metric = "f_meas")

### Finalise our workflow and test on unseen data.
final_wf <-
  grid_results_impute %>%
  extract_workflow(best_model_id) %>%
  finalize_workflow(best_results_parameters) %>%
  last_fit(split = splits, metrics = all_metrics) %>% #fit it against our test
  collect_metrics() # collect metrics about predictive model
```

# Other resources to check




tidymodels / workshops

www.tmwr.org

github.com/tidymodels/workshops

# Thank you!

# Happy coding!

# Hands-on coding!

1. Continue the code from *1.4 Build models*
2. Read each chunk of code and run
3. If training the models (section 1.5) is slow, remove some recipes/models
4. If you finish early, try training new recipes/models and compare