PCS Documentation

December 08, 2021

1 Domain problem formulation

What is the real-world question? This could be hypothesis-driven or discovery-based.

Why is this question interesting and important? What are the implications of better understanding this data?

Briefly describe any background information necessary to understand this problem.

Briefly describe how this question can be answered in the context of a model or analysis.

Outline the rest of the report/analysis.

2 Data

What is the data under investigation? Provide a brief overview/description of the data.

Describe how your data connects to the domain problem.

2.1 Data Collection

How was the data collected or generated (including details on the experimental design)? Be as transparent as possible so that conclusions made from this data are not misinterpreted down the road.

Describe any limitations when using the data to answer the domain problem of interest.

Where is the data stored, and how can it be accessed by others (if applicable)?

2.2 Data Splitting

TODO: add advice for possible data splits, AK getting nice figure together

Decide on the proportion of data in each split.

Decide on the "how" to split the data (e.g., random sampling, stratified sampling, etc.), and explain why this is a reasonable way to split the data.

Split the data into a training, validation, and test set.

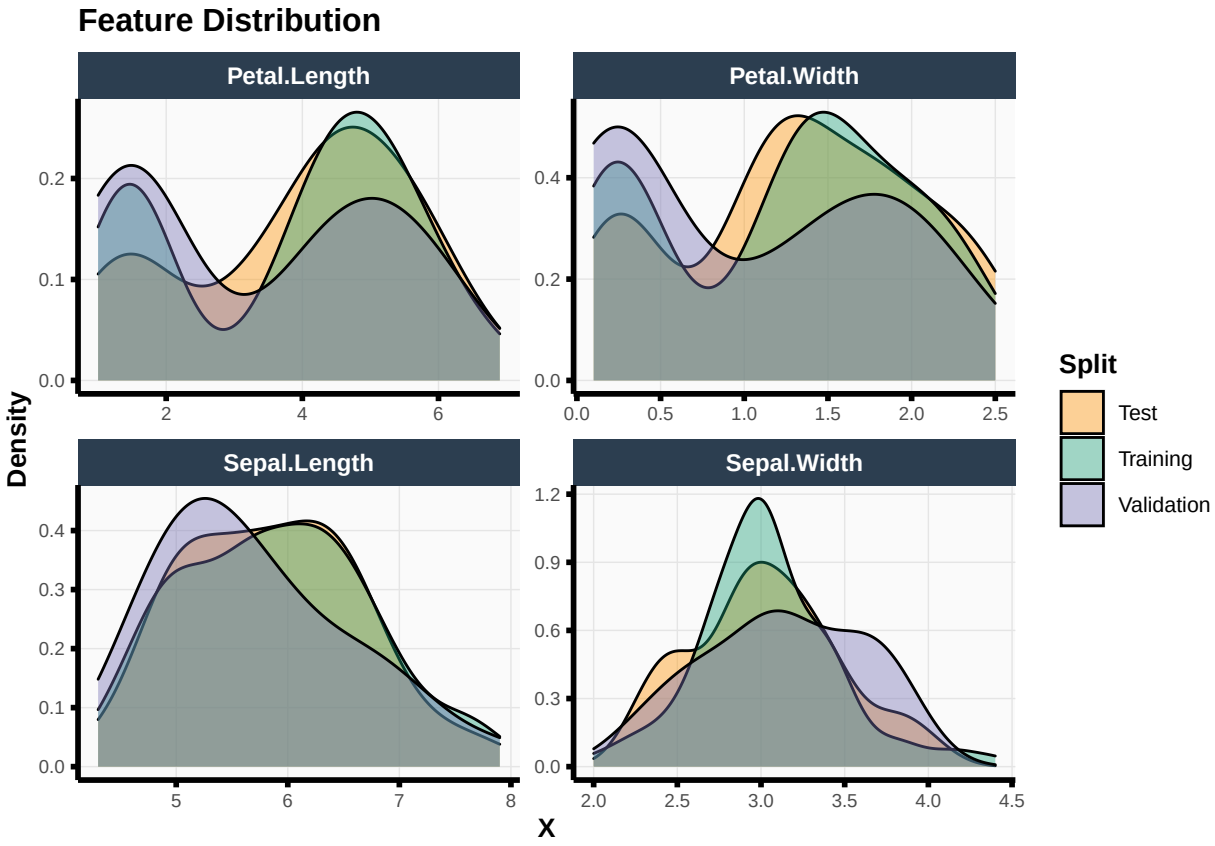
```
data_split <- basicDataSplit(X = X, y = y,
                             train_prop = params$train_prop,
                             valid_prop = params$valid_prop,
                             test_prop = params$test_prop)
Xtrain <- data_split$X$train
Xvalid <- data_split$X$validate
Xtest <- data_split$X$test
ytrain <- data_split$y$train
yvalid <- data_split$y$validate
ytest <- data_split$y$test
```

Provide summary statistics and/or figures of the three data sets to illustrate how similar (or different) they are.

2.2.1 Data Splitting Overview

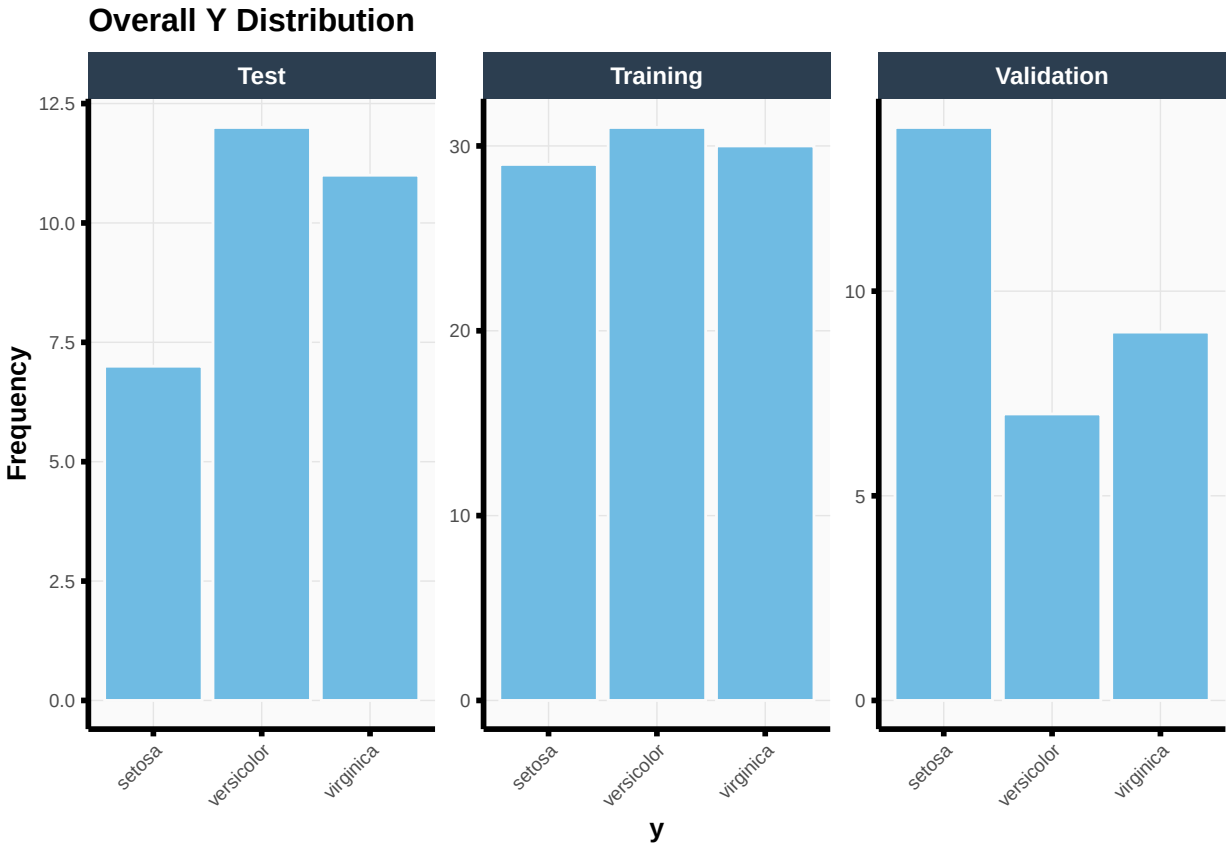
X Data Split

```
plot_X_datasplit(Xtrain, Xvalid, Xtest)
```



Y Data Split

```
plot_y_datasplit(ytrain, yvalid, ytest)
```



2.3 Data Cleaning and Preprocessing

What steps were taken to clean the data? More importantly, why was the data cleaned in this way?

Discuss all inconsistencies, problems, oddities in the data (e.g., missing data, errors in data, outliers, etc.).

Record your preprocessing steps in a way such that if someone else were to reproduce your analysis, they could easily replicate and understand your steps.

It can be helpful to include relevant plots that explain/justify the choices that were made when cleaning the data.

If more than one preprocessing pipeline is reasonable, examine the impacts of these alternative preprocessing pipelines on the final data results.

Again, be as transparent as possible. This allows others to make their own educated decisions on how best to preprocess the data.

2.4 Data Exploration

TODO: Add drag and drop feature in shiny version for other images

The main goal of this section is to give the reader a feel for what the data "looks like" at a basic level.

Provide plots that summarize the data and perhaps even plots that convey some smaller findings which ultimately motivate the main findings.

Provide additional plots representing remaining oddities after pre-processing if applicable.

Add summary statistics in accompanying tables (or in figures) for quick comparisons.

2.4.1 Training Data Overview

```
#> Number of features: 4
#> Number of training samples: 90
#> Number of NAs in training y: 0
#> Number of NAs in training X: 0
#> Number of columns in training X with NAs: 0
#> Number of constant columns in training X: 0
```

Summary Tables

```
# summary of types of features in training X
data_types(Xtrain = Xtrain, ytrain = ytrain)
```

Table 1: Frequency of column

	Factor	Numeric
X	0	4
y	1	0

```
# summary of features in training X, sorted by type
tab_ls <- data_summary(Xtrain = Xtrain, ytrain = ytrain, digits = 2, sigfig = F)
for (dt_name in names(tab_ls)) {
  simChef:::subchunkify(tab_ls[[dt_name]], i = chunk_idx,
                        other_args = "results='asis'")
  chunk_idx <- chunk_idx + 1
}
```

Table 2: Summary of Factor Variables

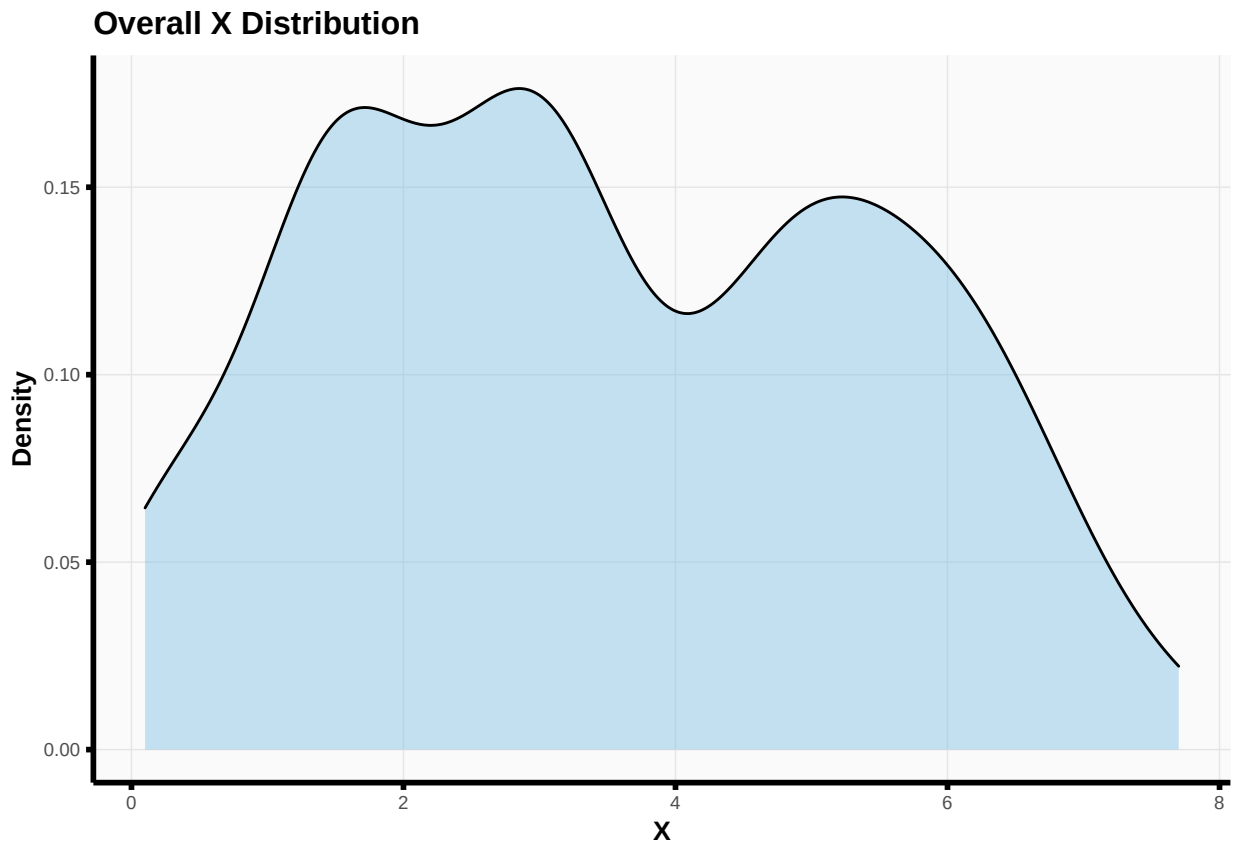
Variable Name	# Missing	% Missing	Ordered Factor	# Unique Factors	Top Factor Counts
.y	0	0.00	False	3	ver: 31, vir: 30, set: 29

Table 3: Summary of Numeric Variables

Variable Name	# Missing	% Missing	Mean	SD	Minimum	Q1	Median	Q3	Maximum
Sepal.Length	0	0.00	5.88	0.83	4.30	5.12	5.85	6.47	7.70
Sepal.Width	0	0.00	3.04	0.42	2.00	2.80	3.00	3.27	4.40
Petal.Length	0	0.00	3.82	1.76	1.10	1.60	4.40	5.10	6.90
Petal.Width	0	0.00	1.22	0.75	0.10	0.30	1.40	1.80	2.50

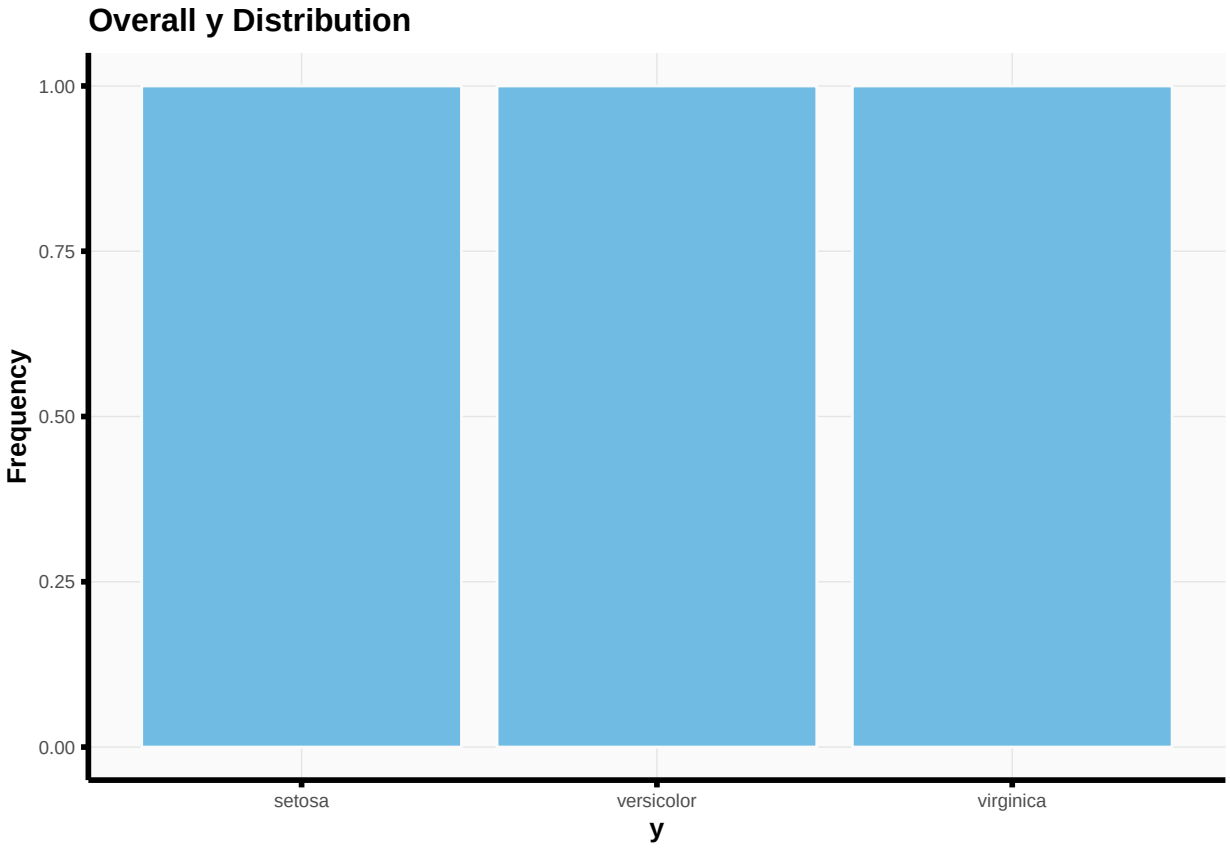
X distribution

```
# plot X distribution
plot_X_distribution(Xtrain, "density")
```



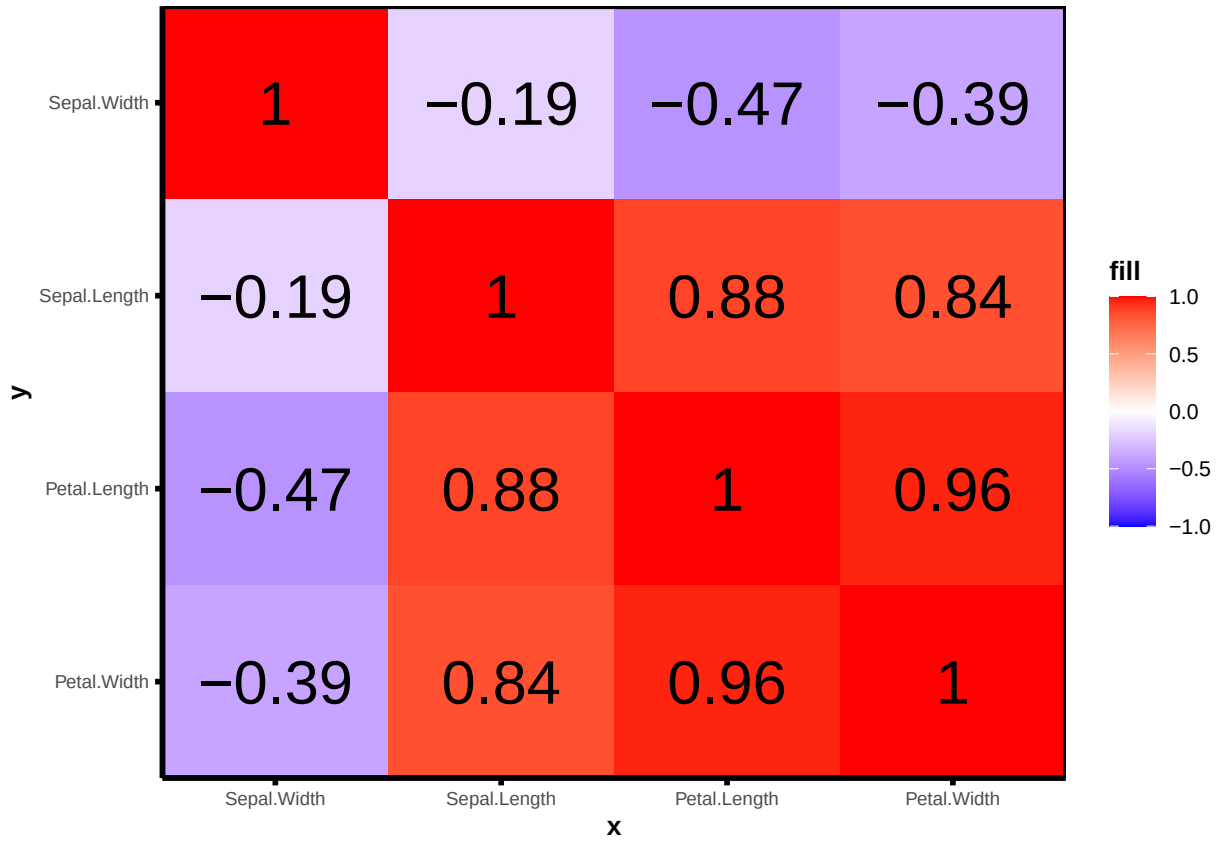
Y distribution

```
# plot y distribution
plot_y_distribution(ytrain, "bar")
```



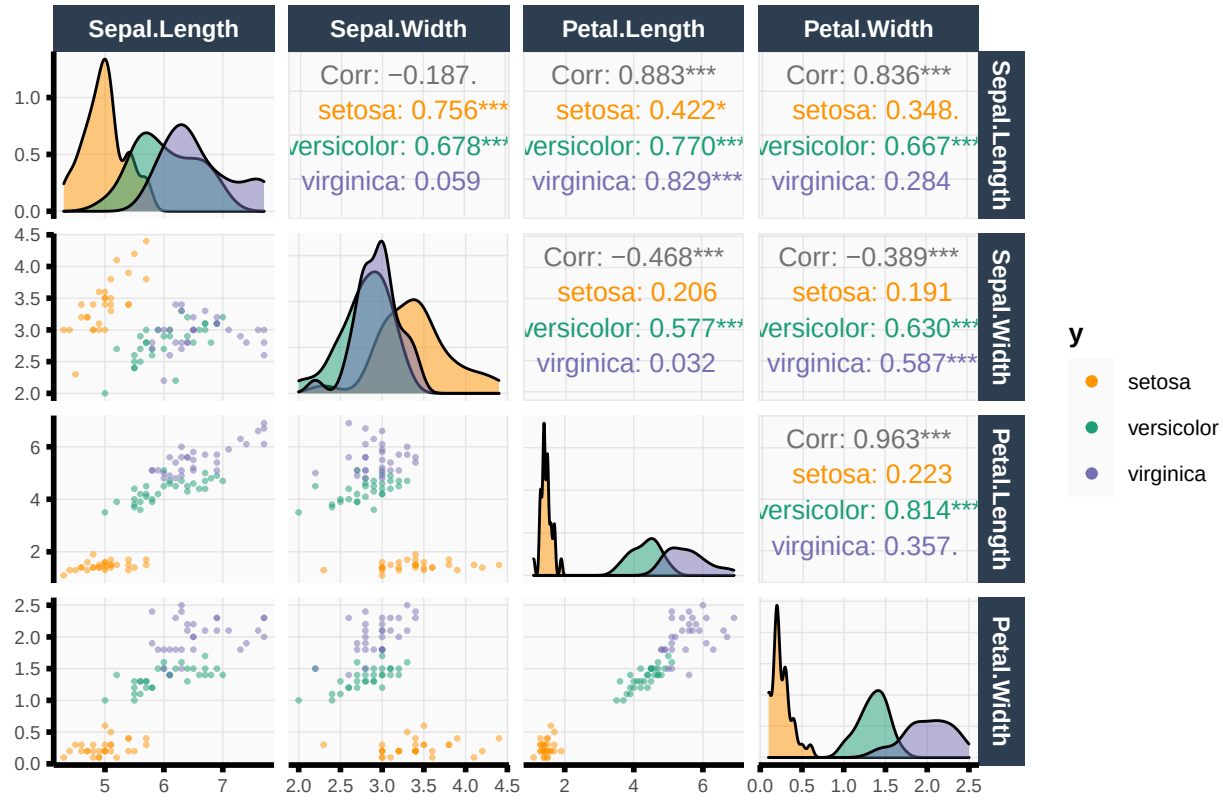
Feature Correlation

```
# correlation heatmap
plotCorHeatmap(X = Xtrain, cor_type = "pearson", clust = TRUE, text_size = 8)
```



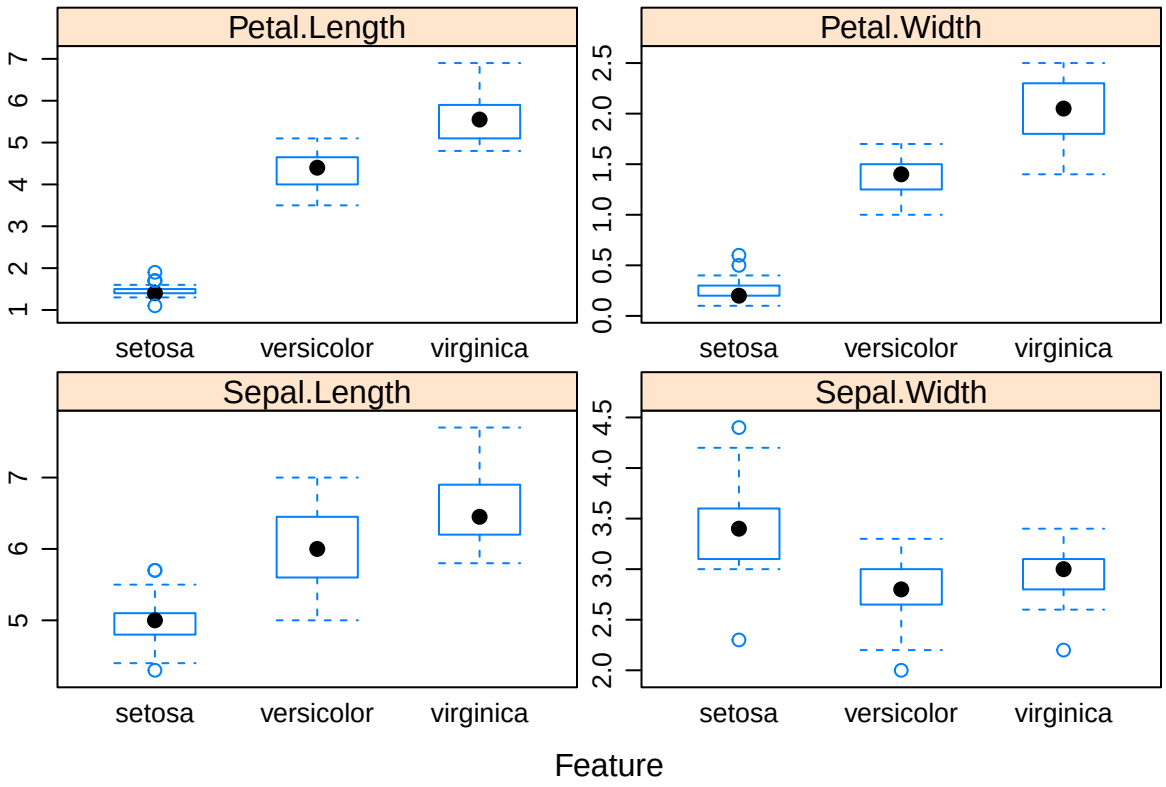
Feature Pair Plots

```
# pair plots
col_ids <- 1:min(ncol(Xtrain), 6)
plotPairs(data = Xtrain, columns = col_ids,
          color = ytrain, color_label = "y")
```



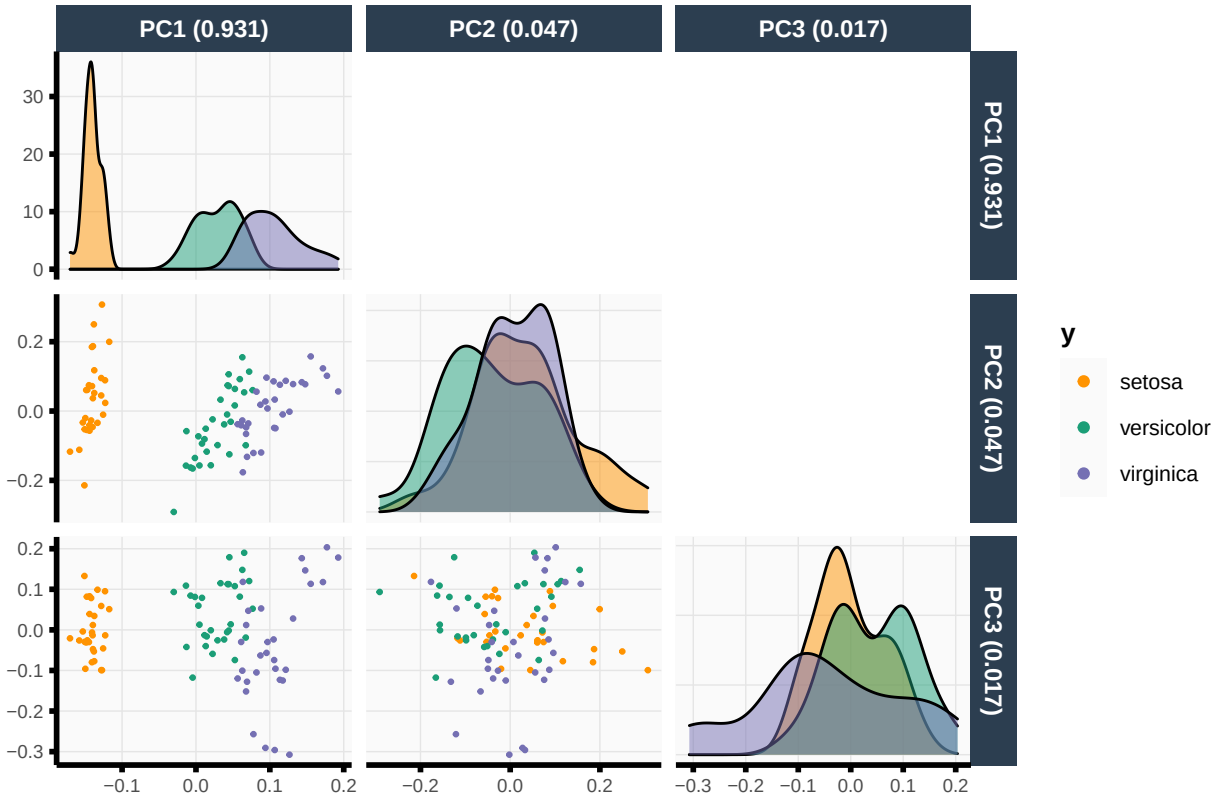
Marginal Associations

```
caret::featurePlot(x = Xtrain,
                   y = ytrain,
                   plot = if (is.factor(ytrain)) "box" else "scatter",
                   # strip = strip.custom(par.strip.text = list(cex = .7)),
                   scales = list(x = list(relation = "free"),
                                 y = list(relation = "free")))
```



PCA

```
# dimension reduction plots
plotPCA(X = Xtrain, npcs = 3, color = ytrain, color_label = "y",
        center = T, scale = FALSE)$plot
```



3 Prediction Modeling

TODO: add advice on which models to select and why

Discuss the prediction methods under consideration, and explain why these methods were chosen.

Discuss the accuracy metrics under consideration, and explain why these metrics were chosen.

Note: there should be multiple methods and metrics under consideration to paint a more holistic picture of the data. At least one method should be a baseline, common approach that may not be optimal for the problem setting, but serves as a helpful comparison.

3.1 Prediction check

Carry out the prediction pipeline, outlined above.

- Fit prediction methods on training data.
- Evaluate prediction methods on validation data.
- Compare results, and filter out poor models.

```r
## Pick models and fitting backend for caret
tr_control <- caret::trainControl(
  method = "cv",
  number = 5,
  classProbs = if (is.factor(ytrain)) TRUE else FALSE,
  summaryFunction = caret::defaultSummary,
  allowParallel = FALSE,
  verboseIter = FALSE
)

response_type <- "raw"
model_list <- list(
  ranger = list(tuneGrid = expand.grid(mtry = seq(sqrt(ncol(Xtrain)),
                                                  ncol(Xtrain) / 3,
                                                  length.out = 3),
                                       splitrule = "gini",
                                       min.node.size = 1),
                importance = "impurity",
                num.threads = 1),
  xgbTree = list(tuneGrid = expand.grid(nrounds = c(10, 25, 50, 100, 150),
                                        max_depth = c(3, 6),
                                        colsample_bytree = 0.33,
                                        eta = c(0.1, 0.3),
                                        gamma = 0,
                                        min_child_weight = 1,
                                        subsample = 0.6),
                 nthread = 1)
)

model_results <- fitCaret(Xtrain = Xtrain, ytrain = ytrain,
                          Xtest = Xvalid, ytest = yvalid,
                          model_list = model_list, tr_control = tr_control,
                          response_type = response_type)
```

```r
# (html) table of accuracy metrics
model_results$errors %>%
  simChef::pretty_DT(digits = 2, sigfig = F, rownames = FALSE,
            caption = "Validation Prediction Accuracies",
            options = list(dom = 't'))
```

```r
# (latex) table of accuracy metrics
model_results$errors %>%
  simChef::pretty_kable(digits = 2, sigfig = F, row.names = FALSE,
            caption = "Validation Prediction Accuracies",
            format = "latex")
```

Table 4: Validation Prediction Accuracies

model	Accuracy	Kappa
ranger	0.93	0.89
xgbTree	0.97	0.95

3.2 Stability check

Taking the prediction methods that pass the prediction check, perform stability analysis.

- Specify and justify the appropriate data perturbation(s).
- Re-fit the prediction methods on these perturbed data sets.
- Evaluate prediction methods on validation data.
- Assess stability across the data perturbations as well as across the various methods.
- Filter out poor models where necessary and interpret stability results.

TODO: Add results for h20, tuning needs to be fixed for h20 + tidymodels

```
n_reps <- 5 # increase for better stability measures when not testing code

# p <- progressr::progressor(steps = n_reps)
# future::plan(multisession, workers = min(n_reps, parallel::detectCores() - 1))
bootstrap_model_results <- future.apply::future_replicate(
  n = n_reps,
  expr = {
    bootstrap <- sample(1:nrow(Xtrain), nrow(Xtrain), replace = TRUE)
    Xtrain_b <- Xtrain[bootstrap, ]
    ytrain_b <- ytrain[bootstrap]

    if (params$modeling_pkg == "caret") {
      model_results_b <- fitCaret(Xtrain = Xtrain_b, ytrain = ytrain_b,
                                  Xtest = Xvalid, ytest = yvalid,
                                  model_list = model_list,
                                  tr_control = tr_control,
                                  response_type = response_type)
    } else if (params$modeling_pkg == "tidymodels") {
      model_results_b <- fitTidyModels(Xtrain = Xtrain_b, ytrain = ytrain_b,
                                       Xtest = Xvalid, ytest = yvalid,
                                       model_list = model_list, kfolds = kfolds)
    } else if (params$modeling_pkg == "h2o") {
      model_results_b <- fith2o(Xtrain = Xtrain_b, ytrain = ytrain_b,
                                Xtest = Xvalid, ytest = yvalid,
                                model_list = model_list)
    }
    return(model_results_b)
  },
  simplify = FALSE
)

bootstrap_model_errs <- purrr::map_dfr(bootstrap_model_results, "errors",
                                       .id = "bootstrap_id")
bootstrap_model_preds <- purrr::map(bootstrap_model_results, "predictions",
                                    .id = "bootstrap_id")
bootstrap_model_imps <- purrr::map(bootstrap_model_results, "importance",
                                   .id = "bootstrap_id") %>%
  dplyr::bind_rows(.id = "bootstrap_id")
```

```
bootstrap_model_errs_summary <- bootstrap_model_errs %>%
  dplyr::group_by(model) %>%
  dplyr::summarise(dplyr::across(-bootstrap_id, list(mean = mean, sd = sd))) %>%
  setNames(stringr::str_replace(colnames(.), "_", " "))
```

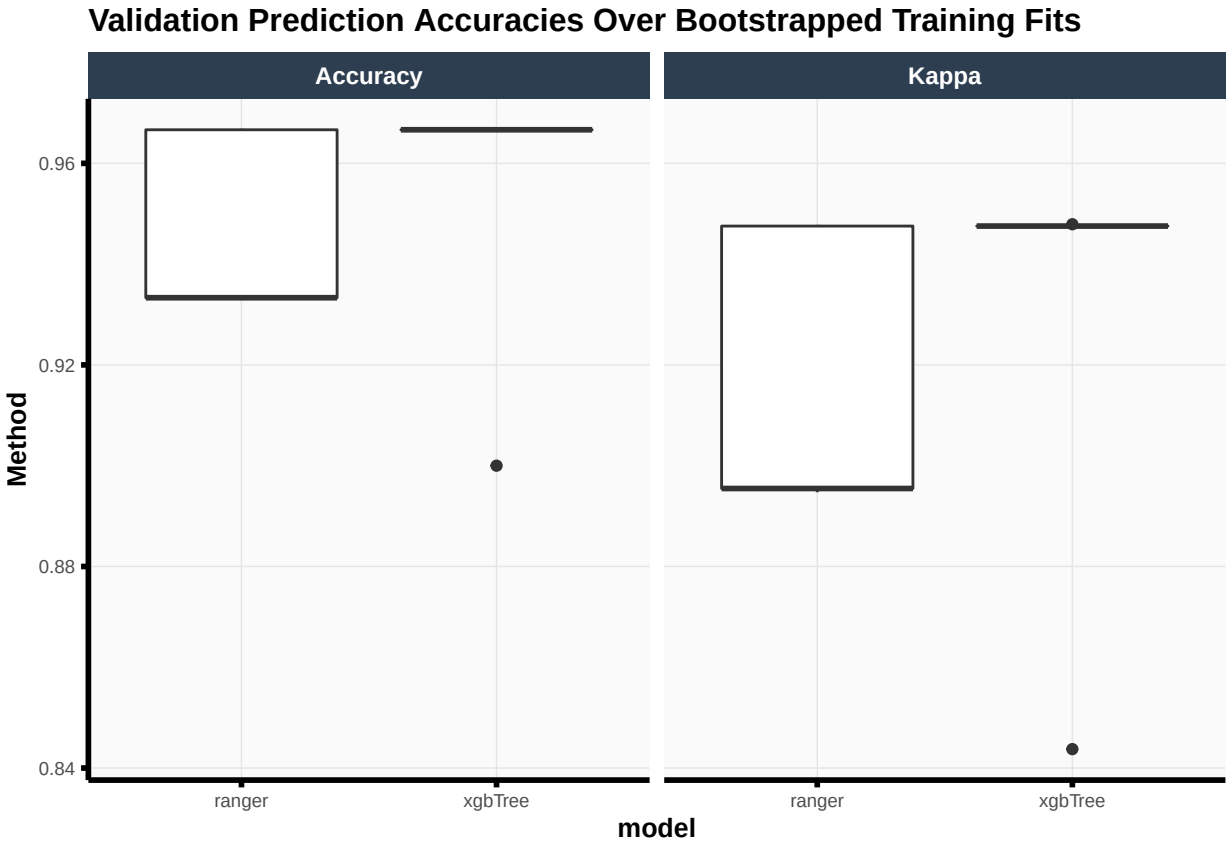
```
# (html) table of accuracy metrics
bootstrap_model_errs_summary %>%
  simChef::pretty_DT(
    digits = 2, sigfig = F, rownames = FALSE,
    caption = "Validation Prediction Accuracies Over Bootstrapped Training Fits",
    bold_function = ". == max(., na.rm = TRUE)", bold_margin = 2,
    bold_scheme = c(F, T, F, T, F),
    options = list(dom = 't')
  )
```

```
# (latex) table of accuracy metrics
bootstrap_model_errs_summary %>%
  simChef::pretty_kable(
    digits = 2, sigfig = F, row.names = FALSE,
    caption = "Validation Prediction Accuracies Over Bootstrapped Training Fits",
    bold_function = ". == max(., na.rm = TRUE)", bold_margin = 2,
    bold_scheme = c(F, T, F, T, F),
    format = "latex"
  )
```

Table 5: Validation Prediction Accuracies Over Bootstrapped Training Fits

model	Accuracy mean	Accuracy sd	Kappa mean	Kappa sd
ranger	0.95	0.02	0.92	0.03
xgbTree	**0.95**	0.03	**0.93**	0.05

```
# boxplots
bootstrap_model_errs %>%
  tidyr::pivot_longer(cols = -c(model, bootstrap_id),
                      names_to = "Metric", values_to = "Value") %>%
  plotBoxplot(x_str = "Value", y_str = "model", horizontal = FALSE) +
  ggplot2::facet_wrap(~ Metric) +
  ggplot2::labs(x = "Method",
                title = "Validation Prediction Accuracies Over Bootstrapped Training Fits")
```



3.3 Interpretability

For the models that pass the prediction and stability checks, extract the important features in the predictive models that are stable across both data and model perturbations. Determining the importance of a feature can be method dependent.

Full Model (without stability)

Table

```
simChef::pretty_DT(model_results$importance, digits = 2, sigfig = F,
                   caption = "Variable Importances")
```

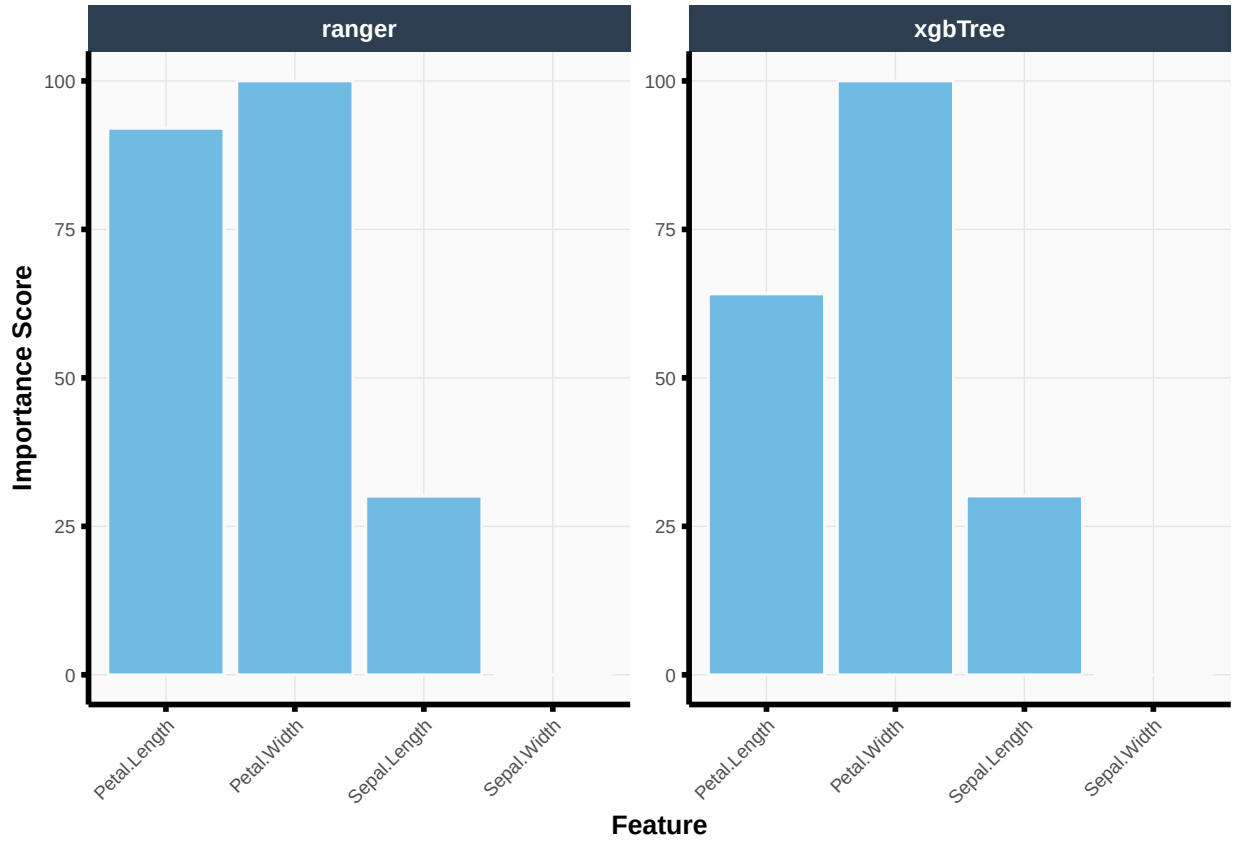
```
simChef::pretty_kable(model_results$importance, digits = 2, sigfig = F,
                      caption = "Variable Importances", format = "latex")
```

Table 6: Variable Importances

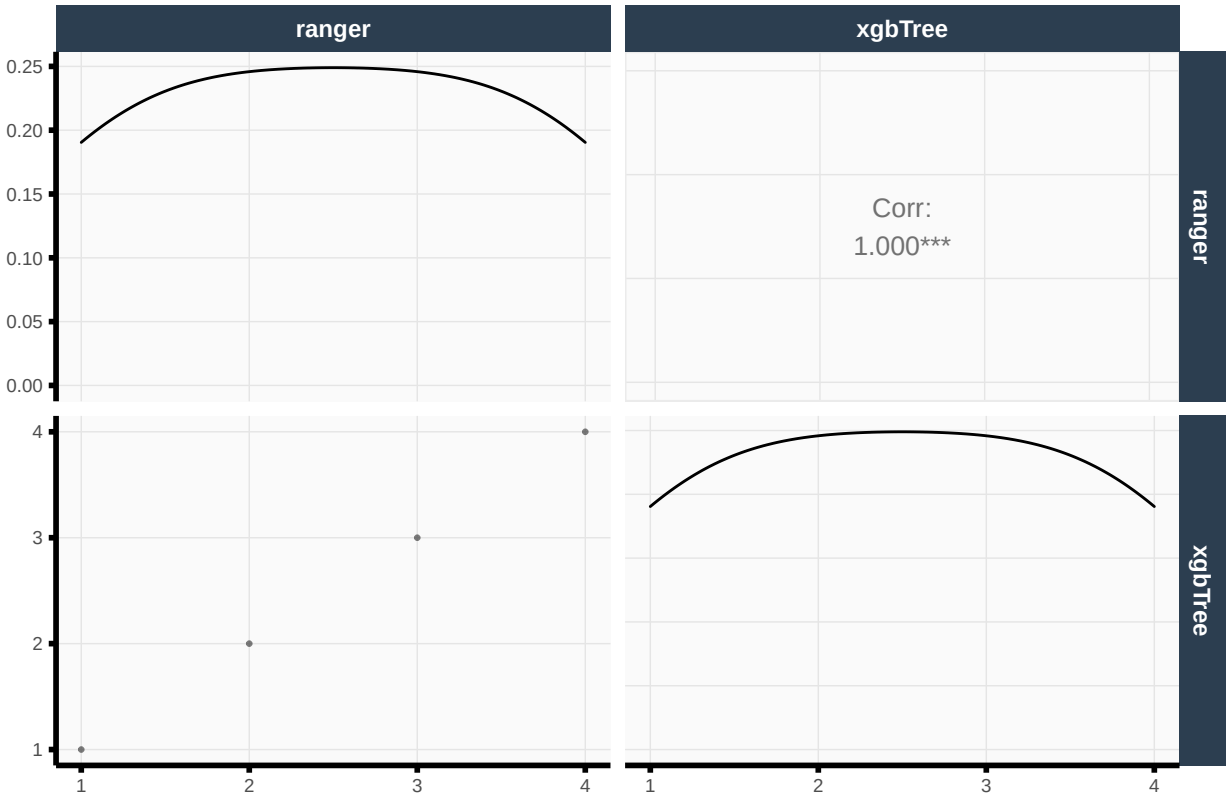
model	variable	importance
ranger	Sepal.Length	30.08
ranger	Sepal.Width	0.00
ranger	Petal.Length	92.03
ranger	Petal.Width	100.00
xgbTree	Petal.Width	100.00
xgbTree	Petal.Length	64.13
xgbTree	Sepal.Length	30.11
xgbTree	Sepal.Width	0.00

Plots

```
plotFeatureImportance(model_results$importance,
                      use_rankings = FALSE,
                      use_facets = TRUE,
                      interactive = FALSE)
```




```
plotFeatureImportancePair(model_results$importance,
                          use_rankings = TRUE,
                          interactive = FALSE)
```



Bootstrapped Model (with stability)

Table

```
bootstrap_model_imps_summary <- bootstrap_model_imps %>%
  dplyr::group_by(model, variable) %>%
  dplyr::summarise(`Mean Importance` = mean(importance),
                   `Median Importance` = median(importance),
                   `SD Importance` = sd(importance),
                   `Min Importance` = min(importance),
                   `Max Importance` = max(importance))
simChef::pretty_DT(bootstrap_model_imps_summary,
                   digits = 2, sigfig = F,
                   caption = "Summary of variable importances across bootstrapped models")
```

```
bootstrap_model_imps_summary <- bootstrap_model_imps %>%
  dplyr::group_by(model, variable) %>%
  dplyr::summarise(`Mean Importance` = mean(importance),
                   `Median Importance` = median(importance),
                   `SD Importance` = sd(importance),
                   `Min Importance` = min(importance),
                   `Max Importance` = max(importance))
simChef::pretty_kable(bootstrap_model_imps_summary,
```

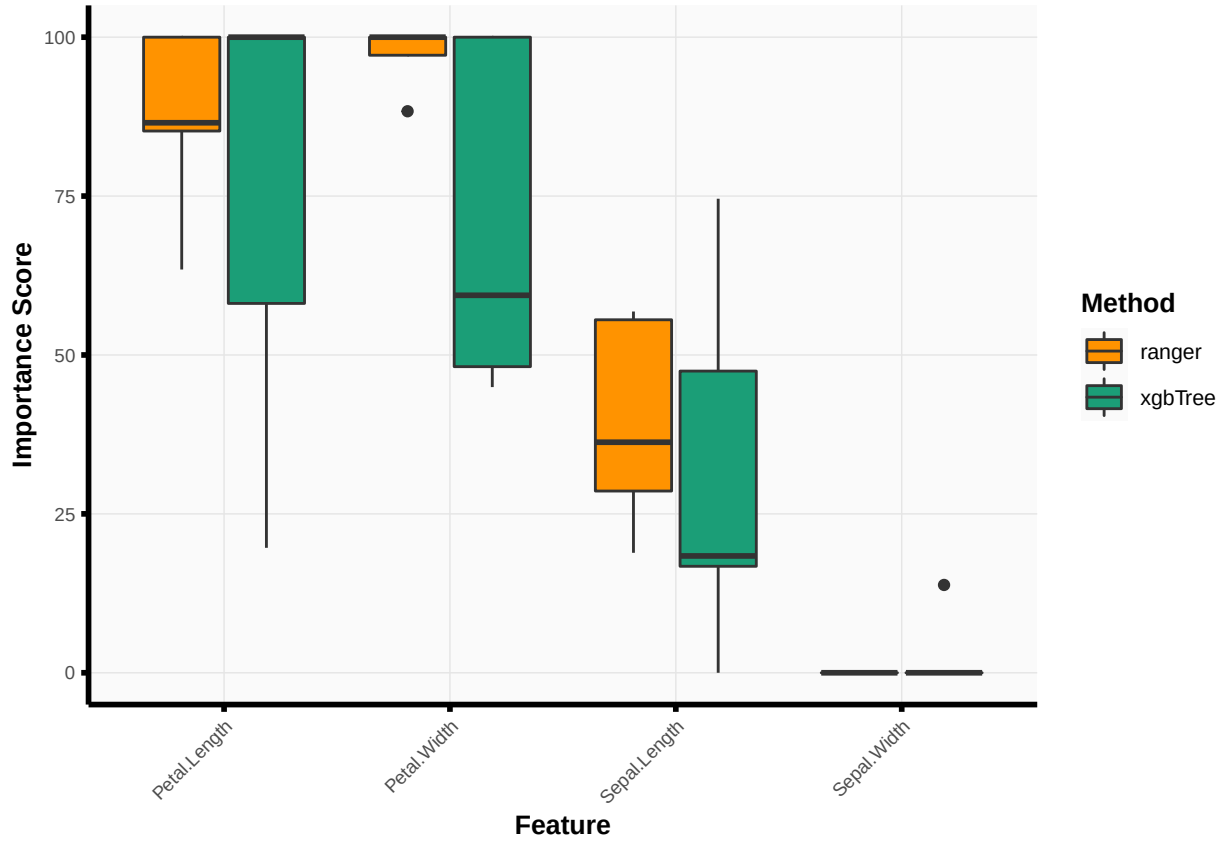
```
                    digits = 2, sigfig = F,
                    caption = "Summary of variable importances across bootstrapped models",
                    format = "latex")
```

Table 7: Summary of variable importances across bootstrapped models

model	variable	Mean Importance	Median Importance	SD Importance	Min Importance	Max Importance
ranger	Petal.Length	87.04	86.53	14.97	63.45	100.00
ranger	Petal.Width	97.10	100.00	5.05	88.35	100.00
ranger	Sepal.Length	39.22	36.27	16.67	18.88	56.84
ranger	Sepal.Width	0.00	0.00	0.00	0.00	0.00
xgbTree	Petal.Length	75.55	100.00	36.13	19.66	100.00
xgbTree	Petal.Width	70.50	59.39	27.46	44.95	100.00
xgbTree	Sepal.Length	31.44	18.38	29.55	0.00	74.58
xgbTree	Sepal.Width	2.76	0.00	6.18	0.00	13.82

Plots

```
plotFeatureImportanceStability(bootstrap_model_imps,
                               use_rankings = FALSE,
                               use_facets = FALSE,
                               interactive = FALSE)
```



4 Main Results

Interpret and summarize the prediction and stability results.

Evaluate pipeline on test data.

```
Xtrain_final <- dplyr::bind_rows(Xtrain, Xvalid)
ytrain_final <- c(ytrain, yvalid)
if (params$modeling_pkg == "caret") {
  final_model_results <- fitCaret(Xtrain = Xtrain_final, ytrain = ytrain_final,
                                  Xtest = Xtest, ytest = ytest,
                                  model_list = model_list,
                                  tr_control = tr_control,
                                  response_type = response_type)
} else if (params$modeling_pkg == "tidymodels") {
  final_model_results <- fitTidyModels(Xtrain = Xtrain_final,
                                       ytrain = ytrain_final,
                                       Xtest = Xtest, ytest = ytest,
                                       model_list = model_list, kfolds = kfolds)
} else if (params$modeling_pkg == "h2o") {
  final_model_results <- fith2o(Xtrain = Xtrain_final, ytrain = ytrain_final,
                                Xtest = Xtest, ytest = ytest,
                                model_list = model_list)
}
```

Summarize test set prediction and/or interpretability results.

```
# (html) table of accuracy metrics
final_model_results$errors %>%
  simChef::pretty_DT(digits = 2, sigfig = F, rownames = FALSE,
            caption = "Test Prediction Accuracies",
            options = list(dom = 't'))
```

```
# (latex) table of accuracy metrics
final_model_results$errors %>%
    simChef::pretty_kable(digits = 2, sigfig = F, row.names = FALSE,
              caption = "Test Prediction Accuracies", format = "latex")
```

Table 8: Test Prediction Accuracies

model	Accuracy	Kappa
ranger	0.97	0.95
xgbTree	0.97	0.95

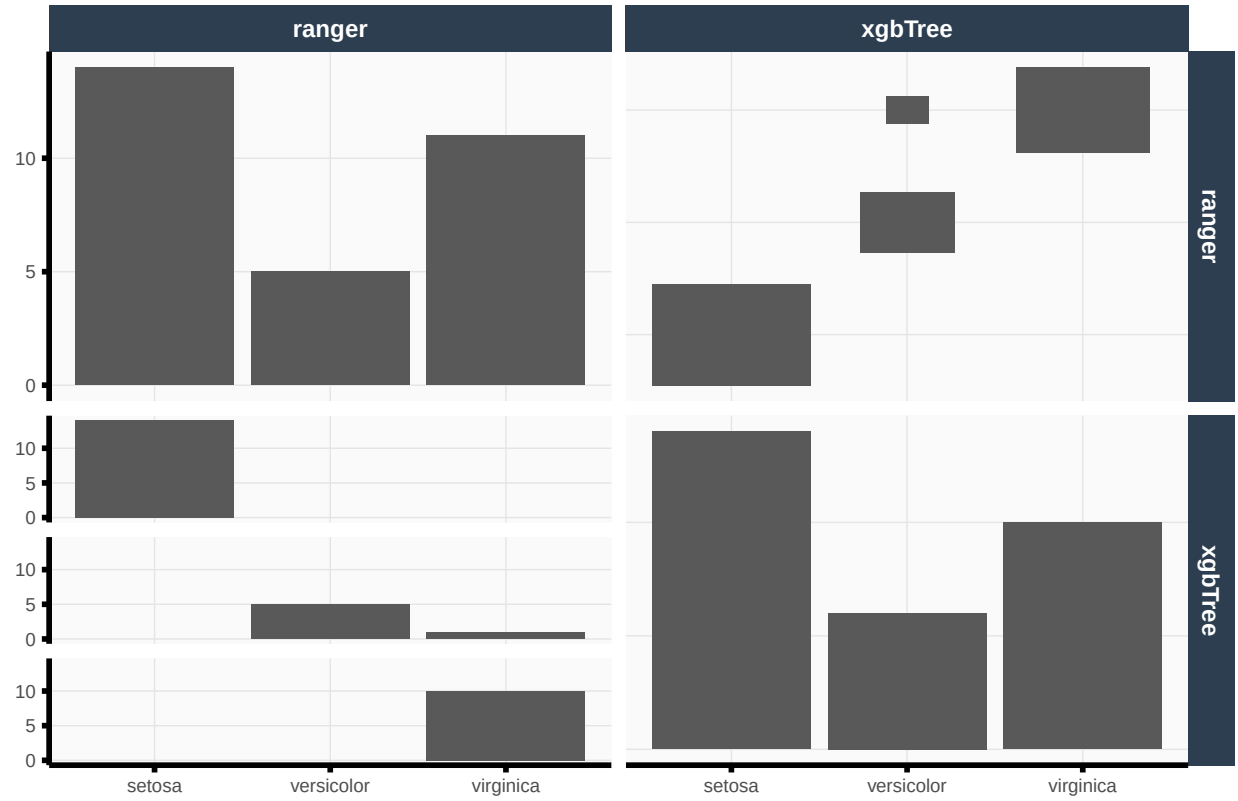
5 Post hoc analysis

Move beyond the global prediction accuracy metrics and dive deeper into individual-level predictions for the validation and/or test set, i.e., provide a more "local" analysis.

- Examine any points that had poor predictions.
- Examine differences between prediction methods.

TODO: Tiffany - Add examples with interesting observations of prediction accuracy metrics so the user knows what to look for.

```
plotPairs(model_results$predictions,
          columns = 1:ncol(model_results$predictions))
```



6 Conclusions

Reiterate main findings, note any caveats, and clearly translate findings/analysis back to the domain problem context.