iris =

	sepal_length	sepal_width	petal_length	petal_width	species
1	5.1	3.5	1.4	0.2	"Iris-setosa"
2	4.9	3.0	1.4	0.2	"Iris-setosa"
3	4.7	3.2	1.3	0.2	"Iris-setosa"
4	4.6	3.1	1.5	0.2	"Iris-setosa"
5	5.0	3.6	1.4	0.2	"Iris-setosa"
6	5.4	3.9	1.7	0.4	"Iris-setosa"
7	4.6	3.4	1.4	0.3	"Iris-setosa"
8	5.0	3.4	1.5	0.2	"Iris-setosa"
9	4.4	2.9	1.4	0.2	"Iris-setosa"
10	4.9	3.1	1.5	0.1	"Iris-setosa"
more					
150	5.9	3.0	5.1	1.8	"Iris-virginica"

```
iris = @pipe CSV.read(download("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"), DataFrame, header=["sepal_length", "sepal_width", "petal_length", "petal_width", "species"]) |> coerce!(_, :species => Multiclass)
```

names	scitypes	types
sepal_length	Continuous	Float64
sepal_width	Continuous	Float64
petal_length	Continuous	Float64
petal_width	Continuous	Float64
species	Multiclass{3}	CategoricalValue{String15, UInt32}

```
schema(iris)
```

0

```
isnothing.(iris) |> Matrix |> sum
```

y_col = "species"

```
y_col = "species"
```

splitting the dataset

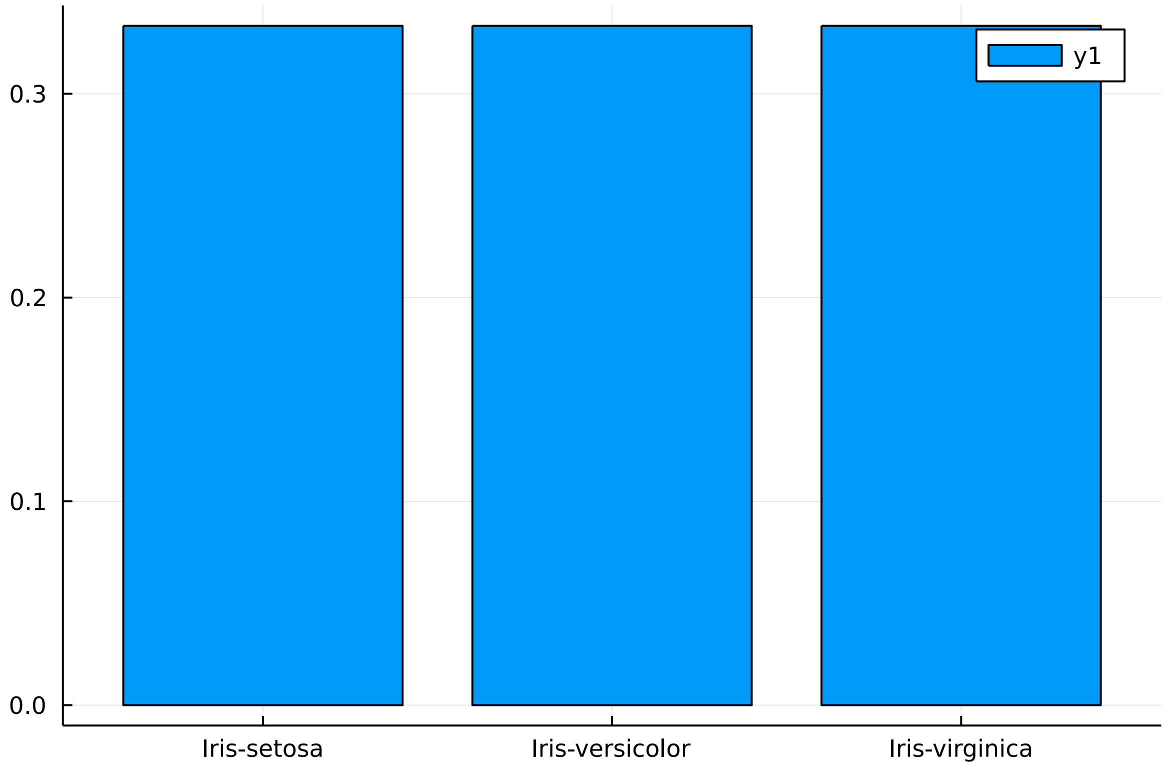
```
train, test = stratifiedobs(row->row[:species], iris); nothing
```

we don't look at testset

	variable	sepal_length	sepal_width	petal_length	petal_width	specie
1	"mean"	5.85143	3.0181	3.76095	1.19143	nothing
2	"std"	0.859686	0.442411	1.795	0.754488	nothing
3	"min"	4.3	2.0	1.0	0.1	"Iris-setosa"
4	"q25"	5.1	2.7	1.5	0.3	nothing
5	"median"	5.8	3.0	4.4	1.3	nothing
6	"q75"	6.4	3.3	5.1	1.8	nothing
7	"max"	7.9	4.2	6.9	2.5	"Iris-virginica"
8	"nunique"	nothing	nothing	nothing	nothing	3
9	"nmissing"	0	0	0	0	0
10	"first"	5.5	4.2	1.4	0.2	"Iris-setosa"
11	"last"	5.4	3.7	1.5	0.2	"Iris-setosa"
12	"eltype"	Float64	Float64	Float64	Float64	CategoricalValue{St

```
@pipe describe(train, :all) |> permutedims(_, 1)
```

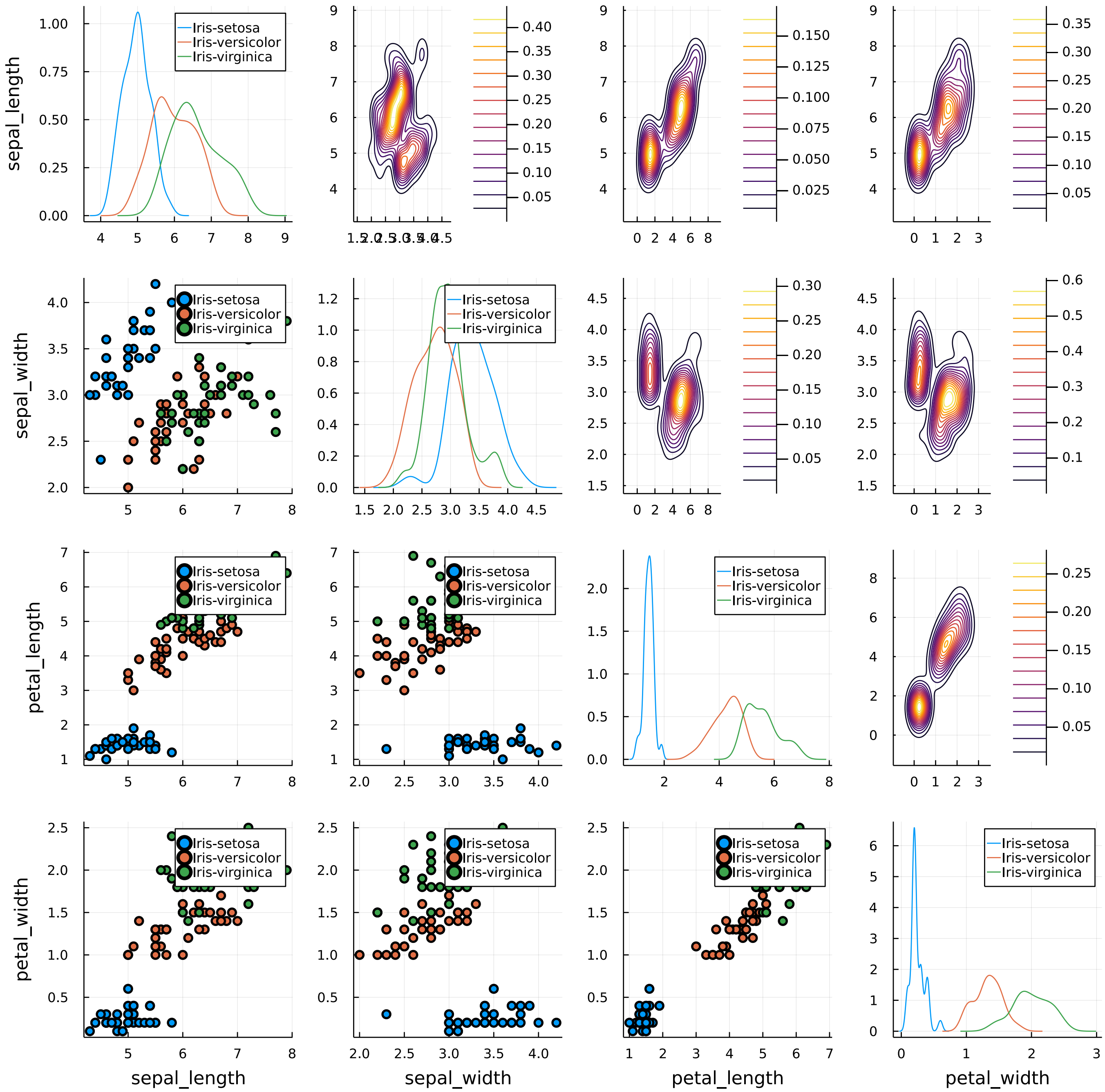
but, it's okay to look at the population distribution of the entire dataset(including testset)




```
let
    species_dist = countmap(iris.species)
    species_dist = convert(Dict{eltype(keys(species_dist)), Float64}, species_dist)
    for (species, count) ∈ species_dist
        species_dist[species] /= size(iris, 1)
    end
    bar(species_dist)
end
```

population is normally distributed: no target imbalance

EDA




```
begin
    features = names(train)[1:end-1]
    n = length(features)
    ps = fill(Plots.plot(), n, n)

    for i ∈ 1:n, j ∈ 1:n
        x, y = train[!, features[i]], train[!, features[j]]
        species = train[!, :species]
        ps[i, j] = if j < i
            dens = KernelDensity.kde((x, y))
            Plots.plot(dens)
        elseif j > i
            Plots.plot(x, y, seriestype=:scatter, group=species)
        else
            Plots.plot(x, y, seriestype=:density, group=species)
        end
```

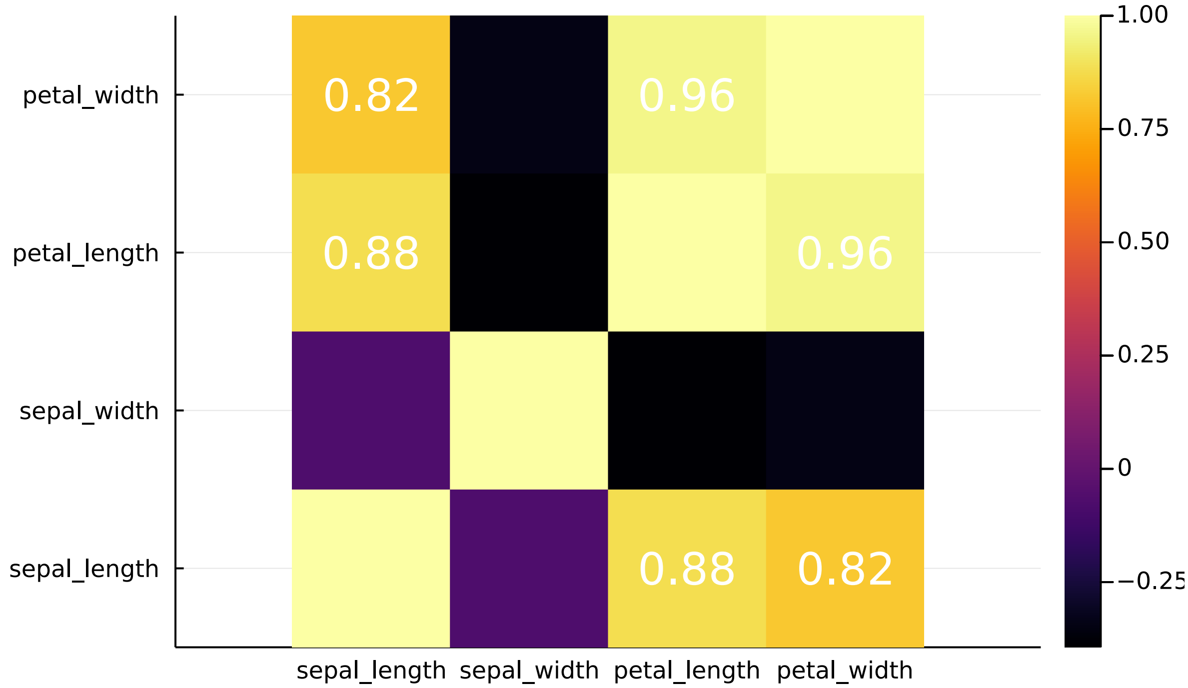
```
        i == 1 && Plots.ylabel!(ps[i, j], features[j])
        j == n && Plots.xlabel!(ps[i, j], features[i])
    end
    Plots.plot(ps..., layout = (n, n), size=(1000,1000))
end
```

```
Assignment to `y` in soft scope is ambiguous because a global variable by th
e same name exists: `y` will be treated as a new local. Disambiguate by usin
g `local y` to suppress this warning or `global y` to assign to the existing
global variable.
```

we find that:

- can't see any significant outliers that would skew the distributions
- Because Patel length and Petal width have saperable distributions for different iris species, these 2 variables would have higher influence over classifying the species.
- We can also observe from the scatterplots of petal*length x petal*width, petal*width x sepal*width and petal*width x sepal*length, that the species clusters are clearly distinguishable and can easily be saperated a line.
- Hence, a sinple linear model(s) would be best suited for this type of classification here, without requiring of any feature transformations as the features are already distinguishable.
- The setosa species is the most easily distinguishable because of its small feature size.

```
plot_heatmap (generic function with 1 method)
```




```
@pipe cor(Matrix(train[!, Not(y_col)])) |> plot_heatmap(_, xlabel=names(train)[1:end-
1])
```

- if we look further we can see that `petal_width` is highly +vely correlated to `petal_length` and `sepal_length`. Similarly, `petal_length` and `sepal_length` are also sognificantly correlated.
- Also, `sepal_width` is slightly -vely correlated with other variables.


```
@pipe cov(Matrix(train[!, Not(y_col)])) |> plot_heatmap(_, xlabel=names(train)[1:end-
1])
```

A strong correlation is present between petal width and petal length.

Modeling

we earlier found out that the dataset is not too complex and simple models(most probably linear models) would be better suited for this problem.

```
((105, 4), (105), (45, 4), (45))
```

```
begin
    X, y = iris[!, Not(y_col)], iris[!, y_col]
    (X_train, y_train), (X_test, y_test) = stratifiedobs((X, y))
    size(X_train), size(y_train), size(X_test), size(y_test)
end
```

```
LogisticClassifier = MLJLinearModels.LogisticClassifier
```

```
LogisticClassifier = @load LogisticClassifier pkg=MLJLinearModels add=true
```

```
F import MLJLinearModels ✓ erbosity=0`.
```

```
pipe = Pipeline431(
         logistic_classifier = LogisticClassifier(
               lambda = 1.0,
               gamma = 0.0,
               penalty = :l2,
               fit_intercept = true,
               penalize_intercept = false,
               scale_penalty_with_samples = true,
               solver = nothing))
```

```
pipe = @pipeline LogisticClassifier
```

```
model =
Machine trained 0 times; caches data
  model: Pipeline431(logistic_classifier = LogisticClassifier(lambda = 1.0, …))
  args:
    1:  Source @863 ⏎ `ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Contin
    2:  Source @847 ⏎ `AbstractVector{ScientificTypesBase.Multiclass{3}}`
```

```
model = machine(pipe, X_train, y_train)
```

```
PerformanceEvaluation object with these fields:
  measure, operation, measurement, per_fold,
  per_observation, fitted_params_per_fold,
  report_per_fold, train_test_rows
Extract:
```

measure	operation	measurement	1.96*SE	per_fold	…
Accuracy()	predict_mode	0.827	0.0577	[0.833, 0.83	…
LogLoss(tol = 2.220446049250313e-16)	predict	0.693	0.0256	[0.682, 0.71	…
					…

1 column omitted

```
MLJ.evaluate!(model, resampling=CV(nfolds=6, rng=StableRNG(32)),
            measures=[MLJ.accuracy, log_loss])
```

```
Evaluating over 6 folds:  33%[========>                ]  ETA: 0:00:0
4 [KEvaluating over 6 folds:  50%[============>            ]  ETA: 0:00:0
3 [KEvaluating over 6 folds: 100%[=========================] Time: 0:00:0
7 [K
```

Model Tuning

```
ranges =
  [NumericRange(0.0 ≤ logistic_classifier.lambda ≤ 1.0; origin=0.5, unit=0.5) on log scale,
```

```
ranges = [range(pipe, :(logistic_classifier.lambda), lower=0.0, upper=1.0,
scale=:log),
        range(pipe, :(logistic_classifier.gamma), lower=0.0, upper=1.0, scale=:log),
        range(pipe, :(logistic_classifier.penalty), values=[:l2, :l1, :en, :none]),
]
```

```
tm =
ProbabilisticTunedModel(
  model = Pipeline431(
        logistic_classifier = LogisticClassifier(lambda = 1.0, …)),
  tuning = RandomSearch(
        bounded = Distributions.Uniform,
        positive_unbounded = Distributions.Gamma,
        other = Distributions.Normal,
        rng = Random._GLOBAL_RNG()),
  resampling = Holdout(
        fraction_train = 0.7,
        shuffle = false,
        rng = Random._GLOBAL_RNG()),
  measure = Accuracy(),
  weights = nothing,
  class_weights = nothing,
  operation = nothing,
  range = MLJBase.ParamRange[NumericRange(0.0 ≤ logistic_classifier.lambda ≤ 1.0; origin=0
  selection_heuristic = MLJTuning.NaiveSelection(nothing),
  train_best = true,
  repeats = 1,
  n = nothing,
  acceleration = CPU1{Nothing}(nothing),
  acceleration_resampling = CPU1{Nothing}(nothing),
  check_measure = true,
  cache = true)
```

```
tuned_model =
Machine trained 0 times; does not cache data
  model: ProbabilisticTunedModel(model = Pipeline431(logistic_classifier = LogisticClassif
  args:
    1:  Source @665 ⏎ `ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Contin
    2:  Source @529 ⏎ `AbstractVector{ScientificTypesBase.Multiclass{3}}`
```

```julia
tuned_model = machine(tm, X_train, y_train)
```

```
PerformanceEvaluation object with these fields:
  measure, operation, measurement, per_fold,
  per_observation, fitted_params_per_fold,
  report_per_fold, train_test_rows
Extract:
```

measure	operation	measurement	1.96*SE	per_fold …
Accuracy() LogLoss(tol = 2.220446049250313e-16)	predict_mode predict	0.914 1.21	0.0422 1.29	[0.889, 0.88 … [0.38, 0.52, … …

1 column omitted

```julia
MLJ.evaluate!(tuned_model, resampling=CV(nfolds=6, rng=StableRNG(32)),
              measures=[MLJ.accuracy, log_loss])
```

```
Evaluating over 6 folds:  33%[========>                 ]  ET/
2 [KEvaluating over 6 folds:  50%[============>             ]
8 [KEvaluating over 6 folds:  67%[================>         ]
0 [KEvaluating over 6 folds:  83%[====================>     ]
5 [KEvaluating over 6 folds: 100%[=========================]
0 [K
```

```
Pipeline431(
  logistic_classifier = LogisticClassifier(
        lambda = 0.37919381454429746,
        gamma = 0.6722497885235214,
        penalty = :none,
        fit_intercept = true,
        penalize_intercept = false,
        scale_penalty_with_samples = true,
        solver = nothing))
```

```
fitted_params(tuned_model).best_model
```

r =

```
(best_model = Pipeline431(                               , best_history_entry = (model =
                logistic_classifier = LogisticClassifier(
                      lambda = 0.379194,
                      gamma = 0.67225,
                      penalty = :none,
                      fit_intercept = true,
                      penalize_intercept = false,
                      scale_penalty_with_samples = true,
                      solver = nothing))
```

```
r = report(tuned_model)
```

0.9230769230769231

```
r.best_history_entry.measurement[1]
```

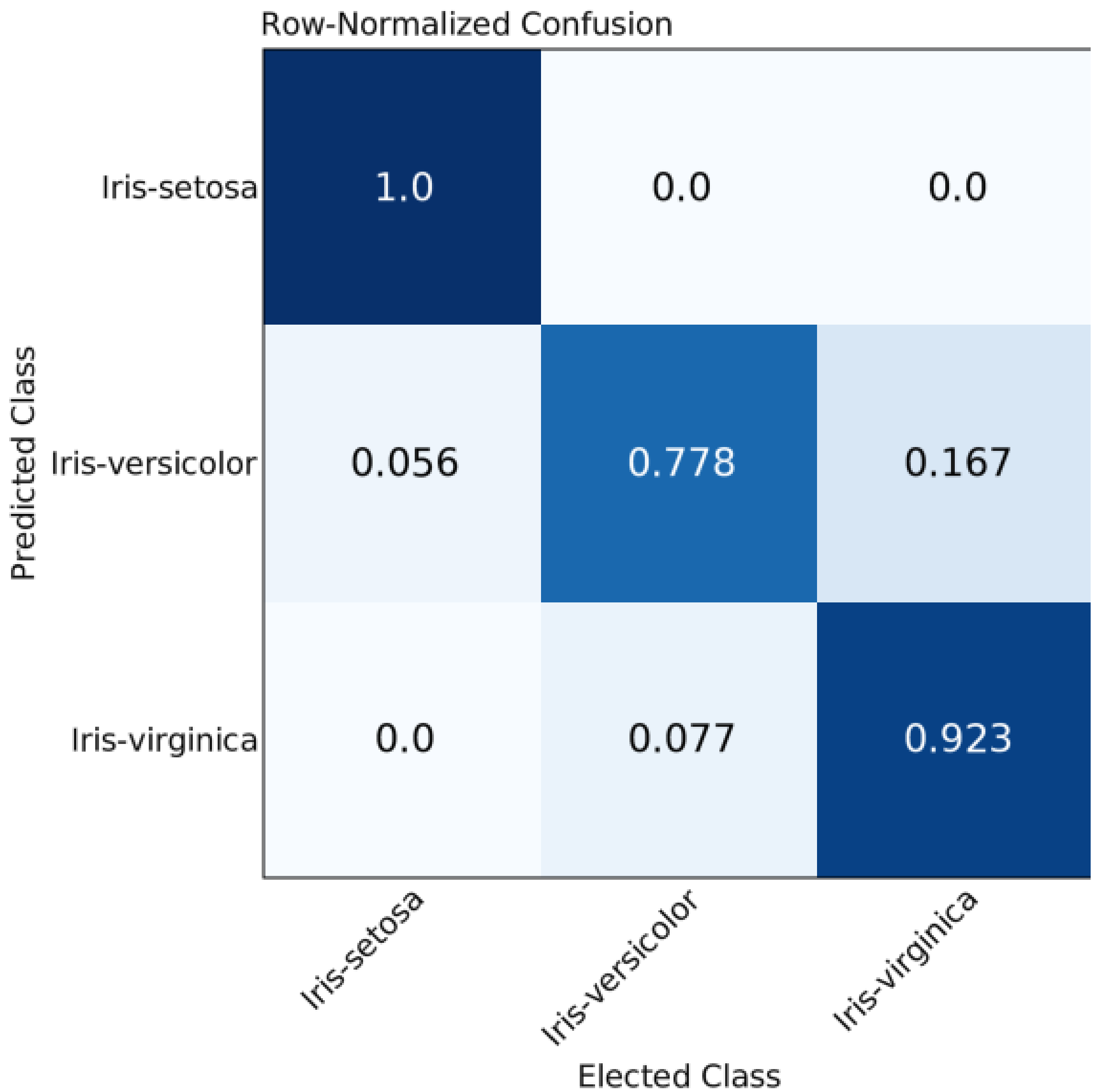
ŷ =

```
CategoricalArrays.CategoricalVector{InlineStrings.String15, UInt32, InlineStrings.String1
```

```
ŷ = MLJ.predict(tuned_model, X_test) .|> mode
```

0.8888888888888888

```
Accuracy()(ŷ, y_test)
```




```
begin
    cnf_mat = ConfusionMatrix()(ŷ, y_test)
    Lighthouse.plot_confusion_matrix(cnf_mat.mat, cnf_mat.labels, :Row)
end
```

```
The classes are un-ordered,
using order: InlineStrings.String15["Iris-setosa", "Iris-versicolor", "Iris-
virginica"].
To suppress this warning, consider coercing to OrderedFactor.
```

todo: classification report

```
WGLMakie.activate!()
```

```
begin
    using Pkg
    Pkg.activate(pwd())
    # Pkg.add(["CSV", "DataFrames"])
    # Pkg.add("StatsBase")
    # Pkg.add("Pipe")
    # Pkg.add("MLDataUtils")
    # Pkg.add("MLJ")
    # Pkg.add("StatsPlots")
    # Pkg.add("KernelDensity")
    # Pkg.add("WGLMakie")
    # # Pkg.rm("WGLMakie")
    # Pkg.add("StableRNGs")
    # Pkg.add(name="Lighthouse", version="0.14")
    # Pkg.add("PlutoUI")
    # Pkg.update()

    using Plots, StatsPlots, PlutoUI
    using KernelDensity
    using WGLMakie
    using DataFrames, CSV
    using Statistics, StatsBase, StableRNGs
    using Pipe
    using MLDataUtils, MLJ, Lighthouse
end
```

```
  Activating project at `~/Documents/my-portfolio-projects/machine learning projects/iris-flowers`
```