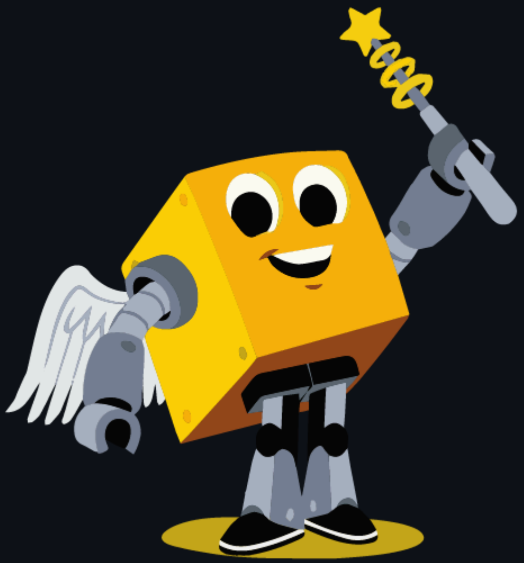
package management with `pixi`



Powered by Pixi

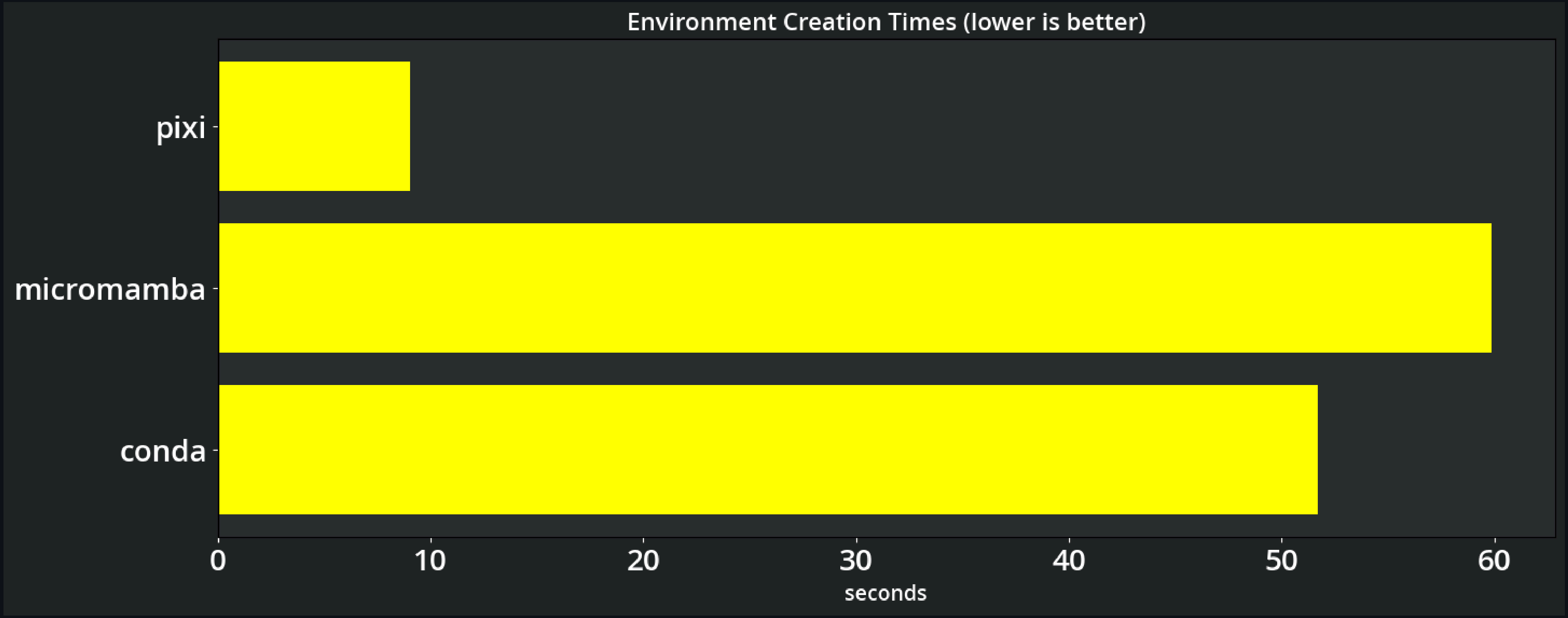
What is `pixi`?

> `pixi` is a fast software package manager built on top of the existing conda ecosystem. Spins up development environments quickly on Windows, macOS and Linux.

Another package manager 🧐?

Tool	Installs Python	Builds Packages	Task Runner	Built-in lockfiles	Fast	Python-Independent	Install Conda Pkgs
`conda`	✅	❌	❌	❌	❌	❌	✅
`mamba`	✅	❌	❌	❌	✅	✅	✅
`pip`	❌	✅	❌	❌	❌	❌	❌
`uv`	❌	❌	❌	❌	✅	✅	❌
`poetry`	❌	✅	❌	✅	❌	❌	❌
`pixi`	✅	✅	✅	✅	✅	✅	✅

`pixi` is fast 🚀


Environment Creation Times (lower is better)
pixi
micromamba
conda
0
10
20
30
40
50
60
seconds

Easy Installation

```
# Linux & macOS
$ curl -fsSL https://pixi.sh/install.sh | bash

# Windows
$ iwr -useb https://pixi.sh/install.ps1 | iex
```

Global Tool Installation

Install conda pkgs globally, with each of them having their own environment.

- no more `base` conda envs!

```
$ pixi global install hyperfine ripgrep fzf zoxide
✓ Installed package hyperfine 1.18.0 h5ef7bb8_0 from conda-forge
✓ Installed package ripgrep 14.1.0 h5ef7bb8_0 from conda-forge
✓ Installed package fzf 0.53.0 h75b854d_0 from conda-forge
✓ Installed package zoxide 0.9.4 h5ef7bb8_1 from conda-forge
  These executables are now globally available:
   -  hyperfine
   -  rg
   -  fzf
   -  zoxide
```

Project level features

Features are a way to group dependencies and tasks together.

- allow for logical grouping, and saves any headaches from dependency conflicts.

```toml
[features.docs]
dependencies = ["mkdocs", "mkdocs-material"]
tasks = { serve = "mkdocs serve" }

[features.r]
dependencies = ["r-base", "r-essentials", "bioconductor-pharmacogx"]
tasks = { pgx = "Rscript -e 'library(pharmacogx); pharmacogx::someFunction()'" }
```

Sidestep activation

No `conda activate` or `conda deactivate` needed!

- per-directory envs are automatically found and commands are run in the correct env.

```
$ pixi run --environment docs mkdocs --serve
```

if you truly wish to activate an env, you can do so with `pixi shell`.

```
$ pixi shell --environment docs

$ (docs) mkdocs --serve
```

Tasks

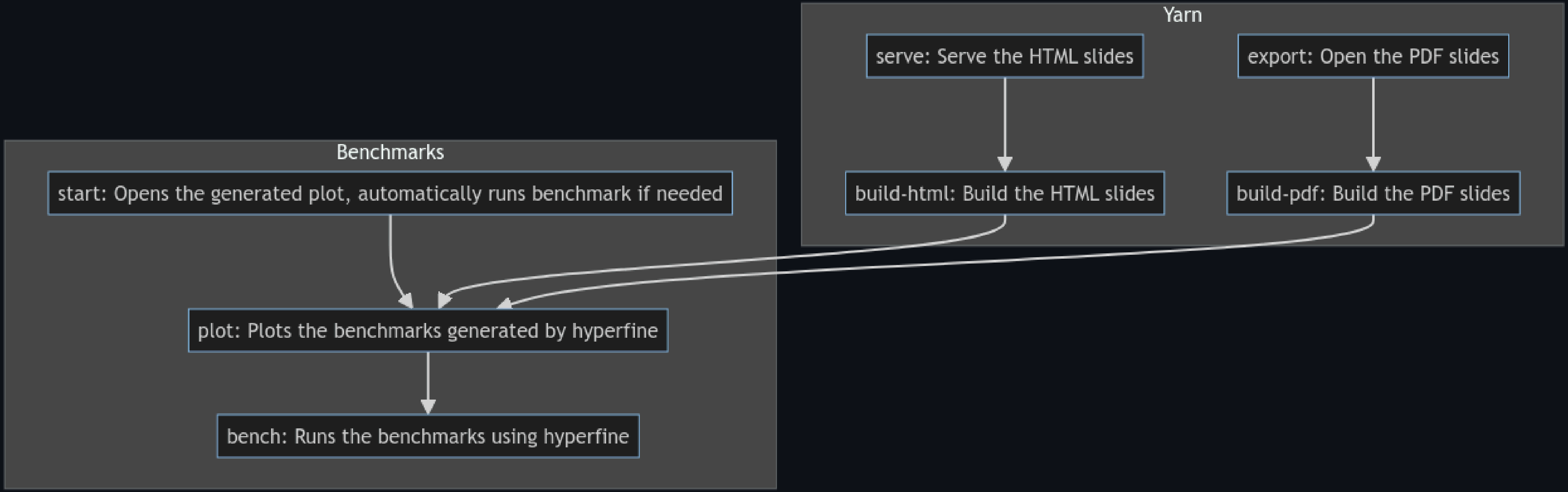
Run tasks in the correct environment, without needing to activate it.

```toml
# assume the following in your pyproject.toml
[tool.pixi.feature.docs.tasks]
serve = "mkdocs serve"
```

`pixi` will automatically activate the correct environment and run the task.

```
$ pixi run serve
```

These Slides Were Generated Using `pixi tasks`



Yarn
serve: Serve the HTML slides
export: Open the PDF slides
build-html: Build the HTML slides
build-pdf: Build the PDF slides
Benchmarks
start: Opens the generated plot, automatically runs benchmark if needed
plot: Plots the benchmarks generated by hyperfine
bench: Runs the benchmarks using hyperfine

Use both `conda` and `pip` packages together

`pixi` solves environments across multiple package managers.

```toml
[dependencies] # conda packages
python = ">3.9"

[pypi-dependencies] # pip packages
damply = "*"
```

Other package managers

```toml
[dependencies]
yarn = ">=4.3.0,<4.4"

[tasks]
build = "yarn build"
serve = "yarn serve"
```

Multi-Environments

Create multiple environments in a single project.

- this allows for different dependencies for different tasks.
- ensures that they are solved with the correct dependencies.

```toml
[environments]
dev = { features = ["docs", "tests"] }
test = { features = ["tests"] }
publish = { features = ["docs", "release"] }
```

Multi-Environments (cont.)

Allows for testing across multiple python-versions across multiple environments.

```toml
[environments]
testpy39 = { features = ["py39", "tests"] }
testpy310 = { features = ["py310", "tests"] }
testpy311 = { features = ["py311", "tests"] }
```

Multi-Platform

No more "it works on my machine"!

- `pixi` lockfiles are platform-independent
- solves every environment across multiple platforms
  - determine in real-time if a package is not available on a platform

```toml
[project]
platforms = ["win-64", "linux-64", "osx-64", "osx-arm64"]
```

Complex Dependency Resolution

all the configurations can be defined at the ***feature*** level.

```toml
[project]
channels = ["conda-forge"]
platforms = ["linux-64", "linux-aarch64","osx-64", "win-64"]

[feature.cuda]
channels = ["nvidia", "conda-forge"]
platforms = ["linux-64"] # only going to use this on linux

[feature.cuda.dependencies]
cudatoolkit = "*"
pytorch-cuda = { version = "12.1", channel = "pytorch" }

[feature.cuda.tasks]
train-model = "python train.py --cuda"
evaluate-model = "python evaluate.py --cuda"
# running `pixi run model` will run both tasks in the correct environment
model = { depends_on = ["train-model", "evaluate-model"]}
```

Lockfiles

`pixi` generates a lockfile `pixi.lock` that:

- is human-readable
- can be checked into version control like `git`
  - no more `environment.yml` files!
- can be used to recreate the exact environment on another machine
- solves multiple environments *across* **multiple machine types**

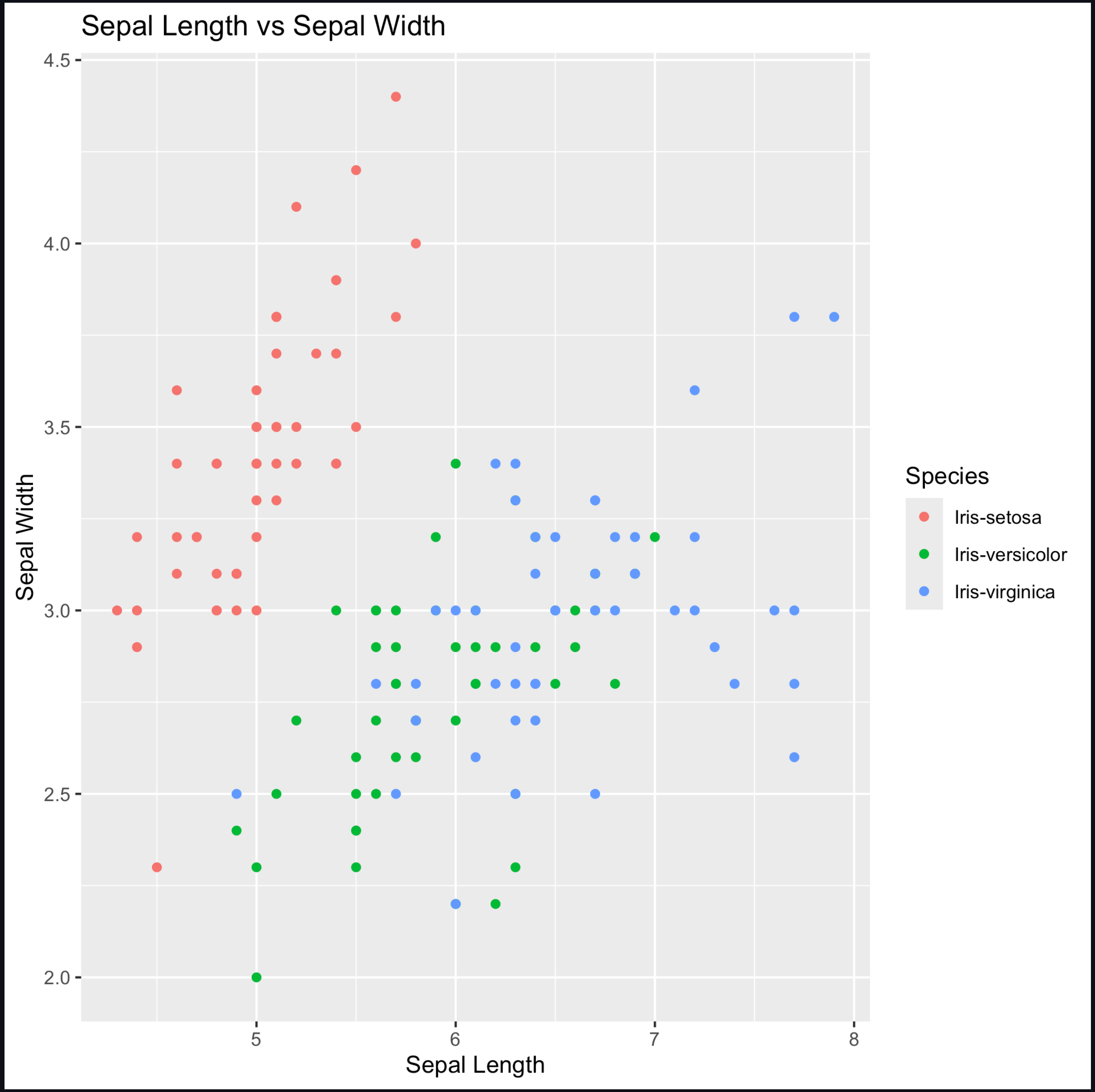
TL;DR

reproducible **EVERYTHING** with `pixi`

Thank you!

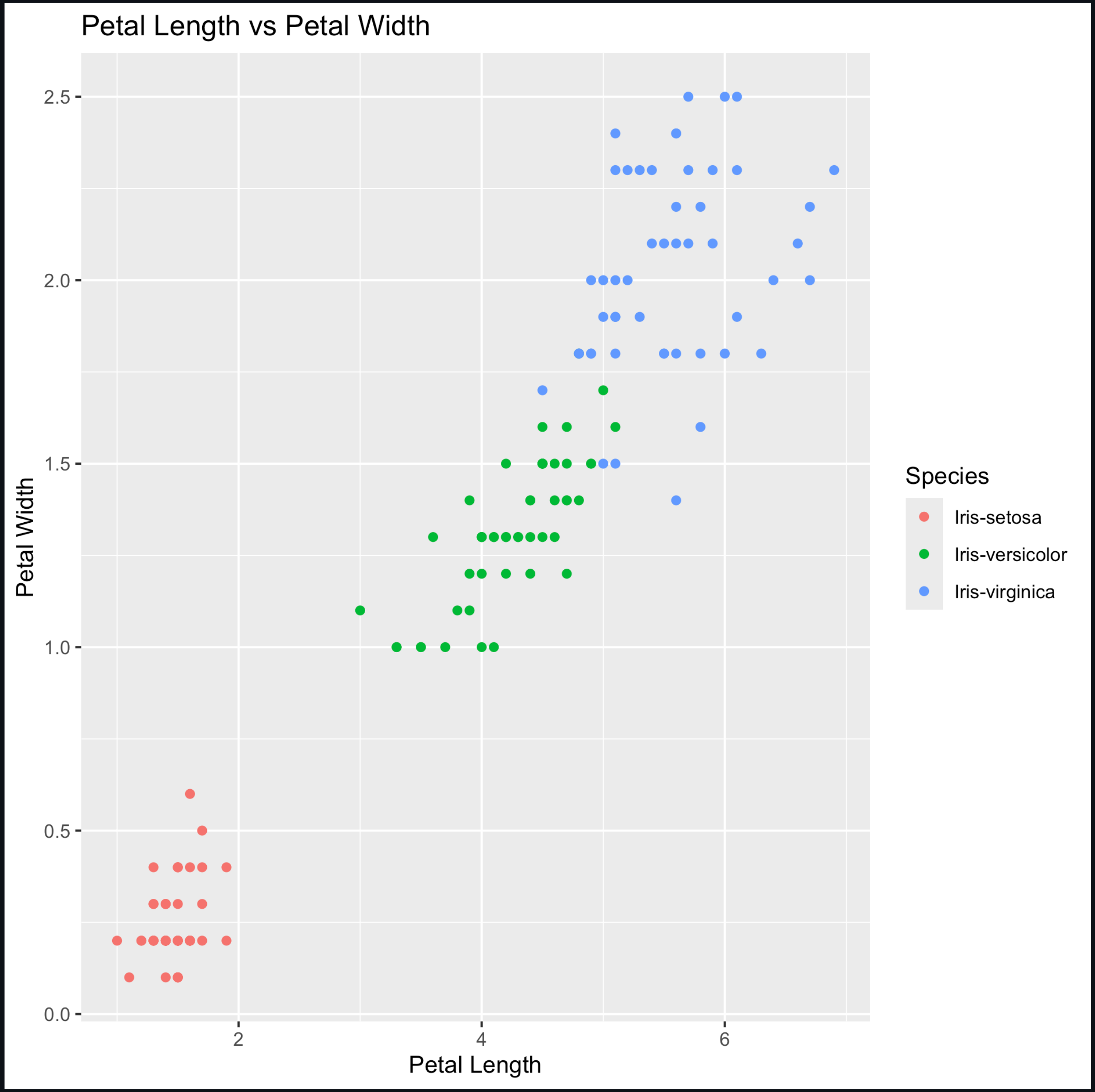
random

graph



random

graph


Petal Length vs Petal Width
Petal Width
Petal Length
Species
Iris-setosa
Iris-versicolor
Iris-virginica