Neural Networks Lab

Last modified: 2026-11-19

```
# The Python chunks on this page import matplotlib, numpy, pandas
# and scikit-learn; reticulate provisions them.
py_packages <- c("matplotlib", "numpy", "pandas", "scikit-learn")
library(reticulate) # nolint: undesirable_function_linter
library(tidymodels) # nolint: undesirable_function_linter
py_require(py_packages)
```

1 The book's Chapter 10 lab

This page holds the code lab for the Neural Networks chapter.

To work the lab without the solutions, download a stub with only the exercises:

- Quarto
- Jupyter notebook[1]
- Python[2]
- R[3]

Each solution shows the code in Python and in R.

Section 10.9.1 of the book [1] is the source of these exercises. The book uses PyTorch and the R package `keras` for this step. This lab uses scikit-learn [2] in Python and the `nnet` engine of tidymodels in R, which fit a single hidden layer with no extra framework.

2 Exploring the Hitters data

Statistics and salaries of major league baseball players on 322 rows make up the `Hitters` data [1, Sec. 10.9.1]. The data are in `data/hitters.csv`[4], one row per player. The columns this course uses are:

- `Salary`: the outcome[5], the player's salary in thousands of dollars, missing for some players.
- `CHits`: the number of hits during the player's career.
- `Years`: the number of years the player has played in the major leagues.
- The other numeric columns: batting and fielding counts for the previous season and for the career.

Dataset 1: The baseball (`Hitters`) data

Exercise 2.1 (Lab: players with a missing salary)**.** Use the `Hitters` data. How many players have a missing `Salary`?

Hint

Count the missing values of a column with a pandas method in Python, or with `is.na()` in R.

[1] https://morrison-lab.github.io/lds/chapters/lab-stubs/neural-nets-lab-stub.ipynb
[2] https://morrison-lab.github.io/lds/chapters/lab-stubs/neural-nets-lab-stub.py
[3] https://morrison-lab.github.io/lds/chapters/lab-stubs/neural-nets-lab-stub.R

Solution

Solution 2.1.

2.0.1 Python

```
import pandas as pd

hit_eda_py = pd.read_csv("data/hitters.csv")
hit_eda_py_missing = int(hit_eda_py["Salary"].isna().sum())
hit_eda_py_missing
```

```
59
```

2.0.2 R

```
hit_eda_r <- readr::read_csv("data/hitters.csv", show_col_types = FALSE)
hit_eda_r_is_missing <- is.na(hit_eda_r$Salary)
hit_eda_r_missing <- sum(hit_eda_r_is_missing)
hit_eda_r_missing
```

```
[1] 59
```

59 of the 322 players, or 18.3%, have no recorded `Salary`. The next exercises drop them, because the outcome[a] has to be known to fit a model.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-outcome-vector

Exercise 2.2 (Lab: salary and career hits)**.** Use the `Hitters` data. Among the players with a recorded `Salary`, what is the sample correlation[a] between `Salary` and `CHits`?

Hint

Drop the rows with a missing value before you compute it, because a missing value makes the result missing in R and is skipped in pandas.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-pearson-r

Solution

Solution 2.2. This solution continues Solution 2.0.2.

2.0.3 Python

```
hit_py_known = hit_eda_py.dropna(subset=["Salary"])
hit_eda_py_cor = float(hit_py_known["Salary"].corr(hit_py_known["CHits"]))
round(hit_eda_py_cor, 3)
```

```
0.549
```

2.0.4 R

```
hit_r_known <- hit_eda_r |>
  tidyr::drop_na(Salary)
hit_eda_r_cor <- cor(hit_r_known$Salary, hit_r_known$CHits)
round(hit_eda_r_cor, 3)
```

```
[1] 0.549
```

The correlation[a] is 0.55 among the 263 players with a recorded `Salary`, so players with more career hits tend to earn more.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-pearson-r

Exercise 2.3 (Lab: a histogram of salary)**.** Use the `Hitters` data. Draw a histogram of `Salary` for the players with a recorded `Salary`.

Hint

Draw a histogram of one column, as in the histogram of the year of each car in the primer.

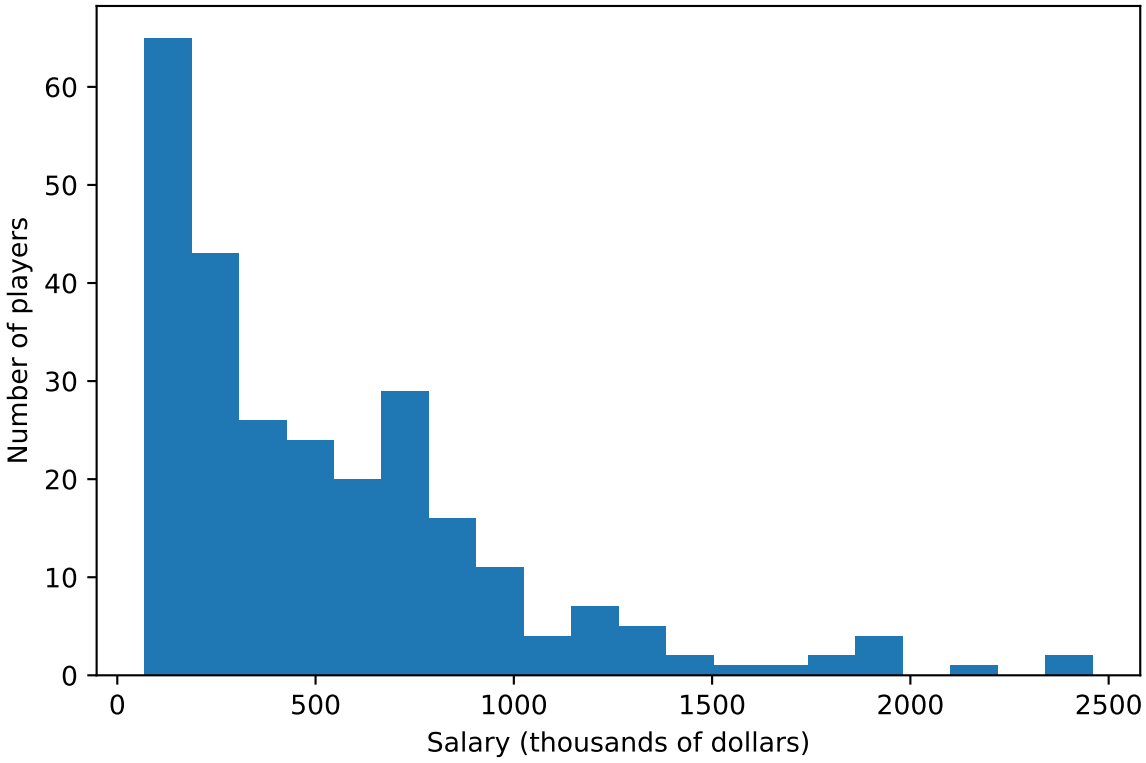
Solution

Solution 2.3. This solution continues Solution 2.0.4.

2.0.5 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.hist(hit_py_known["Salary"], bins=20);
ax.set_xlabel("Salary (thousands of dollars)");
ax.set_ylabel("Number of players");
plt.show()
```



2.0.6 R

```
library(ggplot2) # nolint: undesirable_function_linter

salary_histogram <- hit_r_known |>
  ggplot() +
  aes(Salary) +
  geom_histogram(bins = 20) +
  labs(x = "Salary (thousands of dollars)", y = "Number of players")
salary_histogram
```

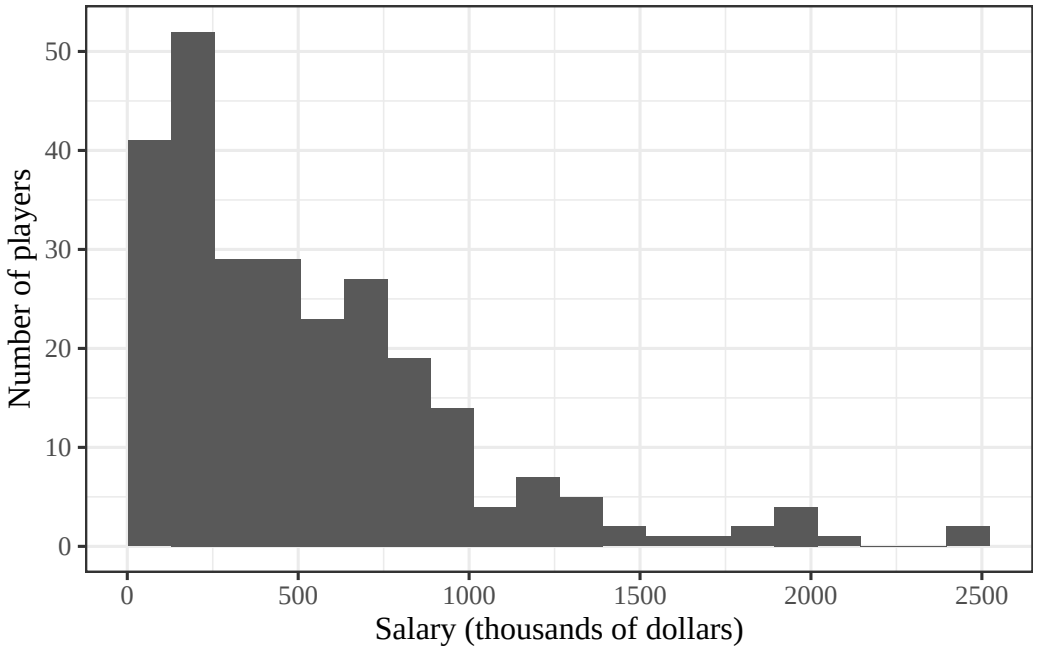


Figure 1: Bars counting the players at each salary, in thousands of dollars.

```
hit_eda_r_median <- median(hit_r_known$Salary)
hit_eda_r_mean <- mean(hit_r_known$Salary)
hit_eda_r_per_bin <- nrow(hit_r_known) / 20
```

I chose 20 bins, which is about 13 players per bar on average. Far fewer bins would hide the long right tail inside a few wide bars, and far more would leave most bars with only a few players, so the outline would look ragged. That is why 20 is a compromise between the two.
Figure 1 shows a strongly right-skewed shape: the sample median[a] is 425 thousand dollars, far below the sample mean[b] of 536 thousand dollars.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-median
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

Exercise 2.4 (Lab: salary against career hits)**.** Use the `Hitters` data. Draw a scatterplot of `Salary` against `CHits` for the players with a recorded `Salary`.

Hint

Draw a scatterplot of one column against another, as in the plot of acceleration against horsepower in the primer.

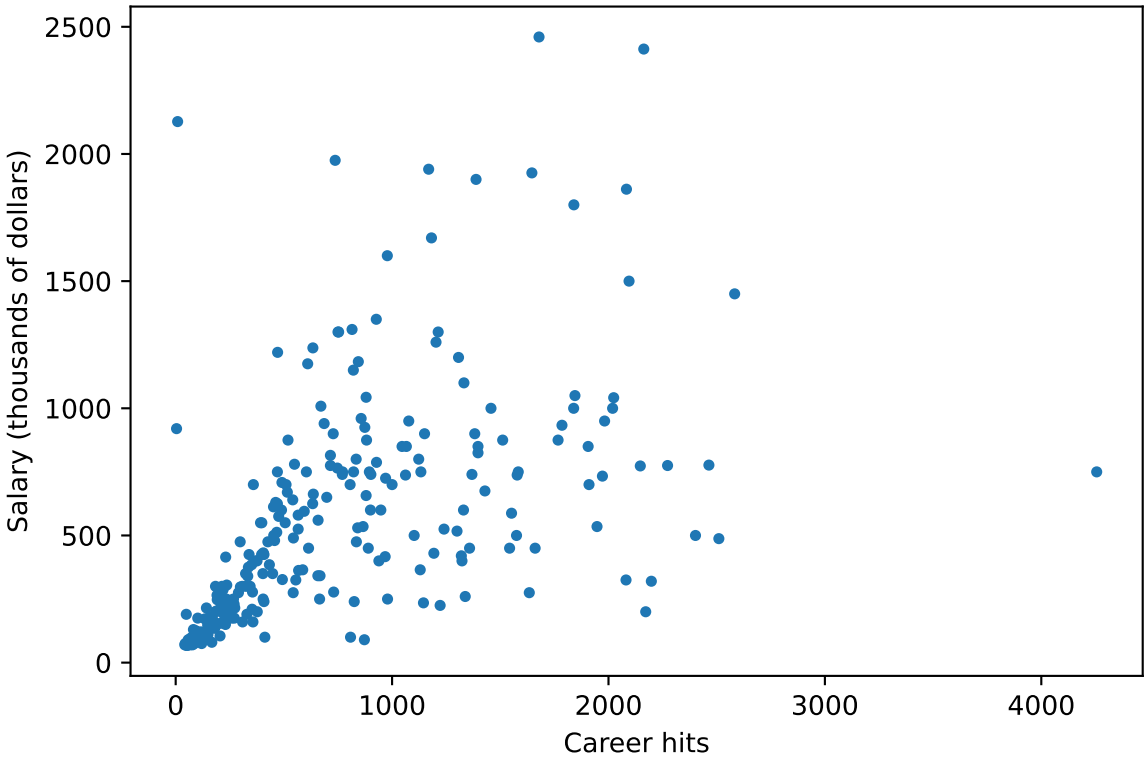
Solution

Solution 2.4. This solution continues Solution 2.0.4.

2.0.7 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.scatter(hit_py_known["CHits"], hit_py_known["Salary"], s=10);
ax.set_xlabel("Career hits");
ax.set_ylabel("Salary (thousands of dollars)");
plt.show()
```



2.0.8 R

```
library(ggplot2) # nolint: undesirable_function_linter

salary_chits_plot <- hit_r_known |>
  ggplot() +
  aes(CHits, Salary) +
  geom_point() +
  labs(x = "Career hits", y = "Salary (thousands of dollars)")
salary_chits_plot
```

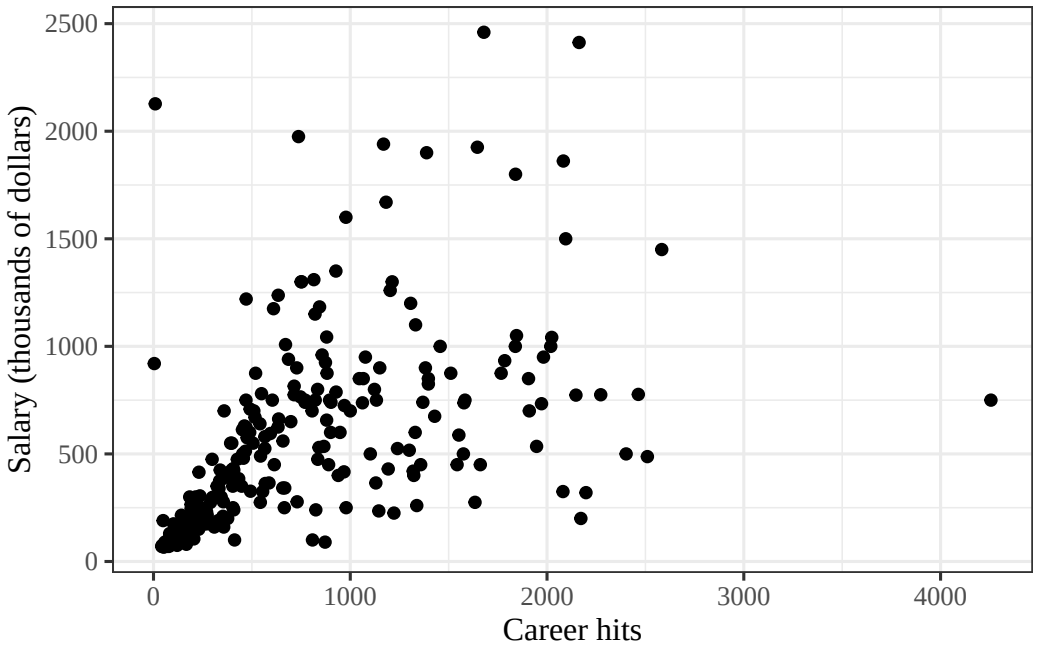


Figure 2: Salary, in thousands of dollars, against career hits, with one dot per player.

In Figure 2, salary rises with career hits but with a widening spread, so a straight line through the points would describe the players with few career hits better than the players with many.

3 Preparing the data

Exercise 3.1 (Lab: load the Hitters data)**.** Read the Hitters data [1], which record the `Salary` in thousands of dollars of major league baseball players, and keep the 16 numeric predictors of `Salary`. Drop the players whose `Salary` is missing.

Hint

The data frame functions read the file and select columns, and `.dropna` removes the rows with a missing value. In R, `drop_na()` does the same.

Solution

Solution 3.1.

3.0.1 Python

```
import numpy as np
import pandas as pd

hitters = pd.read_csv("data/hitters.csv").dropna(subset=["Salary"])
numeric_x = hitters.drop(columns=["Salary", "League", "Division", "NewLeague"])
numeric_x.shape
```

```
(263, 16)
```

3.0.2 R

```
hitters <- readr::read_csv("data/hitters.csv", show_col_types = FALSE)
hitters_num <- hitters |>
  drop_na(Salary) |>
  select(-League, -Division, -NewLeague)
dim(hitters_num)
```

```
[1] 263  17
```

The Hitters data have 322 players, 59 of them with a missing `Salary`. The 263 players left have 16 numeric predictors.

Exercise 3.2 (Lab: split the Hitters data)**.** Split the players from Exercise 3.1 at random into a training set with two thirds of them and a test set with the rest.

Hint

The random number functions draw row positions without repeats, and a selection by position keeps those rows.

Solution

Solution 3.2.

3.0.3 Python

```
from sklearn.model_selection import train_test_split

split_py = train_test_split(
    numeric_x, hitters["Salary"], test_size=1 / 3, random_state=0
)
x_train, x_test, y_train, y_test = split_py
(len(y_train), len(y_test))
```

```
(175, 88)
```

3.0.4 R

```
set.seed(571)
split_r <- initial_split(hitters_num, prop = 2 / 3)
train_r <- training(split_r)
test_r <- testing(split_r)
n_train <- nrow(train_r)
n_test <- nrow(test_r)
split_sizes <- c(n_train, n_test)
split_sizes
```

```
[1] 175  88
```

The training set holds 175 players and the test set 88 in the R run, and 175 and 88 in the Python run. The two runs use different random splits.

Exercise 3.3 (Lab: standardize the Hitters predictors)**.** Standardize each predictor in the training set and the test set from Exercise 3.2 with its mean[a] and standard deviation[b] in the training set.

Hint

Broadcasting subtracts a mean[c] and divides by a standard deviation[d] for every column at once, and both numbers come from the training set alone.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-sd
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-sd

Solution

Solution 3.3.

3.0.5 Python

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(x_train)
x_train_z = scaler.transform(x_train)
x_test_z = scaler.transform(x_test)
x_train_z.mean(axis=0).round(3)
```

```
array([-0., -0.,  0.,  0.,  0.,  0.,  0.,  0., -0., -0., -0., -0.,  0.,
        0., -0., -0.])
```

3.0.6 R

```
train_x <- select(train_r, -Salary)
test_x <- select(test_r, -Salary)
train_center <- colMeans(train_x)
train_sd <- map_dbl(train_x, sd)
train_z <- scale(train_x, center = train_center, scale = train_sd) |>
  as_tibble() |>
  mutate(Salary = train_r$Salary)
test_z <- scale(test_x, center = train_center, scale = train_sd) |>
  as_tibble() |>
  mutate(Salary = test_r$Salary)
train_means_z <- select(train_z, -Salary) |> colMeans()
round(train_means_z, 3)
```

```
  AtBat    Hits   HmRun    Runs     RBI   Walks   Years  CAtBat   CHits  CHmRun
      0       0       0       0       0       0       0       0       0       0
  CRuns    CRBI  CWalks PutOuts Assists  Errors
      0       0       0       0       0       0
```

Each standardized predictor has mean[a] zero in the training set. The test set is shifted and scaled by the training set's mean and standard deviation[b], not its own, so the test set looks the same to the fitted model as new data would.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-sd

4 Fitting the models

Exercise 4.1 (Lab: fit a linear model to the Hitters training set)**.** Fit a linear regression of `Salary` on the standardized predictors in the training set from Exercise 3.3, and report its mean[a] absolute error on the test set.

Hint

Every estimator[b] in scikit-learn is fit with `.fit`, and the error to report is the mean[c] of the absolute differences between the predictions and the actual salaries. In R, the `linear_reg()` specification of tidymodels is fit with `fit()`.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimator
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

Solution

Solution 4.1.

4.0.1 Python

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

lm_py = LinearRegression().fit(x_train_z, y_train)
lm_mae_py = mean_absolute_error(y_test, lm_py.predict(x_test_z))
lm_mae_py
```

```
272.9659606804639
```

4.0.2 R

```
lm_r <- linear_reg() |> fit(Salary ~ ., data = train_z)
lm_pred_r <- predict(lm_r, new_data = test_z) |>
  bind_cols(test_z)
lm_mae_r <- mae(lm_pred_r, truth = Salary, estimate = .pred)$.estimate
lm_mae_r
```

```
[1] 271.5249
```

The test mean[a] absolute error is 271.5 in the R run and 273 in the Python run, in thousands of dollars. The two runs use different random splits, so their errors are not directly comparable.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

Exercise 4.2 (Lab: fit a single-hidden-layer network to the Hitters training set)**.** Fit a neural network with one hidden layer of 10 units and a weight decay penalty of 0.1 to the training set from Exercise 3.3, and report its mean[a] absolute error on the test set.

Hint

The neural network estimator of scikit-learn has settings for the sizes of its hidden layers and for its weight decay penalty.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

Solution

Solution 4.2.

4.0.3 Python

```
from sklearn.neural_network import MLPRegressor

nn_py = MLPRegressor(
    hidden_layer_sizes=(10,), alpha=0.1, max_iter=5000, random_state=0
)
nn_py.fit(x_train_z, y_train)
```

```
/work/.uvcache/archive-v0/A5OOOHFDF-zpr7fU/lib/python3.13/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:785: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
MLPRegressor(alpha=0.1, hidden_layer_sizes=(10,), max_iter=5000, random_state=0)
```

```
nn_mae_py = mean_absolute_error(y_test, nn_py.predict(x_test_z))
nn_mae_py
```

```
324.96139793430615
```

4.0.4 R

```
nn_spec <- mlp(hidden_units = 10, penalty = 0.1, epochs = 500) |>
  set_engine("nnet") |>
  set_mode("regression")
set.seed(571)
nn_r <- fit(nn_spec, Salary ~ ., data = train_z)
nn_pred_r <- predict(nn_r, new_data = test_z) |>
  bind_cols(test_z)
nn_mae_r <- mae(nn_pred_r, truth = Salary, estimate = .pred)$.estimate
nn_mae_r
```

```
[1] 356.5408
```

The test mean[a] absolute error is 356.5 in the R run and 325 in the Python run, in thousands of dollars. The two errors differ partly because the two runs use different random splits, and partly because the two libraries differ: they scale the penalty differently, train for a different number of steps, and use different activation functions in the hidden layer.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

References

[1] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An introduction to statistical learning: With applications in Python*. Springer, 2023. doi: 10.1007/978-3-031-38747-0[6].

[2] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, Available: https://jmlr.org/papers/v12/pedregosa11a.html

[6] https://doi.org/10.1007/978-3-031-38747-0