Franklin

The MBG jupyter exercise tool

Table of contents

1

The student runs a jupyter notebook on their own computer in a way that lets them focus on content and learning without the frustration by library incompatibilities and platform specific dependencies.

On both Mac and Windows, running the following command, prompts the student to select course and exercise and then downloads a folder with a jupyter notebook.

Listing 1.1 `Terminal`

```
franklin download
```

The student runs the following command, which again prompts for course and exercise and launches JupyterLab from an environment on the student's computer with all required dependencies preinstalled.

Listing 1.2 `Terminal`

```
franklin jupyter
```

The environment and notebook is exactly the same, wether on Mac or Windows, allowing them to work together, compare results, and receive uniform and unambiguous feedback from instructors.

Part I

Terminal

2 The terminal

This pages are under construction

A terminal is a text-based interface that allows users to interact with the operating system by typing commands. It provides direct access to the system shell—a program that interprets user input and executes corresponding instructions.

Historically, "terminals" referred to physical devices (e.g., teletypewriters or CRT monitors) connected to mainframe computers. In modern computing, a terminal is typically a software application (e.g., Terminal on macOS, GNOME Terminal on Linux, Command Prompt or PowerShell on Windows) that emulates this interface.

Key characteristics of a terminal include:

Command-Line Interface (CLI): Unlike graphical user interfaces (GUIs), a terminal requires textual input and returns textual output. Shell Access: The terminal runs a shell such as bash, zsh, or fish, which interprets and executes user commands (e.g., file manipulation, process control, networking). Script Execution: Users can run scripts, automate tasks, and chain commands using control operators (e.g., &&, |, >, etc.). Remote Access: Terminals can be used to log into remote systems via protocols such as SSH, providing a low-overhead method for remote system administration and programming. The terminal is a powerful tool, particularly in software development, system administration, and high-performance computing. It enables reproducibility, fine-grained control, and automation, though it requires users to become familiar with a command language and often lacks the immediate feedback and discoverability of graphical interfaces.

Ctrl-C for abort

3 Jupyterlab

Introduction to JupyterLab

JupyterLab is the next-generation web-based user interface for Project Jupyter, designed to offer a flexible and extensible environment for interactive computing. It builds upon the classic Jupyter Notebook interface by integrating a wide range of tools—code consoles, terminals, text editors, data file viewers, and notebooks—into a unified workspace.

The JupyterLab interface is organized into a multi-tab layout, resembling an integrated development environment (IDE). Users can open multiple documents side by side (e.g., a Python script next to a Markdown file and a terminal), drag-and-drop tabs, and link interactive views of the same dataset. It supports live code execution, output rendering (including plots and LaTeX equations), and seamless integration with kernels for many languages, though Python is most common.

Key features of JupyterLab include:

Notebook authoring: Full support for .ipynb notebooks with executable code cells, Markdown, and rich outputs. Code consoles: Interactive REPLs connected to notebooks or scripts for testing code snippets. Text and code editing: Built-in editor with syntax highlighting, version control integration, and support for many file types. Terminal access: Direct command-line interface within the browser for shell-level operations. Extensibility: Modular architecture that supports plugins for additional language kernels, visualizations, or custom workflows. JupyterLab preserves all the functionality of the classic notebook interface while offering a more powerful and customizable platform suitable for complex data science workflows, scientific research, and reproducible computing. It is widely used in academia, industry, and education due to its interactive nature and support for literate programming principles.

JupyterLab is a highly extensible, feature-rich notebook authoring application and editing environment, and is a part of Project Jupyter, a large umbrella project centered around the goal of providing tools (and standards) for interactive computing with computational notebooks.

A computational notebook is a shareable document that combines computer code, plain language descriptions, data, rich visualizations like 3D models, charts, graphs and figures, and interactive controls. A notebook, along with an editor like JupyterLab, provides a fast interactive environment for prototyping and explaining code, exploring and visualizing data, and sharing ideas with others.

JupyterLab is a sibling to other notebook authoring applications under the Project Jupyter umbrella, like Jupyter Notebook and Jupyter Desktop. JupyterLab offers a more advanced, feature rich, customizable experience compared to Jupyter Notebook.MM

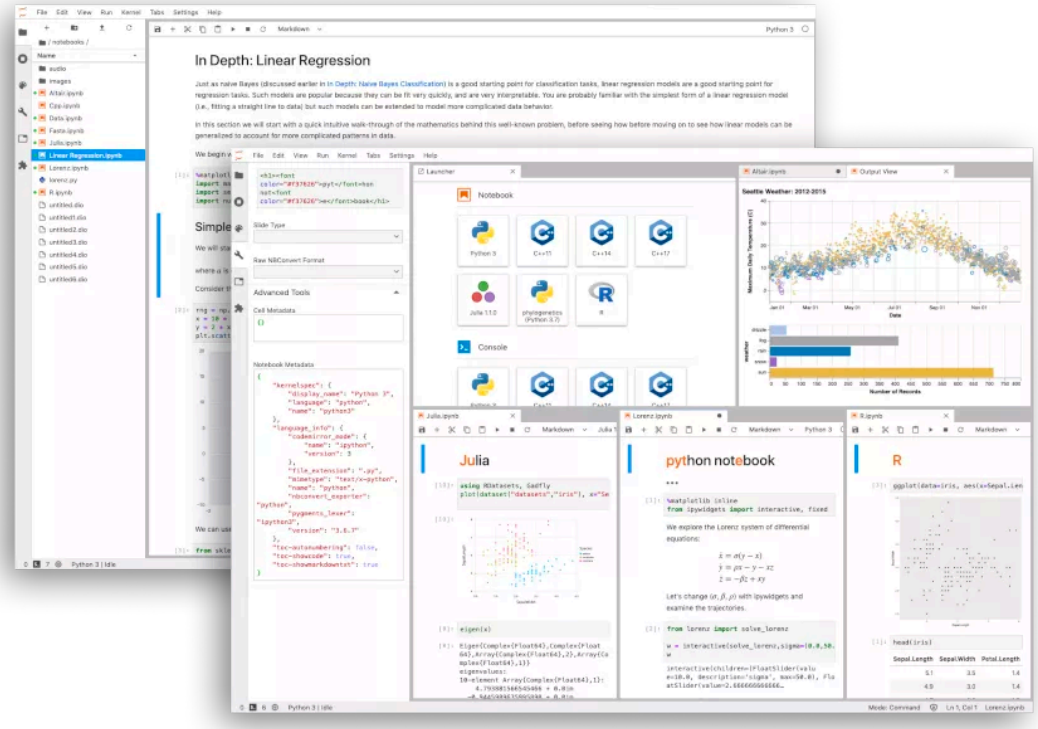


Figure 3.1: image.png

4 Starting jupyter lab

Terminal commands:

```
conda activate ctib
jupyter lab
```

It then appears in your default browser. |

5 Side panes

- alks flaksjf laskjdf laskdfj

asdflaksdfj

Jupyter Notebooks are an interactive computing environment that allow users to create and share documents containing live code, equations, visualizations, and narrative text. Originally developed as part of the IPython project, Jupyter (short for Julia, Python, R) now supports over 100 programming languages and has become a standard tool in data science, scientific computing, and education.

At the core of a Jupyter Notebook is a web-based interface that organizes content into "cells." These cells can contain code (typically in Python, but also in other languages via kernels), formatted text using Markdown, LaTeX for equations, and embedded multimedia elements. Users execute code in-place, and outputs such as plots or tables appear directly below the corresponding cells. This structure enables exploratory data analysis and facilitates reproducibility by interleaving code and its results with documentation.

Notebooks are stored in .ipynb files (JSON format), which preserve the code, outputs, and formatting. They can be run locally using the Jupyter server or hosted in cloud

environments such as Google Colab or Binder. For scientific workflows, notebooks can integrate with tools for version control, containerization, and workflow management, making them a flexible instrument for open and reproducible research.

Despite their strengths, Jupyter Notebooks are not without limitations. Version control can be challenging due to the JSON-based format, and improper use (e.g., out-of-order execution) can compromise reproducibility. Nevertheless, their advantages in accessibility, interactivity, and communication have made them central to modern computational work.

5.1Keyboard shortcuts

5.1.1 Command Mode in Jupyter Notebooks

In Jupyter Notebooks, **Command Mode** is one of the two main interaction modes—the other being **Edit Mode**. Command Mode is active when the cell border is **blue**, indicating that keyboard commands will be interpreted as notebook-level operations rather than editing the cell's content.

Command Mode allows users to manage cells and perform structural modifications without using the mouse. This enhances efficiency, especially when working with large notebooks. Pressing `Esc` while in a cell activates Command Mode.

Common Keyboard Shortcuts in Command Mode 5-1

Shortcut	Description
`Enter`	Switch to Edit Mode in the selected cell
`A`	Insert a new cell **above** the current cell
`B`	Insert a new cell **below** the current cell
`D, D`	Delete the selected cell (press `D` twice quickly)
`Z`	Undo the last cell deletion
`Y`	Change cell type to **code**
`M`	Change cell type to **Markdown**
`C`	Copy the selected cell
`X`	Cut the selected cell
`V`	Paste cell **below**
`Shift + V`	Paste cell **above**
`Shift + Up/Down`	Extend selection to multiple cells
`Ctrl + S` (or `Cmd + S` on macOS)	Save the notebook
`H`	Show all keyboard shortcuts
`0, 0`	Restart the kernel (press `0` twice quickly)

Shortcut	Description
`Shift + M` (in Command Mode)	**Merge** selected cells

These shortcuts make navigation and cell management significantly faster, enabling an efficient coding and documentation workflow within Jupyter Notebooks.

5.1.2 Edit Mode in Jupyter Notebooks

Edit Mode is activated when a cell's border turns **green**, allowing the user to directly modify the contents of the cell. You can enter Edit Mode by pressing `Enter` while a cell is selected in Command Mode. This mode is primarily used for writing and editing code or Markdown content within cells.

Common Keyboard Shortcuts in Edit Mode 5-2

Shortcut	Description
`Ctrl + Enter`	Run the current cell and remain in Edit Mode
`Shift + Enter`	Run the current cell and move to the next cell
`Alt + Enter`	Run the current cell and insert a new cell below
`Esc`	Switch to Command Mode
`Ctrl + /`	Toggle comment on selected lines (code cells only)
`Tab`	Code completion or indent
`Shift + Tab`	Show tooltip/help for the object under cursor
`Ctrl + ]`	Indent the current line or selection
`Ctrl + [`	Dedent the current line or selection
`Ctrl + A`	Select all content in the cell
`Ctrl + Z`	Undo the last change
`Ctrl + Y`	Redo the last undone change
`Ctrl + Shift + -`	Split the current cell at cursor position into two cells
`Ctrl + S` (or `Cmd + S`)	Save the notebook
`Ctrl + Shift + -`	Split the cell at the current cursor position

These shortcuts are optimized for efficient coding and content editing, significantly reducing reliance on the mouse and improving productivity within Jupyter Notebooks.

6 Lauching a new notebook or terminal

Pick the folder icon in the side pane menu on the left, click the big blue button and launch a notebook

7 Multiple notebooks or terminal views

8 Split windows

9 Controls

9.1 Menu, Toolbar, Context menu

9.2 Command palette: `Command/Ctrl Shift C`

9.3 Keyboard short-cuts: `Command/Ctrl Shift H`

10 Three kinds of cells

```
print('hello world')
```

```
hello world
```

This is *formatted* markdown

This is raw text

11 Running code in cells

11.1 Code blocks

```
x = 0
```

11.2 Last value is displayed

```
x = 1
x
```

```
1
```

11.3 Everything is one Python process

```
x += 1
x
```

```
2
```

11.4 Restarting the kernel

12 Two kinds of “Undo”

- `Edit/Undo` to undo stuff in the current cell
- `Edit/Undo Cell Operation` to undo deleting, moving, merging cells

13 Moving and copy/paste cells

```
print("Move this cell somewhere")
```

```
Move this cell somewhere
```

Some other cell

14 Split and merge cells

15 Markdown text

https://quarto.org/docs/authoring/markdown-basics.html

This is some more markdown with **bold** text and *italics* and showing that somthing is a code, like `x = 2 + 2`.

15.1Lists

- foo
- bar
- baz

15.2Numbered lists

1. foo
2. bar
3. baz

15.3Quotes

> This is a quote

15.4Formulas

$\sum_{i=0}^{n} i$

HTML

15.5Header 2

15.5.1 Header 3

Header 4 15-1

15.6Formulas

$\sum_{i=0}^{n} i$

15.7Tables

Name	Value
foo	2
bar	3

16 HTML magic

```
%%html
<style>
table {float:left}
</style>
```

```
<IPython.core.display.HTML object>
```

17 Images

Make markdown cell and then drag an image into it:

Some other cell

18 Table of content and cell folding

19 Shell commands

```
! ls
```

```
BP TAs.md
ERDA rollout 260923.pptx
FemaleDEGall.pdf
Harmonic mean.xlsx
Marias plots
My Drive 08.10.50
Pictures
Screenshot 2023-09-28 at 14.03.06.png
Screenshot 2023-10-23 at 15.59.13.png
Training project.pdf
Untitled.ipynb
hc38_ECH_90%_regions.txt
jupyter_walk_through.ipynb
nb_20-updated_sweep_enrichments (3).html
nn.png
prob_of_nr_runs.ipynb
rejsUd
runs_of_ones_paper.pdf
timing_code.ipynb
```

20 Developing exercises

20.1TLDR;

```
conda create -n franklin -c conda-forge munch-group::franklin
conda activate franklin

franklin exercise down
cd <repo>
franklin jupyter run
franklin up
cd ..
```

20.2Create an exercise repository

A git repository is where an exercise lives

- **exercise.ipynb:**: blah blah
- **Dockerfile:** blah blah
- **README.md:**: blah blah
- **docker-entrypoint.sh:**: blah blah
- **tagged-release.sh:**: blah blah

20.3Develop/change a new exercise on GitLab

21 Data analysis in Python and R

21.1R

```r
fizz_buzz <- function(fbnums = 1:50) {
  output <- dplyr::case_when(
    fbnums %% 15 == 0 ~ "FizzBuzz",
    fbnums %% 3 == 0 ~ "Fizz",
    fbnums %% 5 == 0 ~ "Buzz",
    TRUE ~ as.character(fbnums)
  )
  print(output)
}
```

21.2Python

```python
def fizz_buzz(num):
  if num % 15 == 0:
    print("FizzBuzz")
  elif num % 5 == 0:
    print("Buzz")
  elif num % 3 == 0:
    print("Fizz")
  else:
    print(num)
```

21.3R

```r
fizz_buzz <- function(fbnums = 1:50) {
  output <- dplyr::case_when(
    fbnums %% 15 == 0 ~ "FizzBuzz",
    fbnums %% 3 == 0 ~ "Fizz",
```

```
    fbnums %% 5 == 0 ~ "Buzz",
    TRUE ~ as.character(fbnums)
  )
  print(output)
}
```

21.4Python

```
def fizz_buzz(num):
  if num % 15 == 0:
    print("FizzBuzz")
  elif num % 5 == 0:
    print("Buzz")
  elif num % 3 == 0:
    print("Fizz")
  else:
    print(num)
```

22 Numpy and Pandas

Fast computation using vectors and matrices

```
list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

summed = []
for i in range(len(list1)):
    summed.append(list1[i] + list2[i])
summed
```

```
[9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
```

22.1 Arrays

```
import numpy as np

a = np.array(list1)
b = np.array(list2)
a, b
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0]))
```

22.2 Broadcasting

```
a + b
```

```
array([9, 9, 9, 9, 9, 9, 9, 9, 9, 9])
```

```
a * b
```

```
array([ 0,  8, 14, 18, 20, 20, 18, 14,  8,  0])
```

```
a - 10
```

```
array([-10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1])
```

```
a.sum()
```

```
45
```

```
a.mean()
```

```
4.5
```

22.3Multidimentional arrays

```
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list_of_lists
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
list_of_lists[1][1]
```

```
5
```

```
matrix = np.array(list_of_lists)
matrix
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
matrix[1][1] # not efficient
```

```
5
```

```
matrix[1, 1] # efficient
```

```
5
```

```
matrix - 10
```

```
array([[-9, -8, -7],
       [-6, -5, -4],
       [-3, -2, -1]])
```

```
matrix.sum()
```

```
45
```

```
list_of_lists_of_lists = [[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[1, 2, 3], [4, 5, 6], [7, 8, 9]]]
list_of_lists_of_lists
```

```
[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[1, 2, 3], [4, 5, 6], [7, 8, 9]]]
```

```
tensor = np.array(list_of_lists_of_lists)
tensor
```

```
array([[[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]],

       [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]])
```

```
tensor[1, 1, 1]
```

```
5
```

23 Pandas

Fast computations on data tables (on top of Numpy).

```
import pandas as pd
```

23.1 DataFrame

```
df = pd.DataFrame({'name': ['Mike', 'Mia', 'Jake'], 'weight': [82, 62, 75]})
df
```

	name	weight
0	Mike	82
1	Mia	62
2	Jake	75

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df = pd.DataFrame(dict(name=['Mike', 'Mia', 'Jake'], weight=[82, 62, 75]))
df
```

	name	weight
0	Mike	82
1	Mia	62
2	Jake	75

```
records = [('Mike', 82), ('Mia', 62), ('Jake', 75)]

df = pd.DataFrame().from_records(records, columns=['age', 'weight'])
df
```

	age	weight
0	Mike	82
1	Mia	62
2	Jake	75

```
df.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
df.index.values
```

```
array([0, 1, 2])
```

```
df.columns
```

```
Index(['age', 'weight'], dtype='object')
```

```
df.dtypes
```

```
age       object
weight     int64
dtype: object
```

Add a column to an existing dataframe:

```
df['height'] = [182.5, 173.0, 192.5]
df
```

	age	weight	height
0	Mike	82	182.5
1	Mia	62	173.0
2	Jake	75	192.5

Add another, categorical, column:

```
df['sex'] = pd.Categorical(['male', 'female', 'male'], categories=['female', 'male'], ordered=True)
df
```

	age	weight	height	sex
0	Mike	82	182.5	male
1	Mia	62	173.0	female
2	Jake	75	192.5	male

```
df.dtypes
```

```
age         object
weight       int64
height     float64
sex       category
dtype: object
```

A Series just wraps an array:

```
df.height.to_numpy()
```

```
array([182.5, 173. , 192.5])
```

23.2Example penguin data set

```
import seaborn as sns

penguins = sns.load_dataset('penguins')

penguins
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0
...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0

```
penguins.dtypes
```

```
species               object
island                object
bill_length_mm       float64
bill_depth_mm        float64
flipper_length_mm    float64
body_mass_g          float64
sex                   object
dtype: object
```

```
penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0
3	Adelie	Torgersen	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0

```
penguins.tail()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0

23.3Series

```
penguins['flipper_length_mm']
```

```
0      181.0
1      186.0
2      195.0
3        NaN
4      193.0
       ...
339      NaN
340    215.0
341    222.0
342    212.0
343    213.0
Name: flipper_length_mm, Length: 344, dtype: float64
```

```
penguins.flipper_length_mm
```

```
0      181.0
1      186.0
2      195.0
3        NaN
4      193.0
       ...
339      NaN
340    215.0
341    222.0
342    212.0
343    213.0
Name: flipper_length_mm, Length: 344, dtype: float64
```

```
type(penguins.flipper_length_mm)
```

```
pandas.core.series.Series
```

23.4Broadcasting

```
penguins.bill_depth_mm - 1000
```

```
0     -981.3
1     -982.6
2     -982.0
3        NaN
4     -980.7
       ...
339      NaN
340   -985.7
341   -984.3
342   -985.2
343   -983.9
Name: bill_depth_mm, Length: 344, dtype: float64
```

```
penguins.bill_depth_mm * penguins.flipper_length_mm
```

```
0      3384.7
1      3236.4
2      3510.0
3         NaN
4      3724.9
        ...
339       NaN
340    3074.5
341    3485.4
342    3137.6
343    3429.3
Length: 344, dtype: float64
```

23.5Indexing

23.5.1 Get a cell

```
penguins.loc[4, 'island']
```

```
'Torgersen'
```

23.5.2 Get a row

```
penguins.loc[4]
```

```
species                Adelie
island               Torgersen
bill_length_mm            36.7
bill_depth_mm             19.3
flipper_length_mm        193.0
body_mass_g             3450.0
sex                     Female
Name: 4, dtype: object
```

23.5.3 Get a column

```
penguins['bill_depth_mm']
```

```
0      18.7
1      17.4
2      18.0
3       NaN
4      19.3
       ...
339     NaN
340    14.3
341    15.7
342    14.8
343    16.1
Name: bill_depth_mm, Length: 344, dtype: float64
```

```
penguins.bill_depth_mm
```

```
0      18.7
1      17.4
2      18.0
3       NaN
4      19.3
       ...
339     NaN
```

```
340    14.3
341    15.7
342    14.8
343    16.1
Name: bill_depth_mm, Length: 344, dtype: float64
```

23.5.4 Get a range of rows and multiple columns

```
penguins.loc[40:45, ['island', 'body_mass_g']]
```

	island	body_mass_g
40	Dream	3150.0
41	Dream	3900.0
42	Dream	3100.0
43	Dream	4400.0
44	Dream	3000.0
45	Dream	4600.0

23.5.5 Use boolean series as index to subset data

```
idx = penguins.bill_length_mm > 55
idx
```

```
0      False
1      False
2      False
3      False
4      False
       ...
339    False
340    False
341    False
342    False
343    False
Name: bill_length_mm, Length: 344, dtype: bool
```

```
penguins.loc[idx]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
169	Chinstrap	Dream	58.0	17.8	181.0	3700.0	Female
215	Chinstrap	Dream	55.8	19.8	207.0	4000.0	Male
253	Gentoo	Biscoe	59.6	17.0	230.0	6050.0	Male
321	Gentoo	Biscoe	55.9	17.0	228.0	5600.0	Male
335	Gentoo	Biscoe	55.1	16.0	230.0	5850.0	Male

```
penguins.loc[(penguins.bill_length_mm > 55) & (penguins.sex == 'Female')]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
169	Chinstrap	Dream	58.0	17.8	181.0	3700.0	Female

23.5.6 Setting and resetting the index

```
penguins
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

```
df = penguins.set_index(['species', 'sex', 'island'])
df.head(10)
```

species	sex
Adelie	Male
	Female
	NaN
	Female
	Male
	Female
	Male
	NaN

```
df.reset_index()
```

	species	sex	island	bill_length_mm	bill_depth_mm	flipper_length_mm	boc
0	Adelie	Male	Torgersen	39.1	18.7	181.0	375
1	Adelie	Female	Torgersen	39.5	17.4	186.0	380
2	Adelie	Female	Torgersen	40.3	18.0	195.0	325
3	Adelie	NaN	Torgersen	NaN	NaN	NaN	Na
4	Adelie	Female	Torgersen	36.7	19.3	193.0	345
...	...	...	...	...	...	...	...
339	Gentoo	NaN	Biscoe	NaN	NaN	NaN	Na
340	Gentoo	Female	Biscoe	46.8	14.3	215.0	485
341	Gentoo	Male	Biscoe	50.4	15.7	222.0	575
342	Gentoo	Female	Biscoe	45.2	14.8	212.0	520
343	Gentoo	Male	Biscoe	49.9	16.1	213.0	540

23.6Sorting rows

```
sorted_df = penguins.sort_values(by="bill_length_mm")
sorted_df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
142	Adelie	Dream	32.1	15.5	188.0	3050.0
98	Adelie	Dream	33.1	16.1	178.0	2900.0
70	Adelie	Torgersen	33.5	19.0	190.0	3600.0

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
92	Adelie	Dream	34.0	17.1	185.0	3400.0	Female
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN

```
sorted_df.index.values
```

```
array([142,  98,  70,  92,   8,  18,  54,  80,  14, 100,  52,  83, 124,
        25,  66,  74, 136,  60,  90, 118,  68,  22,  42,  48, 150, 148,
        78,  94, 120,  86,  34,  64,  58,  40,  15, 147,   4,  82, 132,
        87,  44, 138,  77,  31, 144, 117,  84,  47, 133,  62,  38,  59,
        21, 121, 102, 103,  10,  20,  11, 149, 104,  28,  96, 108, 134,
       110, 107,  23,  88, 130,  13, 106, 116,  16,  24, 126,  36,  89,
         6, 128, 145,  56,   0,  35, 146,   7,   5,  30,   1,  32, 114,
        45,  50,  72,  93, 112, 105, 139,  71,  39,  51, 137, 140, 122,
        97,   2,  27,  29, 125, 141,  26,  57, 143,  95,  41, 230, 182,
        33,  76, 101, 135, 119,  46,  63,  91,  67,  12,  61,  85, 123,
        55, 127, 151,  65, 326,  69, 236,  53,   9,  79, 113,  37,  49,
       172, 184, 206,  17, 256, 115, 260,  75, 251,  81, 244, 131, 278,
       109, 174,  99, 228, 328, 306, 216, 332, 288, 265, 276, 258, 129,
        43, 257, 246, 336, 314, 268, 304, 275, 241, 252, 272, 208, 298,
       299, 269, 342, 157, 280, 262, 226, 155, 232, 277, 195, 312, 266,
       111, 211, 214, 204, 282,  73, 284, 234, 166, 160,  19, 158, 245,
       220, 286, 238, 334, 281, 193, 243, 170, 180, 291, 294, 293, 225,
       274, 152, 242, 162, 270, 227, 176, 325, 229, 340, 213, 317, 190,
       164, 338, 322, 324, 250, 302, 310, 296, 187, 308, 188, 224, 290,
       247, 329, 202, 248, 292, 318, 233, 255, 271, 173, 320, 295, 259,
       239, 222, 337, 192, 199, 231, 300, 323, 254, 171, 237, 235, 209,
       316, 313, 179, 287, 263, 261, 217, 186, 331, 201, 285, 343, 303,
       153, 221, 223, 249, 198, 273, 210, 219, 240, 168, 279, 341, 267,
       330, 167, 178, 264, 175, 289, 205, 305, 218, 197, 315, 196, 194,
       185, 297, 319, 154, 159, 161, 307, 203, 333, 200, 163, 212, 165,
       177, 189, 309, 207, 311, 301, 156, 181, 327, 191, 183, 283, 335,
       215, 321, 169, 253,   3, 339])
```

Click to the left of an output cell to enable/disable scrolling of the output (usefull for large amounts of output).

```
sorted_df.loc[0]
```

```
species                Adelie
island              Torgersen
```

```
bill_length_mm            39.1
bill_depth_mm             18.7
flipper_length_mm        181.0
body_mass_g             3750.0
sex                       Male
Name: 0, dtype: object
```

```
sorted_df.flipper_length_mm[0]
```

```
181.0
```

```
sorted_df.iloc[0] # iloc !!!
```

```
species              Adelie
island               Dream
bill_length_mm        32.1
bill_depth_mm         15.5
flipper_length_mm    188.0
body_mass_g         3050.0
sex                 Female
Name: 142, dtype: object
```

```
sorted_df.flipper_length_mm.iloc[0]
```

```
188.0
```

23.7Summary stats

```
penguins.describe()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	342.000000	342.000000	342.000000	342.000000
mean	43.921930	17.151170	200.915205	4201.754386
std	5.459584	1.974793	14.061714	801.954536
min	32.100000	13.100000	172.000000	2700.000000
25%	39.225000	15.600000	190.000000	3550.000000
50%	44.450000	17.300000	197.000000	4050.000000

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

```
penguins.bill_length_mm.mean()
```

```
43.9219298245614
```

```
penguins.bill_length_mm.count()
```

```
342
```

23.8Grouping

```
penguins.groupby('island')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x15cc9e5e0>
```

23.8.1 Aggregate

Aggregating produces a **single** value for each variable in each group:

Means for all numeric variables for each island:

```
penguins.groupby('island').aggregate("mean", numeric_only=True)
```

island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
Biscoe	45.257485	15.874850	209.706587	4716.017964
Dream	44.167742	18.344355	193.072581	3712.903226
Torgersen	38.950980	18.429412	191.196078	3706.372549

```
penguins.groupby('island').mean(numeric_only=True)
```

island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
Biscoe	45.257485	15.874850	209.706587	4716.017964
Dream	44.167742	18.344355	193.072581	3712.903226
Torgersen	38.950980	18.429412	191.196078	3706.372549

Means for `bill_length_mm` and `flipper_length_mm`:

```
penguins.groupby('island')[['bill_length_mm', 'flipper_length_mm']].mean()
```

island	bill_length_mm	flipper_length_mm
Biscoe	45.257485	209.706587
Dream	44.167742	193.072581
Torgersen	38.950980	191.196078

Just for `flipper_length_mm`:

```
penguins.groupby('island').flipper_length_mm.mean()
```

```
island
Biscoe       209.706587
Dream        193.072581
Torgersen    191.196078
Name: flipper_length_mm, dtype: float64
```

23.8.2 Transform

Transforming produces new colums with the *same length* as the input:

```
penguins.groupby('island')[['bill_length_mm', 'flipper_length_mm']].transform("mean")
```

	bill_length_mm	flipper_length_mm
0	38.950980	191.196078
1	38.950980	191.196078
2	38.950980	191.196078
3	38.950980	191.196078

	bill_length_mm	flipper_length_mm
4	38.950980	191.196078
...	...	...
339	45.257485	209.706587
340	45.257485	209.706587
341	45.257485	209.706587
342	45.257485	209.706587
343	45.257485	209.706587

```
def z_value(sr):
    return (sr - sr.mean()) / sr.std()

penguins.groupby('island')[['bill_length_mm', 'flipper_length_mm']].transform(z_value)
```

	bill_length_mm	flipper_length_mm
0	0.049258	-1.636022
1	0.181475	-0.833742
2	0.445910	0.610362
3	NaN	NaN
4	-0.744048	0.289450
...	...	...
339	NaN	NaN
340	0.323193	0.374297
341	1.077478	0.869267
342	-0.012044	0.162167
343	0.972717	0.232877

23.8.3 Apply

Flexible method allowing any operation on grouped data.

Return a single value:

```
def fun(df):
    return df.bill_length_mm + df.flipper_length_mm.mean() / df.body_mass_g

penguins.groupby('island').apply(fun)#.to_frame('my_stat')


island
Biscoe     20     37.861678
```

```
            21     37.758252
            22     35.955186
            23     38.253090
            24     38.855186
                       ...
Torgersen   127    41.544464
            128    39.062687
            129    44.147799
            130    38.557503
            131    43.154627
Length: 344, dtype: float64
```

Return a dataframe:

```
def fun(df):
    return pd.DataFrame({'sqrt_bill': np.sqrt(df.bill_length_mm),
                         'bill_squared': df.bill_length_mm**2})

penguins.groupby('island').apply(fun)
```

island		sqrt
Biscoe	20	6.14
	21	6.14
	22	5.99
	23	6.18
	24	6.22
...	...	...
Torgersen	127	6.44
	128	6.24
	129	6.64
	130	6.20
	131	6.56

24 Plots with matplotlib

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
sns.set_style("darkgrid")

# make graphics sharper on a good screen
from matplotlib_inline.backend_inline import set_matplotlib_formats
set_matplotlib_formats('retina', 'png')


penguins = sns.load_dataset("penguins")
penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

```
plt.scatter(penguins.bill_length_mm, penguins.flipper_length_mm) ;  # semi colon makes last value
```

```
sns.set_style("ticks")
# sns.set_style("darkgrid")
# sns.set_style("whitegrid")
# sns.set_style("white")
# sns.set_style("dark")

plt.scatter(penguins.bill_length_mm, penguins.flipper_length_mm)
sns.despine()
```

```
plt.hist(penguins.bill_length_mm) ;
```

25 Better plotting with seaborn (on top of matplotlib)

```
sns.scatterplot(data=penguins, x="bill_length_mm", y="flipper_length_mm") ;

sns.scatterplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species") ;

sns.scatterplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species", style="se

sns.scatterplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species",
                style="sex", size="body_mass_g") ;

def legend_outside():
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

sns.scatterplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species", style="se

legend_outside()

plt.title("Penguin measurements")
plt.ylabel("flipper length (mm)")
plt.xlabel("bill length in (mm)") ;
```

26 Wide or long data

Wide format data:

```
penguins[['bill_length_mm', 'bill_depth_mm']]
```

	bill_length_mm	bill_depth_mm
0	39.1	18.7
1	39.5	17.4
2	40.3	18.0
3	NaN	NaN
4	36.7	19.3
...	...	...
339	NaN	NaN
340	46.8	14.3
341	50.4	15.7
342	45.2	14.8
343	49.9	16.1

Long format data:

```
long_df = penguins.melt(value_vars=['bill_length_mm', 'bill_depth_mm'])
long_df
```

	variable	value
0	bill_length_mm	39.1
1	bill_length_mm	39.5
2	bill_length_mm	40.3
3	bill_length_mm	NaN
4	bill_length_mm	36.7
...	...	...
683	bill_depth_mm	NaN
684	bill_depth_mm	14.3
685	bill_depth_mm	15.7

	variable	value
686	bill_depth_mm	14.8
687	bill_depth_mm	16.1

long format is required when you want a number of columns to appear as a "variable" in the plot. As in the example below where the color "variable" reflects whether the point is `bill_depth_mm` or `bill_length_mm`.

Retain other information for each observation:

```
long_df = penguins.melt(id_vars=['species', 'body_mass_g', 'island'], value_vars=['bill_
long_df
```

	species	body_mass_g	island	variable	value
0	Adelie	3750.0	Torgersen	bill_length_mm	39.1
1	Adelie	3800.0	Torgersen	bill_length_mm	39.5
2	Adelie	3250.0	Torgersen	bill_length_mm	40.3
3	Adelie	NaN	Torgersen	bill_length_mm	NaN
4	Adelie	3450.0	Torgersen	bill_length_mm	36.7
...	...	...	...	...	...
683	Gentoo	NaN	Biscoe	bill_depth_mm	NaN
684	Gentoo	4850.0	Biscoe	bill_depth_mm	14.3
685	Gentoo	5750.0	Biscoe	bill_depth_mm	15.7
686	Gentoo	5200.0	Biscoe	bill_depth_mm	14.8
687	Gentoo	5400.0	Biscoe	bill_depth_mm	16.1

```
sns.scatterplot(data=long_df, x='body_mass_g', y='value', hue='variable', style='species
legend_outside()
```

```
sns.boxplot(data=long_df, x='species', y='value', hue='variable') ;
```

```
sns.boxplot(data=long_df, x='variable', y='value', hue='species') ;
```

```
sns.boxplot(data=long_df, x='species', y='value', hue='variable') ;
```

27 Plots with multiple facets

```
g = sns.FacetGrid(penguins, col="island")
g ;
```

Map plotting to each facet:

```
g = sns.FacetGrid(penguins, col="island", hue="species", height=3)
g.map(sns.scatterplot, "bill_length_mm", "flipper_length_mm") ;
```

Grid of facets representing combinations of two variables:

```
g = sns.FacetGrid(penguins, row="sex", col="island", hue="species", height=3) ;
g.map(sns.scatterplot, "bill_length_mm", "flipper_length_mm") ;
```

```
g = sns.FacetGrid(penguins, row="sex", col="island", hue="species", height=3) ;
g.map(sns.regplot, "bill_length_mm", "flipper_length_mm") ;
```

```
sns.lmplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", row="sex", col="island", hue=
```

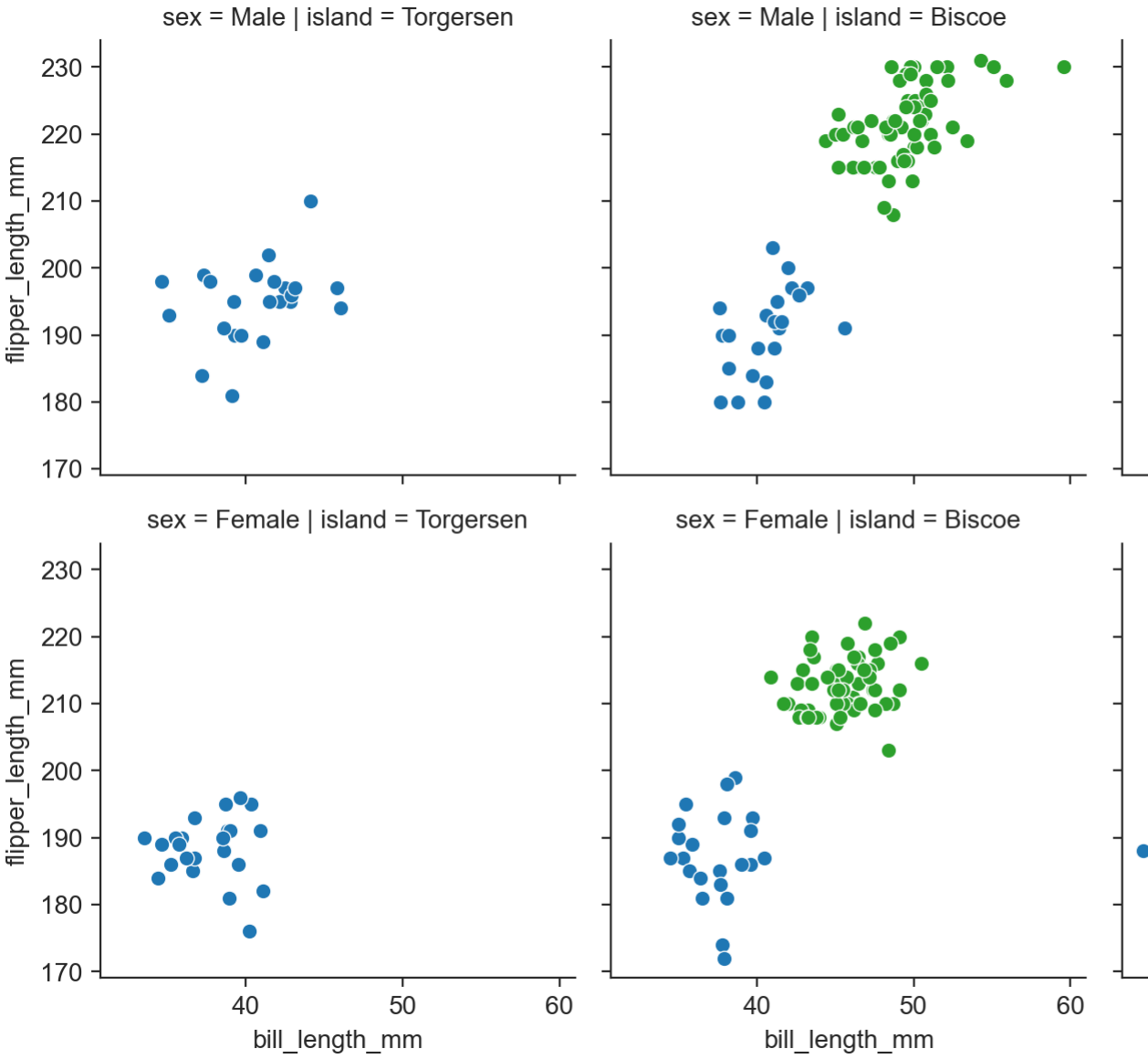
27.1 FacetGrid.map vs. FacetGrid.map_dataframe

When you use `FacetGrid.map(func, "col1", "col2", ...)`, the function `func` is passed the values of the columns `"col1"` and `"col2"` (and more if needed) as parameters 1 and 2 (`args[0]`, `args[1]`, ...). In addition, the function always receives a keyword argument named `color=`.

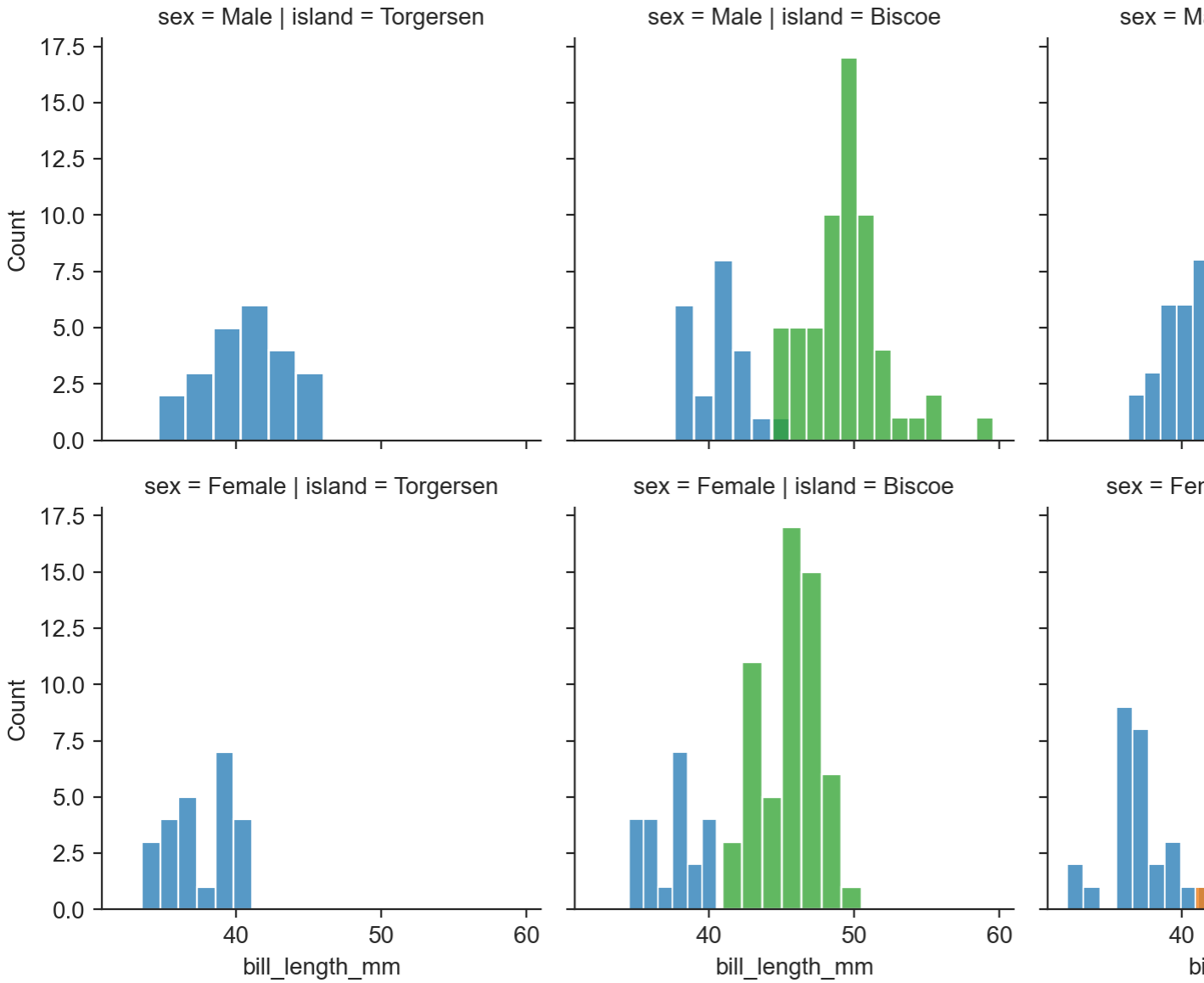
```
def scatter(*args, **kwargs):
    return plt.scatter(args[0], args[1], **kwargs)

g = sns.FacetGrid(penguins, row="sex", col="island", hue="species") ;
g.map(scatter, "bill_length_mm", "flipper_length_mm") ;
```

When you use `FacetGrid.map_dataframe(func, "col1", "col2", ...)`, the function `func` is passed the names `"col1"` and `"col2"` (and more if needed) as parameters 1 and 2 (`args[0]`, `args[1]`, ...), and the filtered dataframe as keyword argument `data=`. In addition, the function always receives a keyword argument named `color=`.

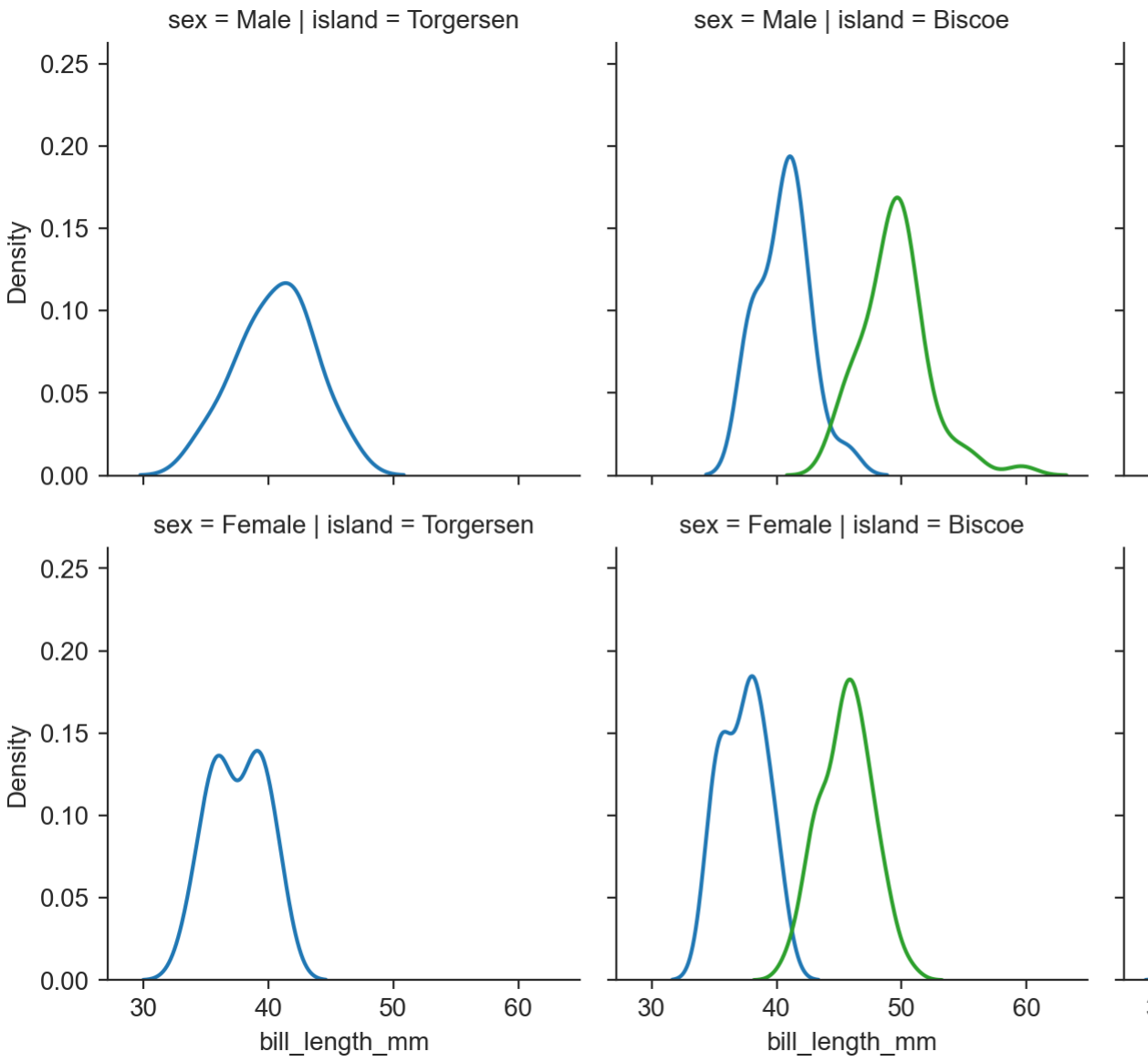
```
def scatterplot(*args, **kwargs):
    return sns.scatterplot(x=args[0], y=args[1], **kwargs)

g = sns.FacetGrid(penguins, row="sex", col="island", hue="species") ;
g.map_dataframe(scatterplot, "bill_length_mm", "flipper_length_mm") ;
```



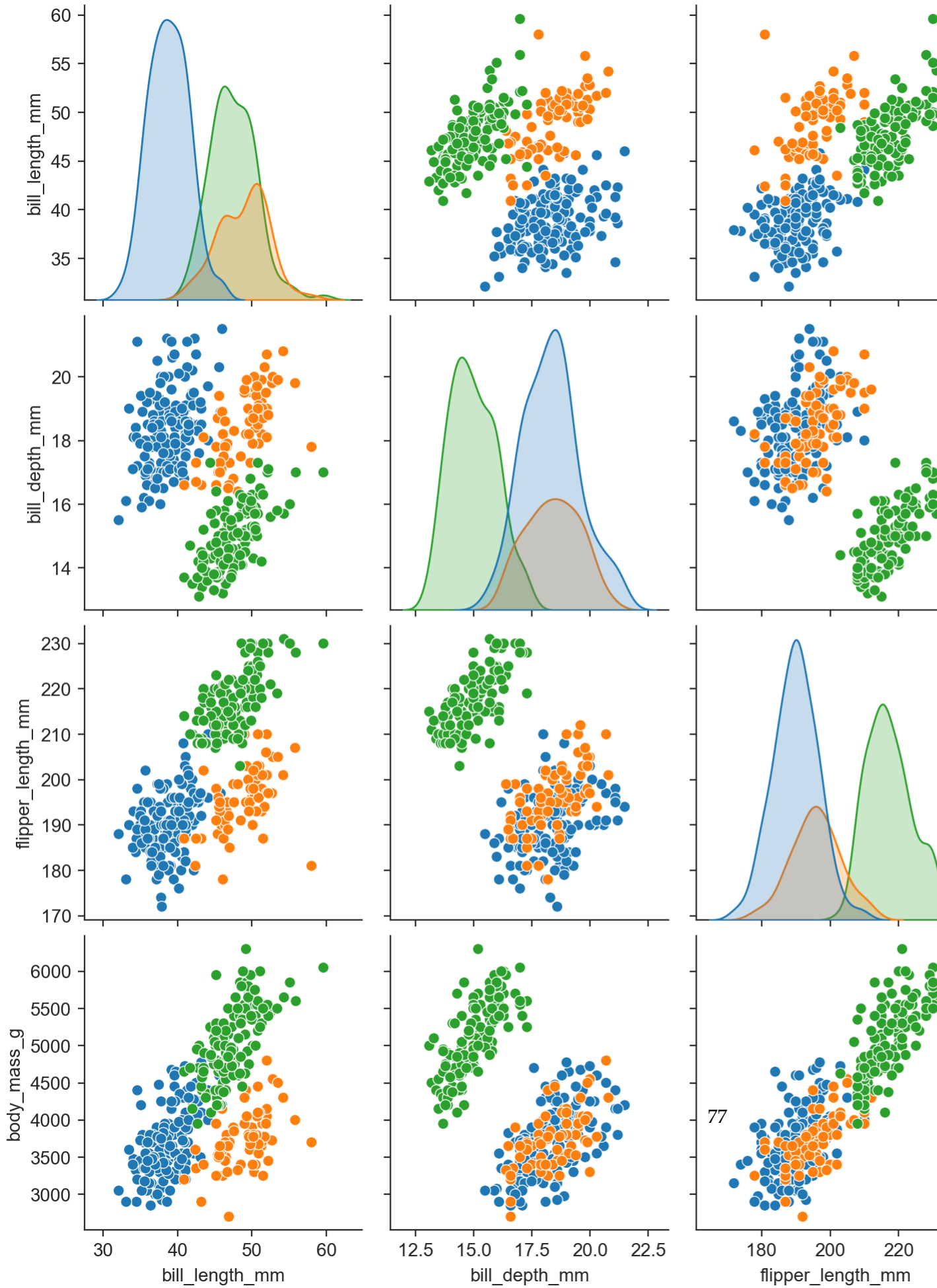
```
g = sns.FacetGrid(penguins, row="sex", col="island", hue="species") ;
g.map(sns.histplot, "bill_length_mm") ;
```



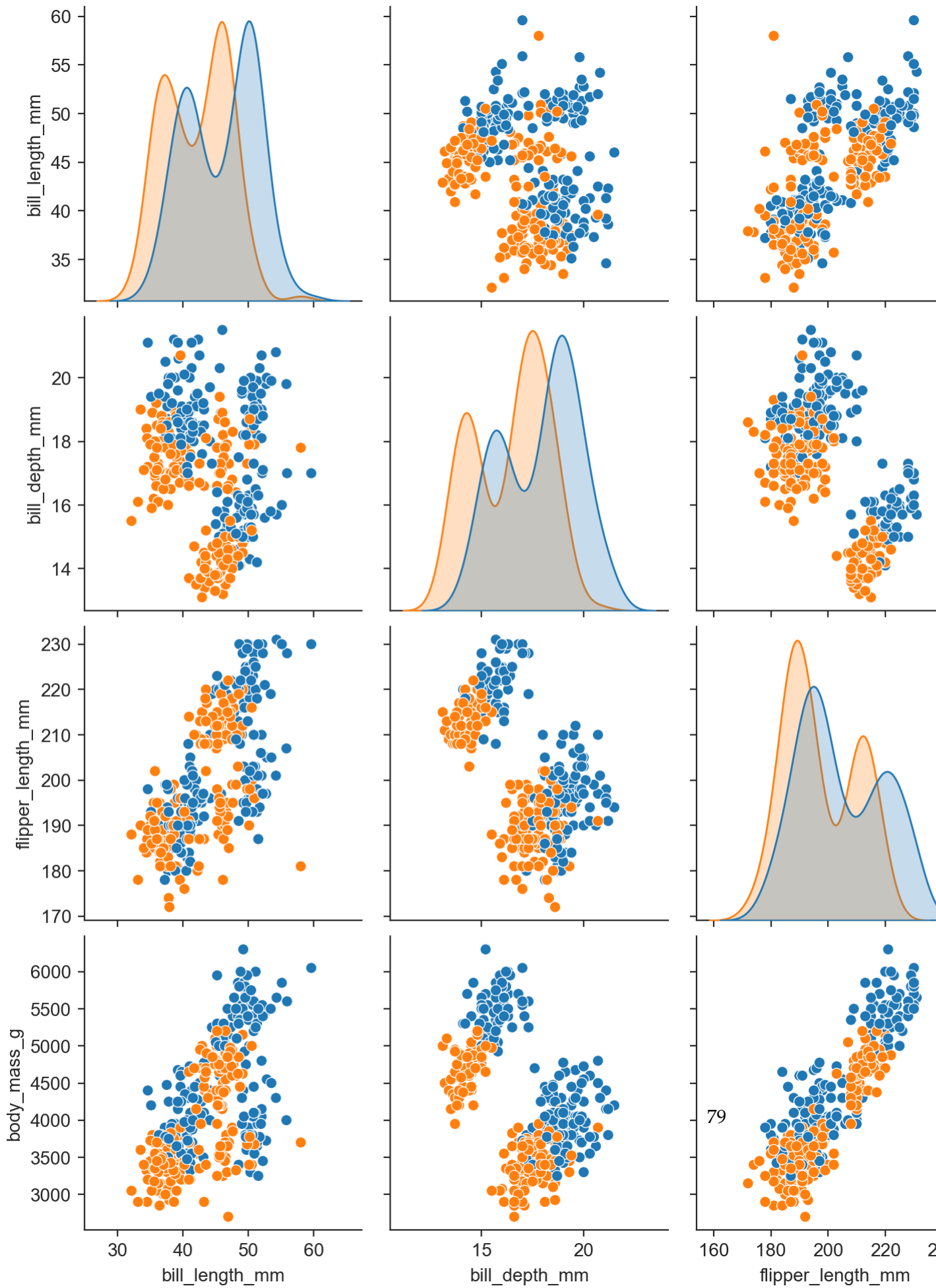

```
g = sns.FacetGrid(penguins, row="sex", col="island", hue="species") ;
g.map(sns.kdeplot, "bill_length_mm") ;
```




```
sns.pairplot(penguins, hue="species") ;
```


bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g
60
55
50
45
40
35
20
18
16
14
230
220
210
200
190
180
170
6000
5500
5000
4500
4000
3500
3000
30
40
50
60
12.5
15.0
17.5
20.0
22.5
180
200
220

```
sns.pairplot(penguins, hue="sex") ;
```


bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g
60
55
50
45
40
35
20
18
16
14
230
220
210
200
190
180
170
6000
5500
5000
4500
4000
3500
3000
30
40
50
60
15
20
160
180
200
220