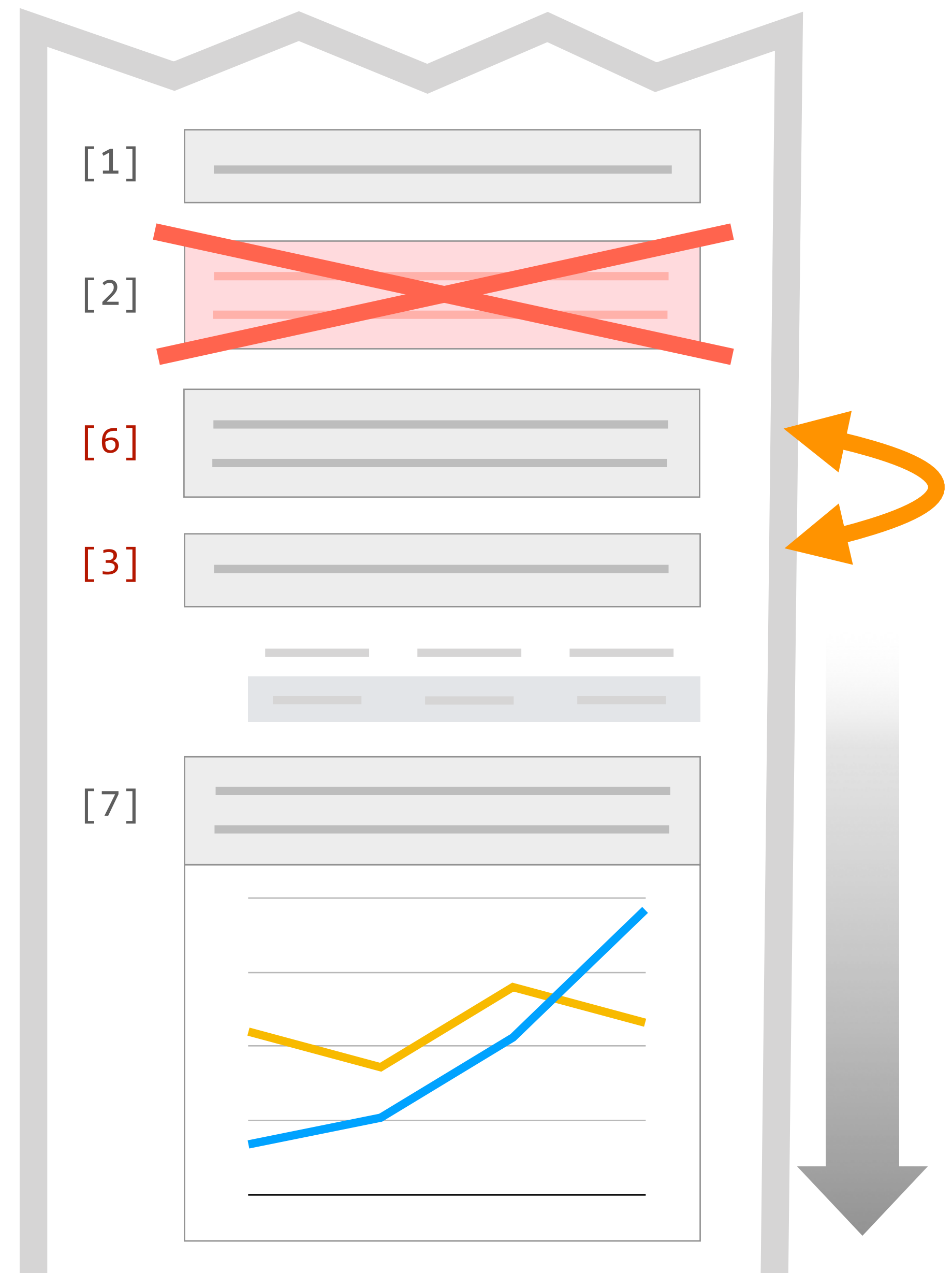


# Managing Messes in Computational Notebooks

Andrew Head · Fred Hohman ·

Titus Barik · Steven M. Drucker · and Robert DeLine

UC Berkeley · Georgia Tech · Microsoft Research



# Computational Notebooks: Code, Text, and Output

I ♥ [xkcd](#)! It reliably provides top-rate [insights](#), [humor](#), or [both](#). I was thrilled when I got to [introduce Randall Monroe](#) for a talk in 2007. But in [xkcd #1313](#),



```
In [1]: from __future__ import division, print_function
import re
import itertools

def words(text): return set(text.split())

winners = words(''washington adams jefferson jefferson madis
monroe adams jackson jackson van-buren harrison polk tayl
lincoln lincoln grant graptant hayes garfield cleveland
mckinley roosevelt taft wilson wilson harding coolidge h
roosevelt roosevelt roosevelt truman eisenhower eisenhow
nixon carter reagan reagan bush clinton clinton bush bush

losers = words(''clinton jefferson adams pinckney pinckney <
jackson adams clay van-buren van-buren clay cass scott fi
mcclellan seymour greeley tilden hancock blaine cleveland
parker bryan roosevelt hughes cox davis smith hoover lanc
stevenson stevenson nixon goldwater humphrey mcgovern for
dukakis bush dole gore kerry mccain romney'')
```

We can see that there are multiple names that are both winners and losers:

```
In [2]: winners & losers
```

```
Out[2]: {'adams',
'bush',
'carter',
'cleveland',
'clinton',
'harrison',
'hoover',
'jackson',
'jefferson',
'nixon',
'roosevelt'}
```

Rich descriptions

Code

Output



# Notebook Programming Interfaces Abound



I ♥ [xkcd](#)! It reliably provides top-rate [insights](#), [humor](#), or [both](#). I was thrilled when I got to [introduce Randall Monroe](#) for a talk in 2007. But in [xkcd #1313](#),



```
In [1]: from __future__ import division, print_function
import re
import itertools

def words(text): return set(text.split())

winners = words('''washington adams jefferson jefferson madis
monroe adams jackson jackson van-buren harrison polk tayl
lincoln lincoln grant graptartnt hayes garfield cleveland
mckinley roosevelt taft wilson wilson harding coolidge hc
roosevelt roosevelt roosevelt truman eisenhower eisenhowe
nixon carter reagan reagan bush clinton clinton bush bush

losers = words('''clinton jefferson adams pinckney pinckney c
jackson adams clay van-buren van-buren clay cass scott f
mcclellan seymour greeley tilden hancock blaine cleveland
parker bryan roosevelt hughes cox davis smith hoover lanc
stevenson stevenson nixon goldwater humphrey mcgovern fo
dukakis bush dole gore kerry mccain romney''')
```

We can see that there are multiple names that are both winners and losers:

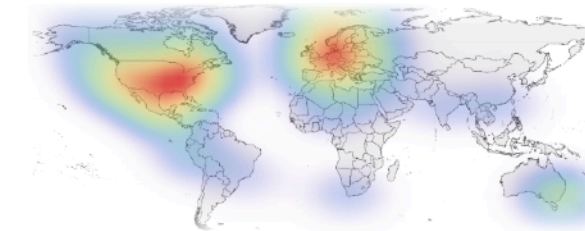
```
In [2]: winners & losers
```

```
Out[2]: {'adams',
'bush',
'carter',
'cleveland',
'clinton',
'harrison',
...}
```



## Introduction

Examining the information available through social media can tell you a lot about your online presence. This is a quick exploration of some social media analytics, with examples using a personal Twitter account.

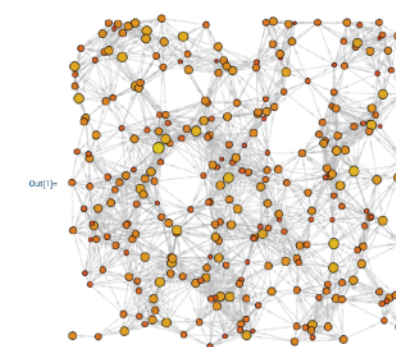


## Getting Data

In order to do any meaningful analysis, we'll need to pull data from a few different sources. The first step is to connect to a social network—in this case, Twitter. We'll also pull in some geographic data from the Wolfram Knowledgebase, as well as some data collected through an email survey.

Import a follower network from Twitter:

```
net = network = ServiceExecute["Twitter", "FollowerNetwork"]
```



Grab entities for the 30 nearest cities:

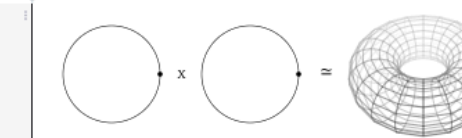
```
cities = GeoEntities[Santa Clara County, California, United States, "City"]
```

```
cities = {
  "Cambridge Park", "Fruitdale", "Burbank", "San Martin", "Los Altos Hills", "Sunol-Midway",
  "San Jose", "Morgan Hill", "Milpitas", "Lexington Hills", "Sunnyvale", "Gilroy", "East Foothills",
  "Loyola", "Santa Clara", "Los Altos", "Alum Rock", "Mountain View", "Cupertino"
}
```



## Torus Knots

First, what is a torus? The torus is a (cartesian) product of two circles, i.e.  $S^1 \times S^1$  where a  $S^1$  is the usual circle.



```
{
  var width = 640;
  var height = 210;

  var svg = d3.select(DOM.svg(width, height))
    .style("overflow", "visible");

  var c1 = GetInputCircle(100, 100, 70, svg, TorusCallback);
  var c2 = GetInputCircle(300, 100, 70, svg, TorusCallback);

  svg.append("text")
    .attr("x", 195)
    .attr("y", 185)
    .text("x");

  svg.append("text")
    .attr("x", 400)
    .attr("y", 185)
    .style("font-size", "34px")
    .text("y");

  var torus = GetTorus(svg, 550, 90, 350, 0.5, 20);
  var tp = torus.GetTorusPoint();
  tp.UpdateTorusPoint(c1.GetPoint(), c2.GetPoint());

  function TorusCallback() {
    tp.UpdateTorusPoint(c1.GetPoint(), c2.GetPoint());
  }

  return svg.node();
}
```

You are probably more used to the torus on the right, but for each pair of points in the torus on the right, you can associate a point in the torus on the left via a simple transformation. Given a point  $z \in S^1$  in the first circle and a point  $w \in S^1$  in the second circle, we can associate the following point in the torus:

$$f(z, w) = (z(1 + \alpha w), z(1 + \alpha w), \alpha w)$$

This transformation is what is called an *homeomorphism* – a continuous map with continuous inverse. In practice it means that the product of the two circles and the torus have the same shape. Drag the point in the circles on the left to see what is the corresponding point in the torus.

We can open the two circles into segments and then the cartesian product



Maple™



BeakerX

Iodide<sub>α</sub>



Apache Zeppelin

R Markdown



SOXE



databricks®

# Notebook Model of Exploratory Programming

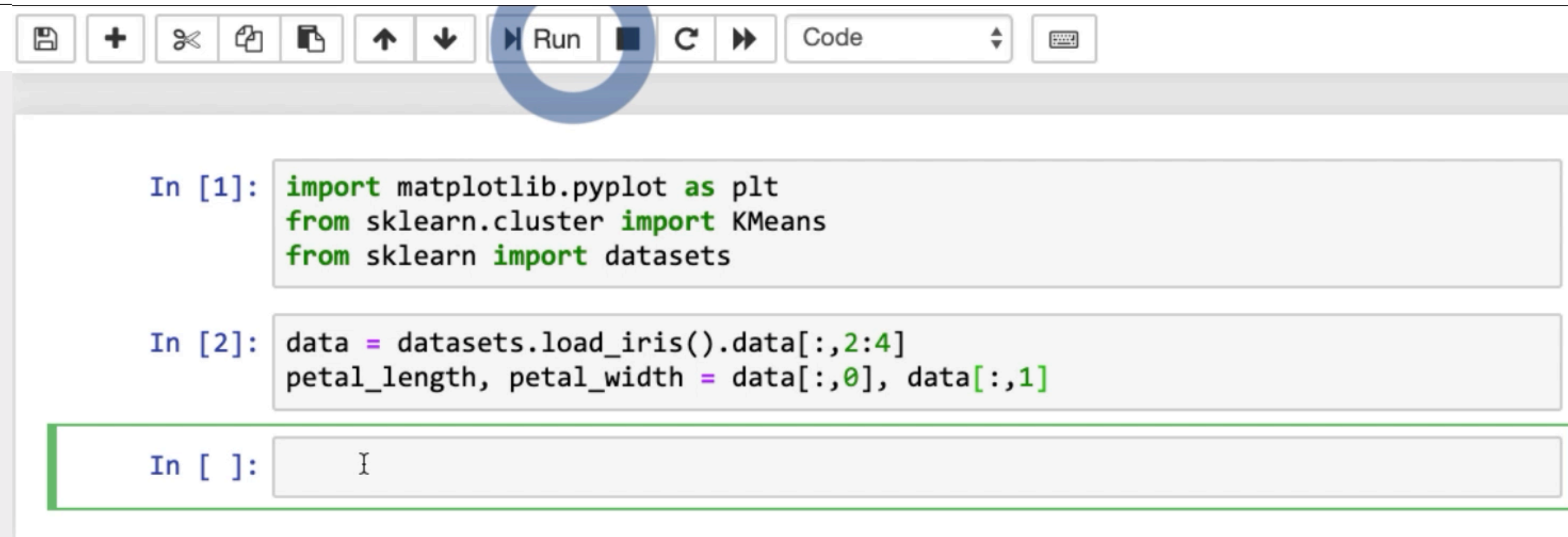


In [ ]:

1. Incremental execution



# Notebook Model of Exploratory Programming



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]

In [ ]: 
```

1. Incremental execution
2. In-situ output

# Notebook Model of Exploratory Programming



```
In [1]: import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import datasets
```

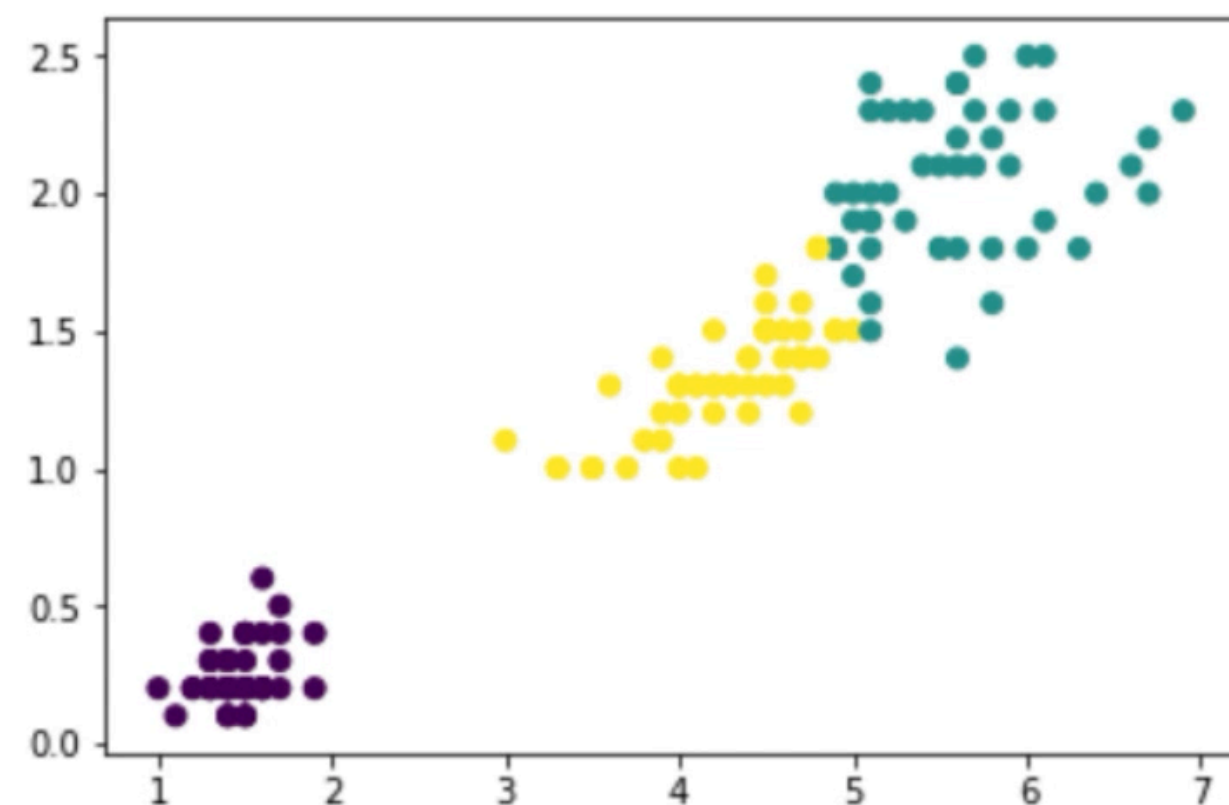
```
In [2]: data = datasets.load_iris().data[:,2:4]  
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))  
Average petal length: 3.758
```

```
In [4]: clusters = KMeans(n_clusters=3).fit(data).labels_
```

```
In [5]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[5]: <matplotlib.collections.PathCollection at 0x124e294e0>
```



```
In [ ]: |
```

1. Incremental execution
2. In-situ output
3. Incremental changes

# Notebook Model of Exploratory Programming



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

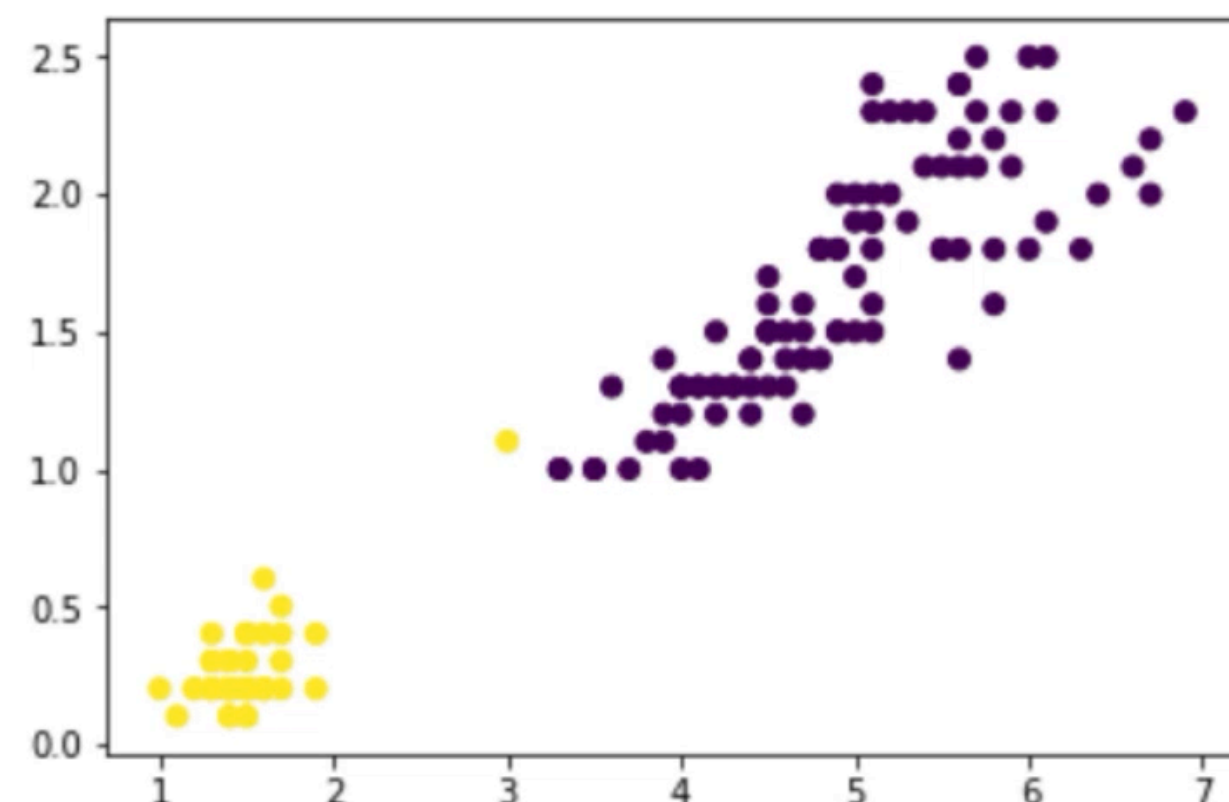
```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
Average petal length: 3.758
```

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

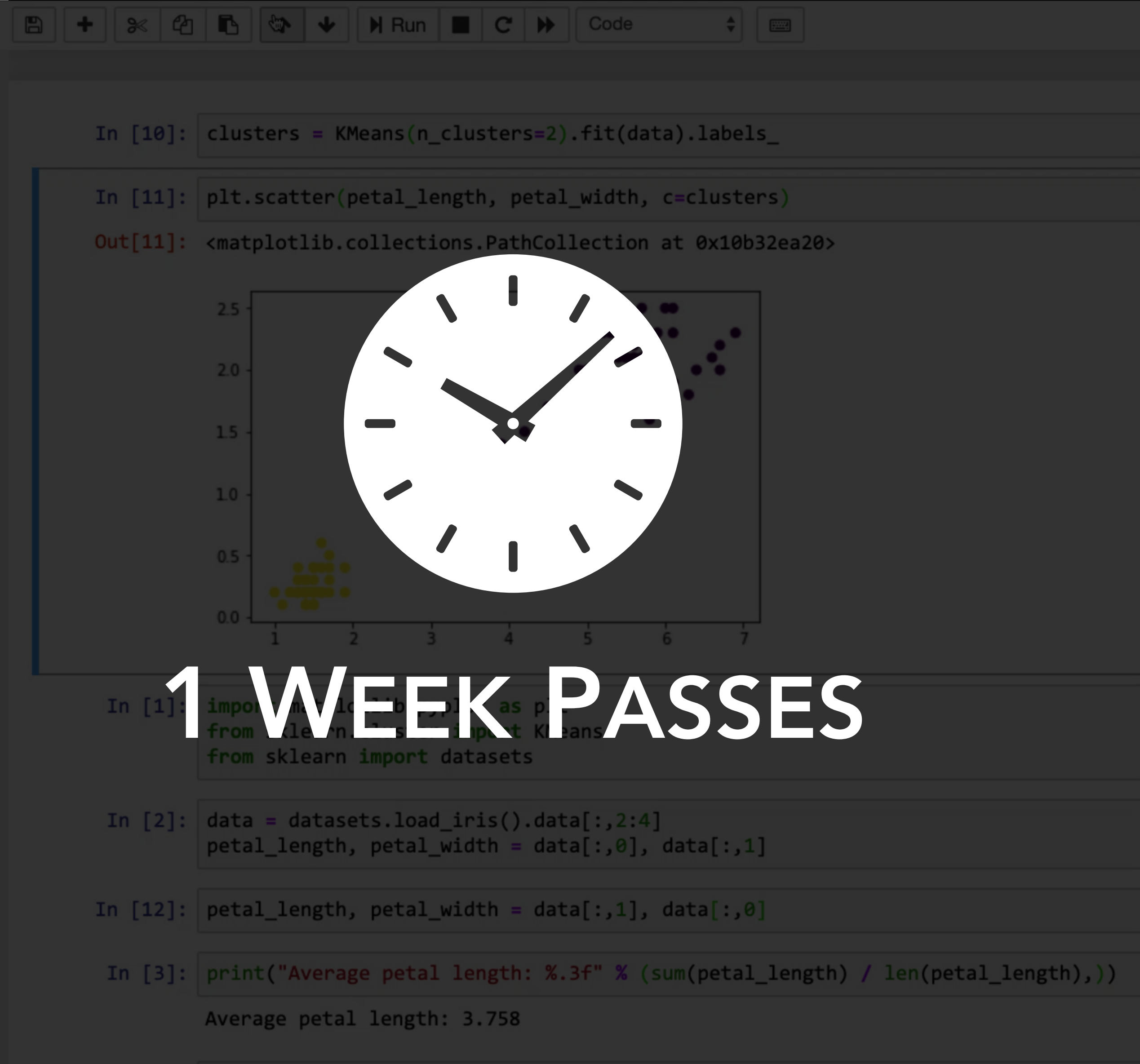
```
Out[11]: <matplotlib.collections.PathCollection at 0x10b32ea20>
```



1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout



# Notebook Model of Exploratory Programming



1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

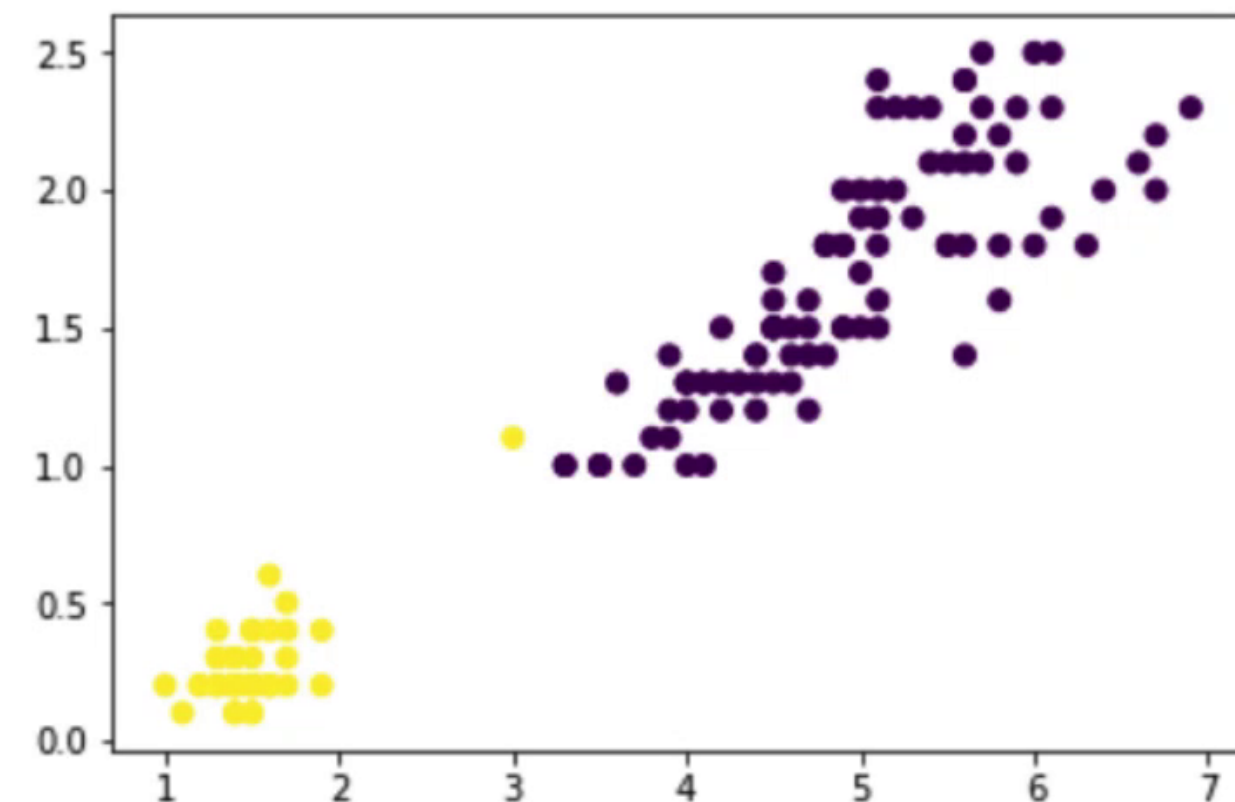
# Notebook Model of Exploratory Programming



```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x10b32ea20>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

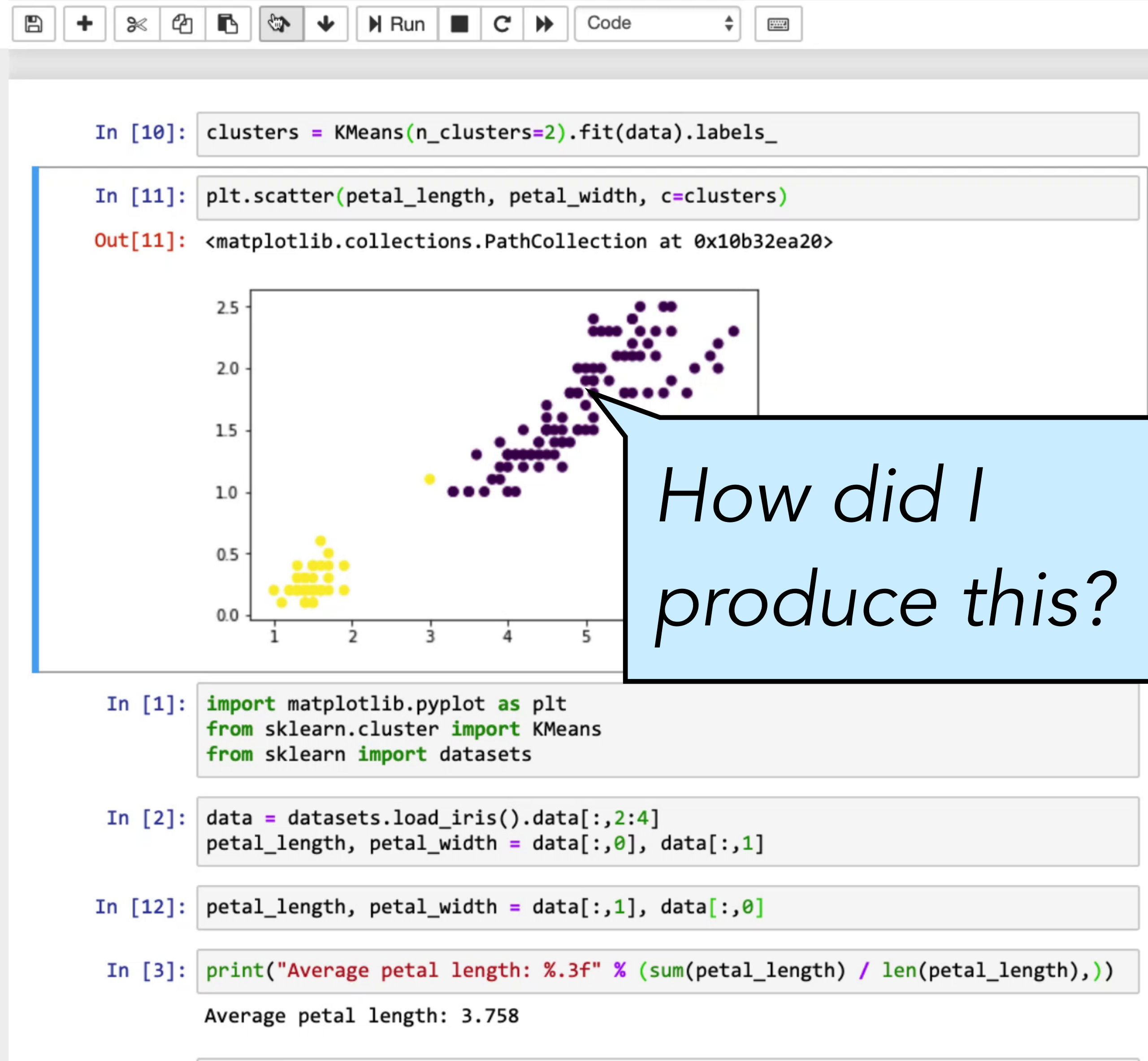
```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
Average petal length: 3.758
```

1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

# Notebook Model of Exploratory Programming



1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

## 1 WEEK LATER

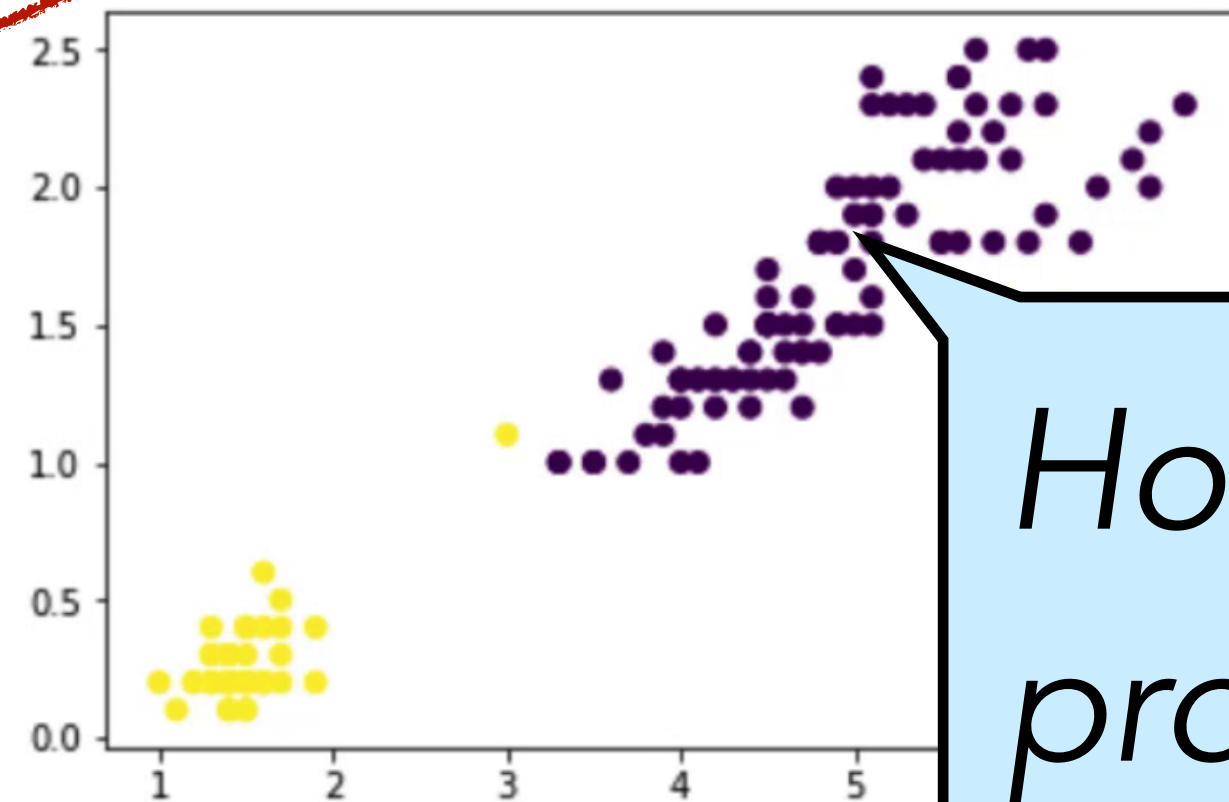
1. How did I produce this result?



# Notebook Model of Exploratory Programming



```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_  
  
In [11]: plt.scatter(petal_length, petal_width, c=clusters)  
Out[11]: <matplotlib.collections.PathCollection at 0x10b32ea20>
```



which  
petal\_length?

*How did I  
produce this?*

```
In [1]: import matplotlib.pyplot as plt  
        from sklearn.cluster import KMeans  
        from sklearn import datasets  
  
In [2]: data = datasets.load_iris().data[:,2:4]  
        petal_length, petal_width = data[:,0], data[:,1]  
  
In [12]: petal_length, petal_width = data[:,1], data[:,0]  
  
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))  
Average petal length: 3.758
```

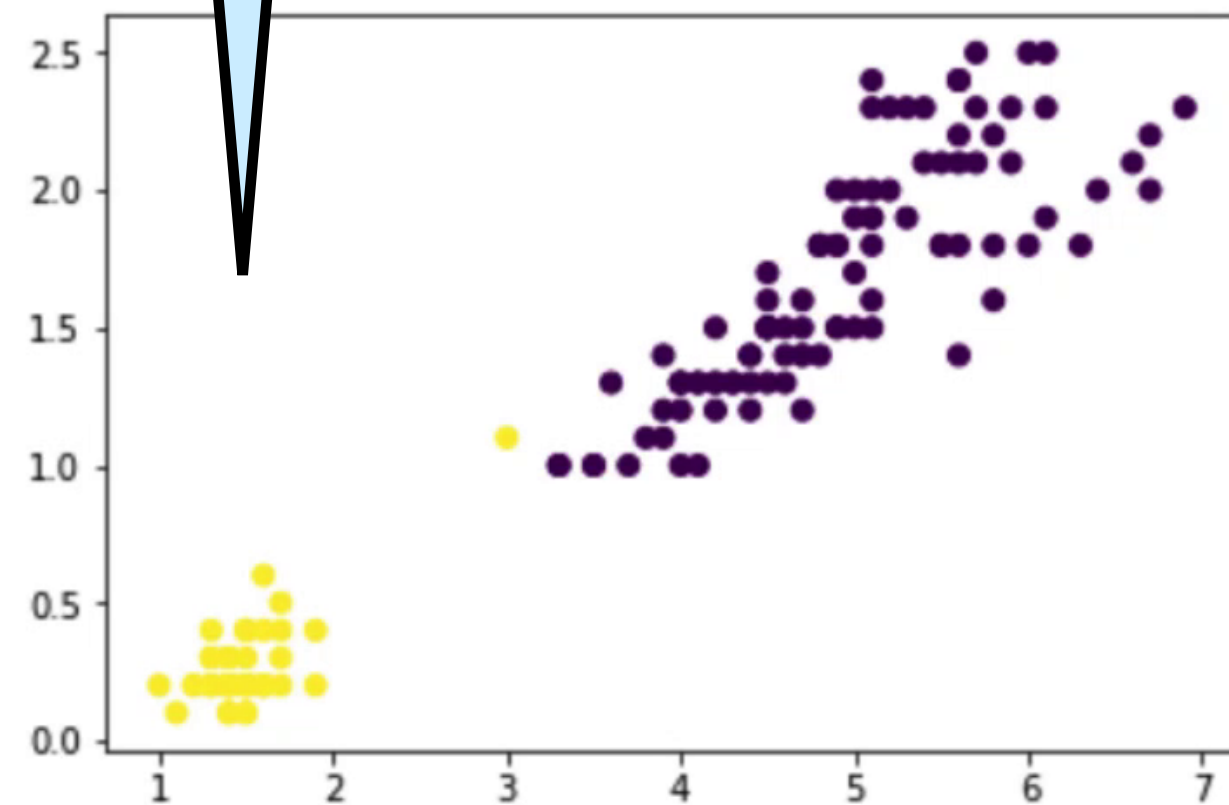
1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

## 1 WEEK LATER

1. How did I produce this result?

# Notebook Model of Exploratory Programming

*Didn't I have a better version of this?*



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

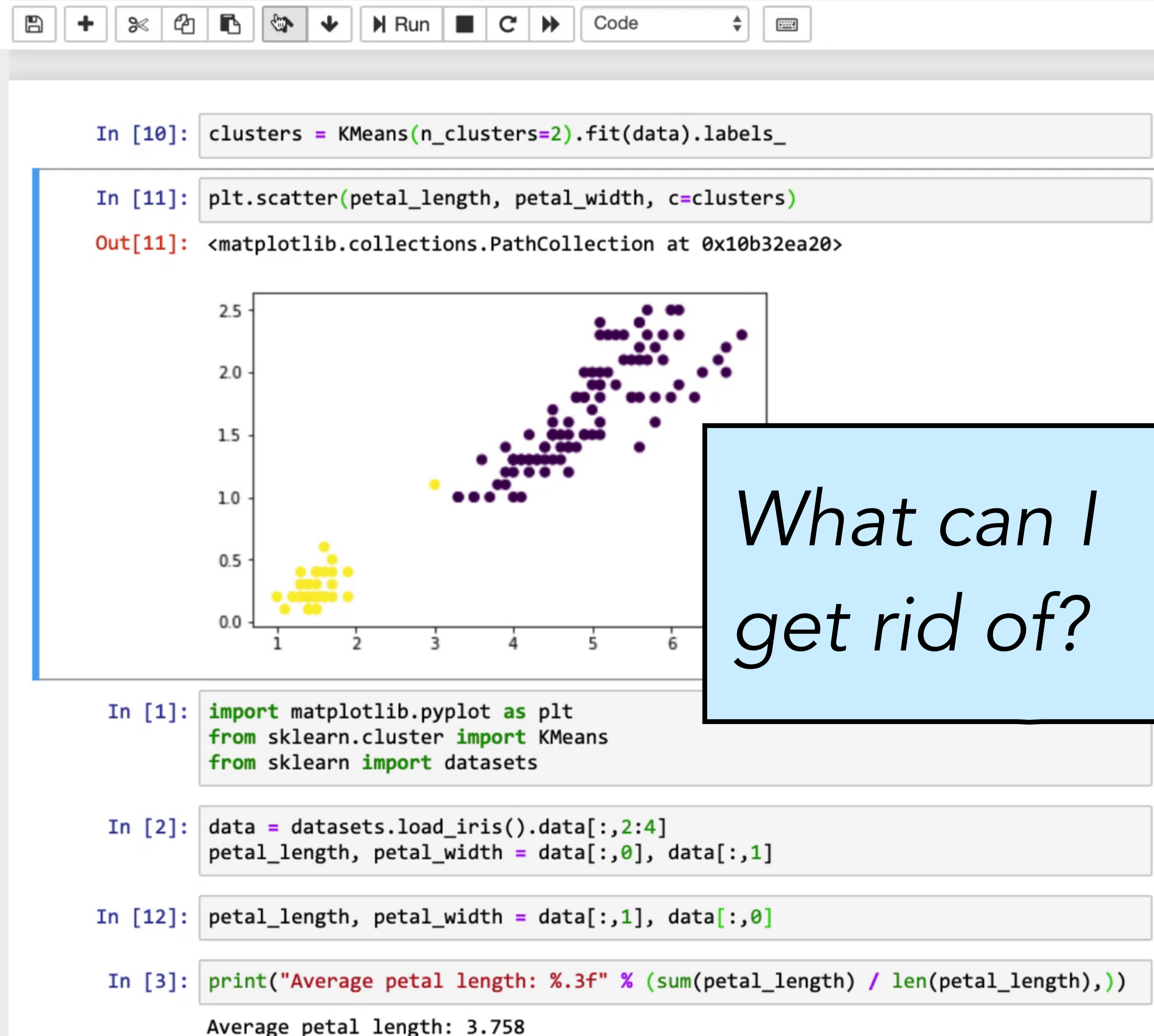
```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
Average petal length: 3.758
```

1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

## 1 WEEK LATER

1. How did I produce this result?
2. Didn't I have a better version of this?

# Notebook Model of Exploratory Programming



1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

## 1 WEEK LATER

1. How did I produce this result?
2. Didn't I have a better version of this?
3. What can I get rid of?



# Messes in Computational Notebooks

## Disappearance

Deleted / overwritten code

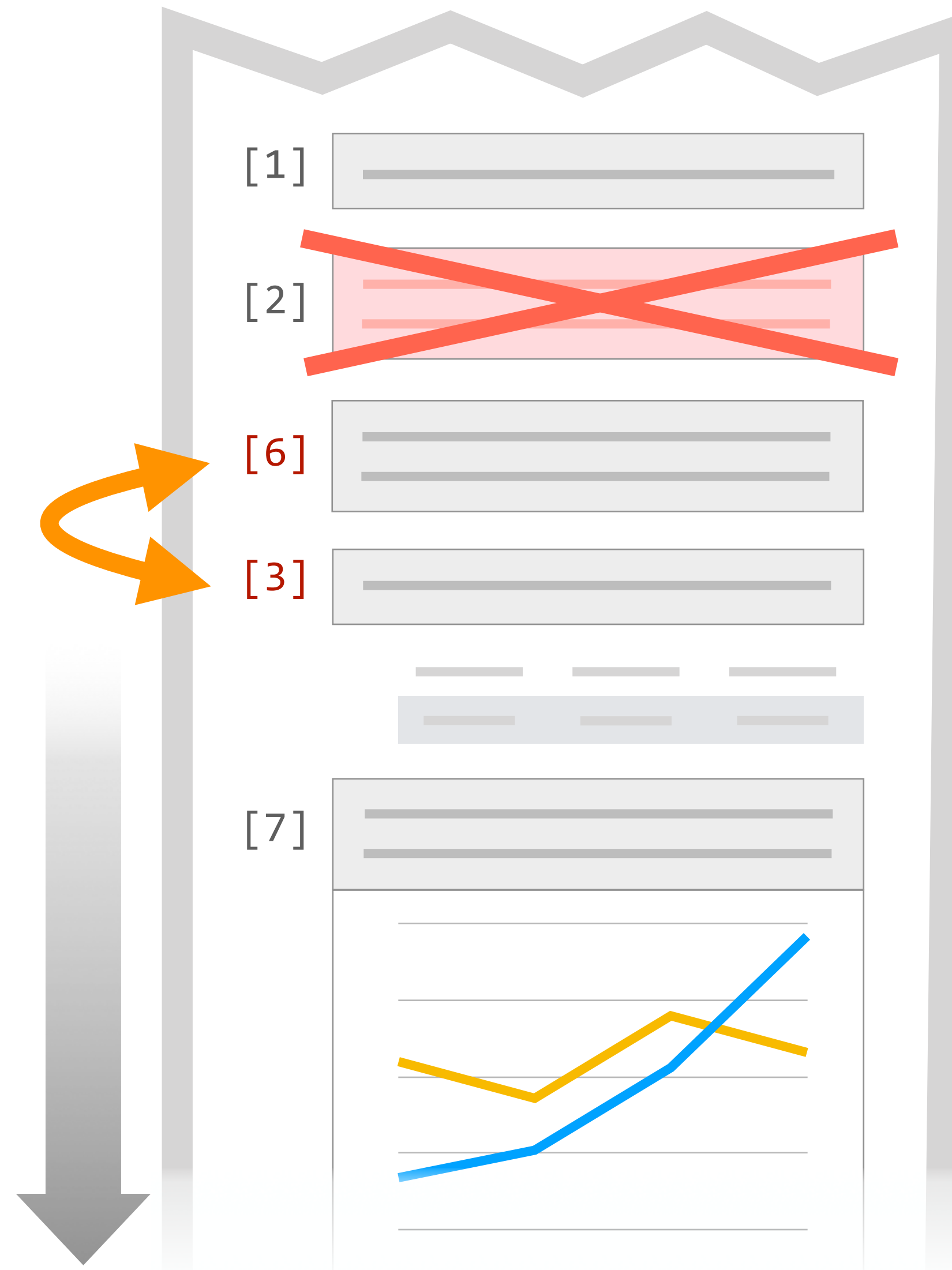
## Disorder

Out-of-order execution

1/2 of notebooks on  
GitHub [Rule et al. 2018]

## Dispersion

Too many cells



Notebooks contain  
*ugly code* and *dirty tricks* [Rule et al. 2018]

31 / 41 surveyed  
participants had trouble  
finding prior analyses  
[Kery et al. 2018]

# Managing Messes in Computational Notebooks

How can tools help analysts find, recover, and compare code in messy notebooks?

[ 1 ]

How messes happen

[ \* ]

CODE GATHERING TOOLS

[ ]

Tools in context





[ ]

Implementation

[ ]

Qualitative usability study

# CODE GATHERING TOOLS Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



## 1 WEEK PASSES

```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```


```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```



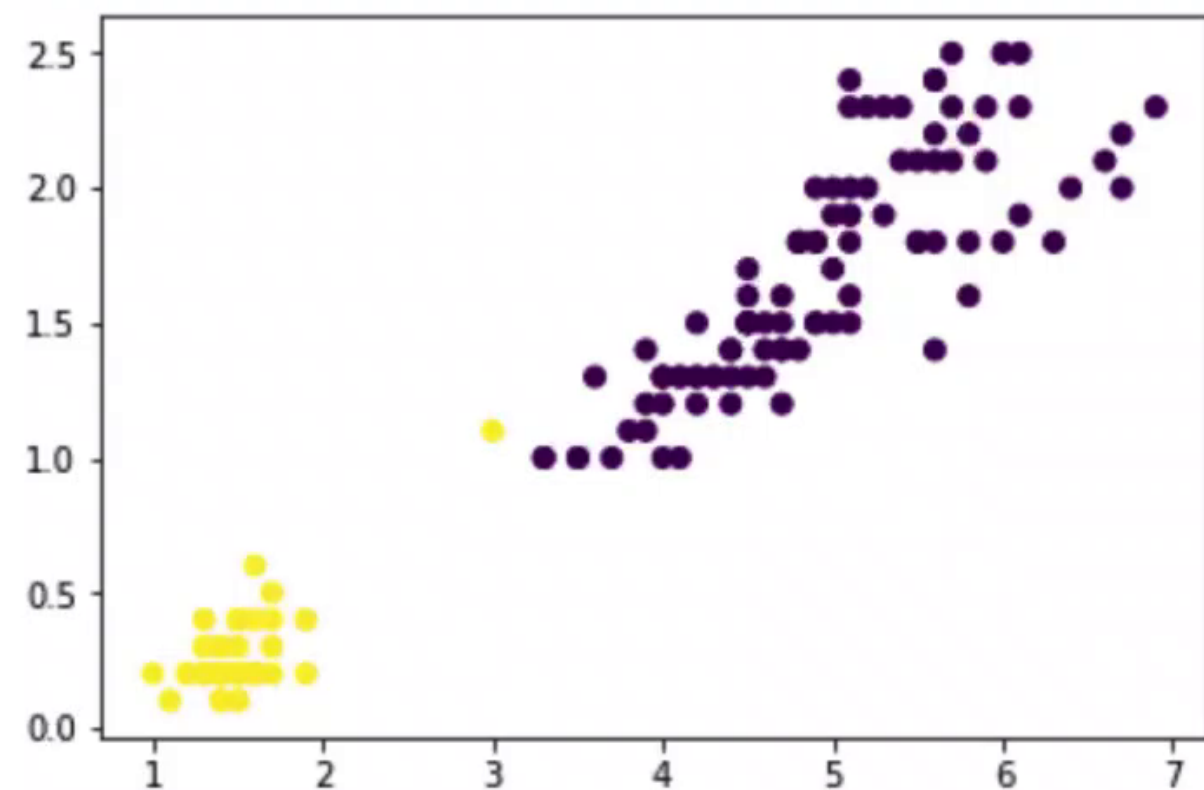
# CODE GATHERING TOOLS Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```





```
Average petal length: 3.758
```

## Task 1: Recovering Code



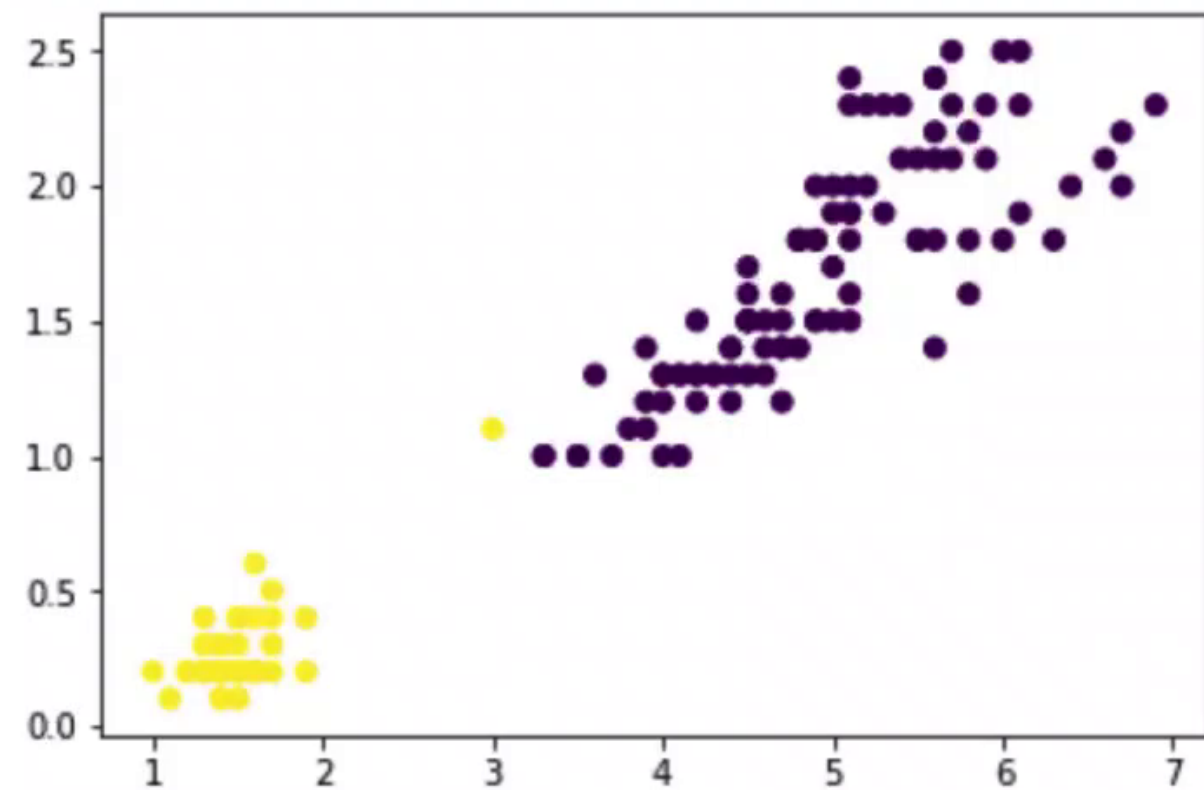
*How did I  
produce this?*

# CODE GATHERING TOOLS Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

Variables

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_  
plt.scatter(petal_length, petal_width, c=clusters)  
  
<matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]  
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length)/len(petal_length),))
```

```
Average petal length: 3.758
```

Outputs

## Task 1: Recovering Code



*How did I  
produce this?*

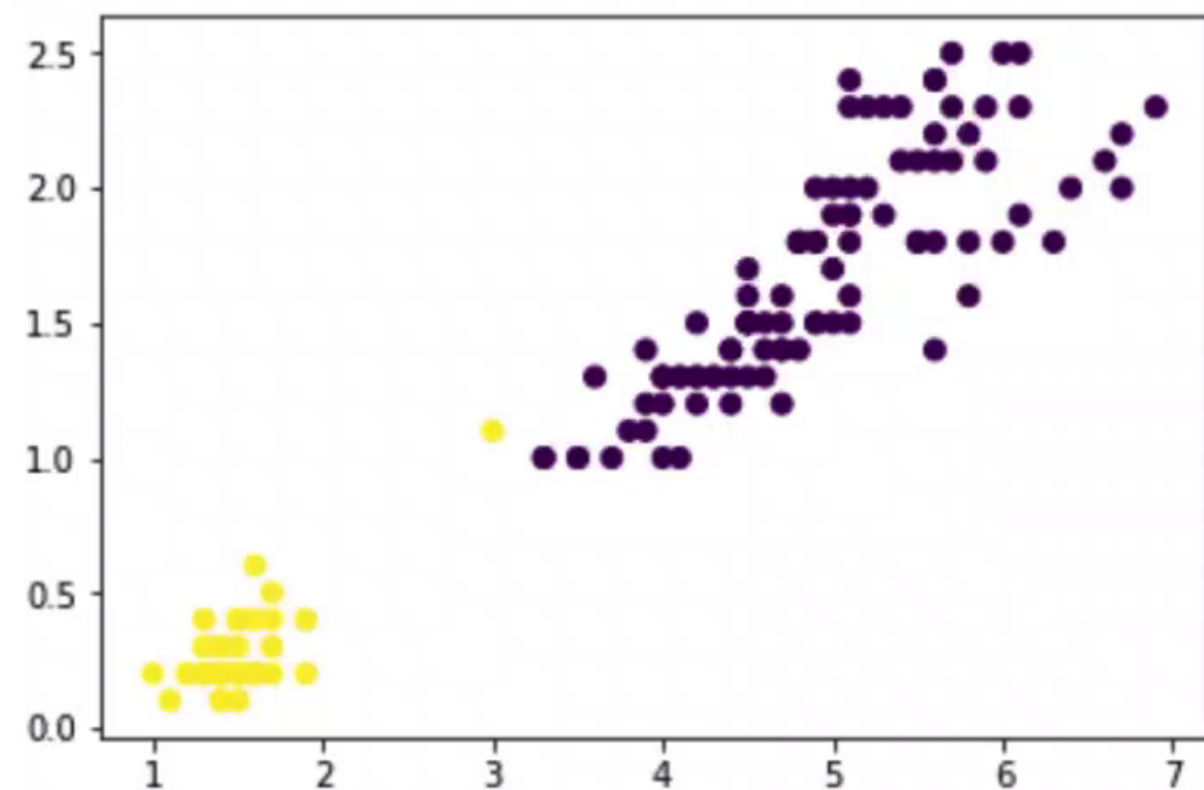
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```





## Task 1: Recovering Code



*How did I  
produce this?*



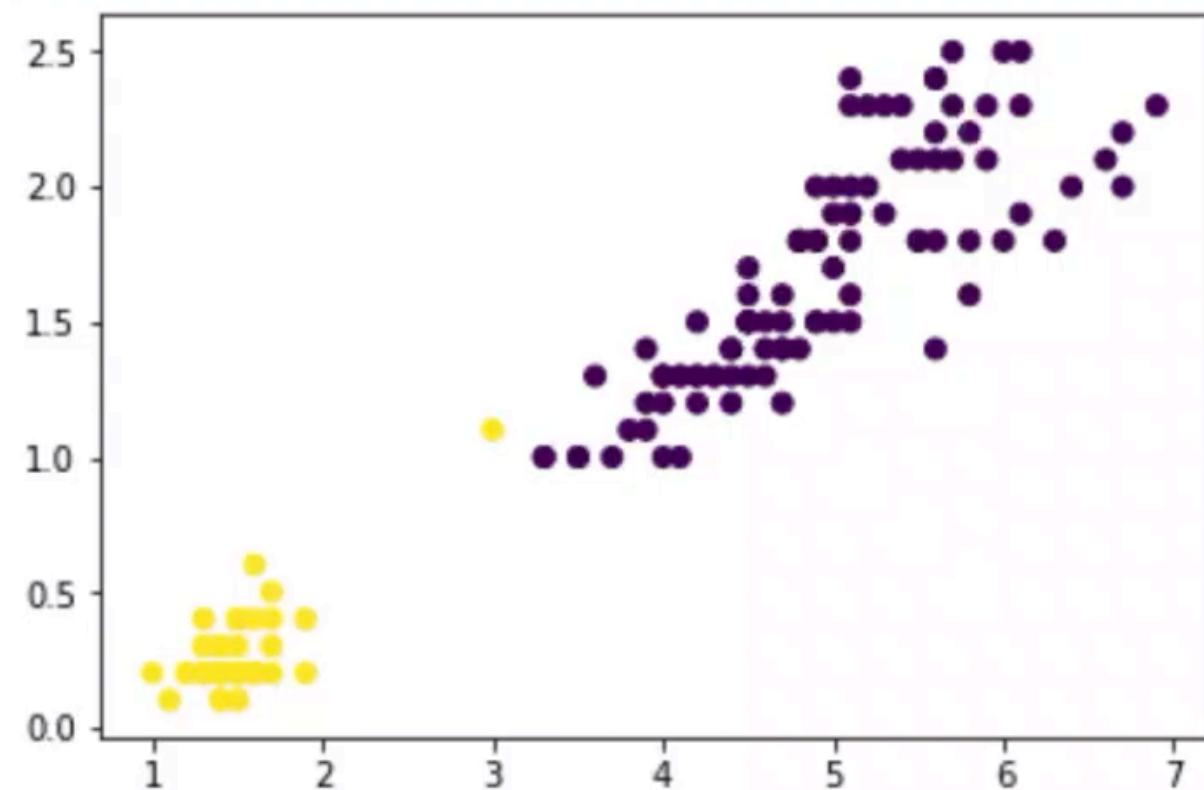
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```

## Task 1: Recovering Code



*How did I  
produce this?*

*Request cell subset that produced the result.*



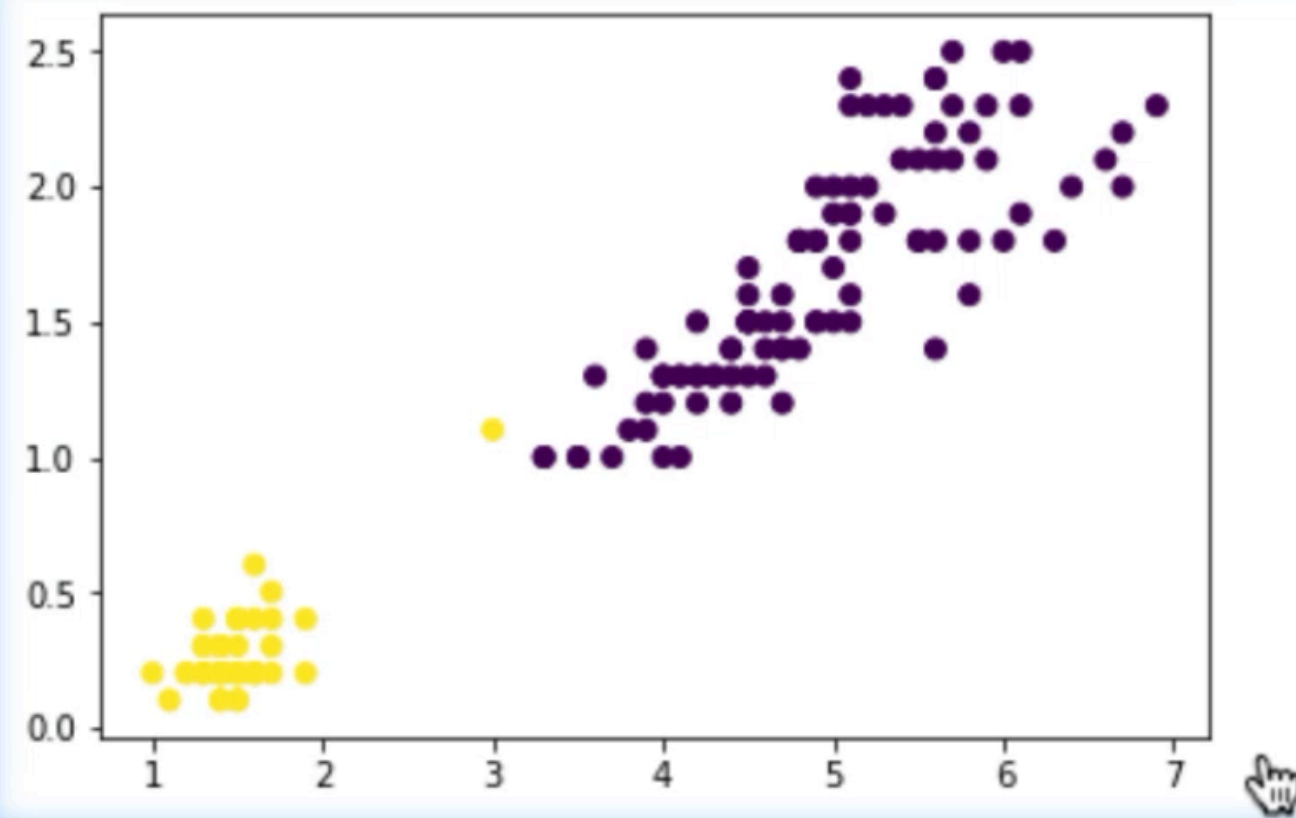
# CODE GATHERING Tools Demo

Gather to: [Clipboard](#) [Notebook](#) [Revisions](#) [Clear](#)

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8> ★
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```

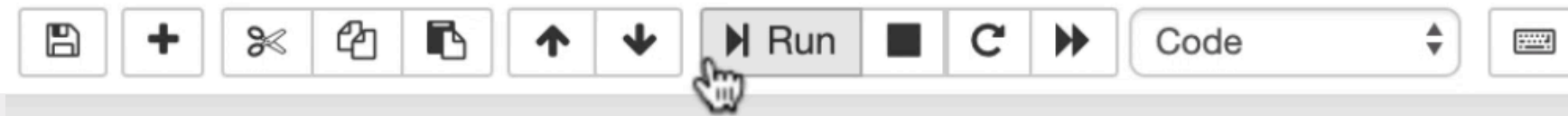
## Task 1: Recovering Code



*How did I  
produce this?*

*Request cell subset that produced the result.*

# CODE GATHERING Tools Demo

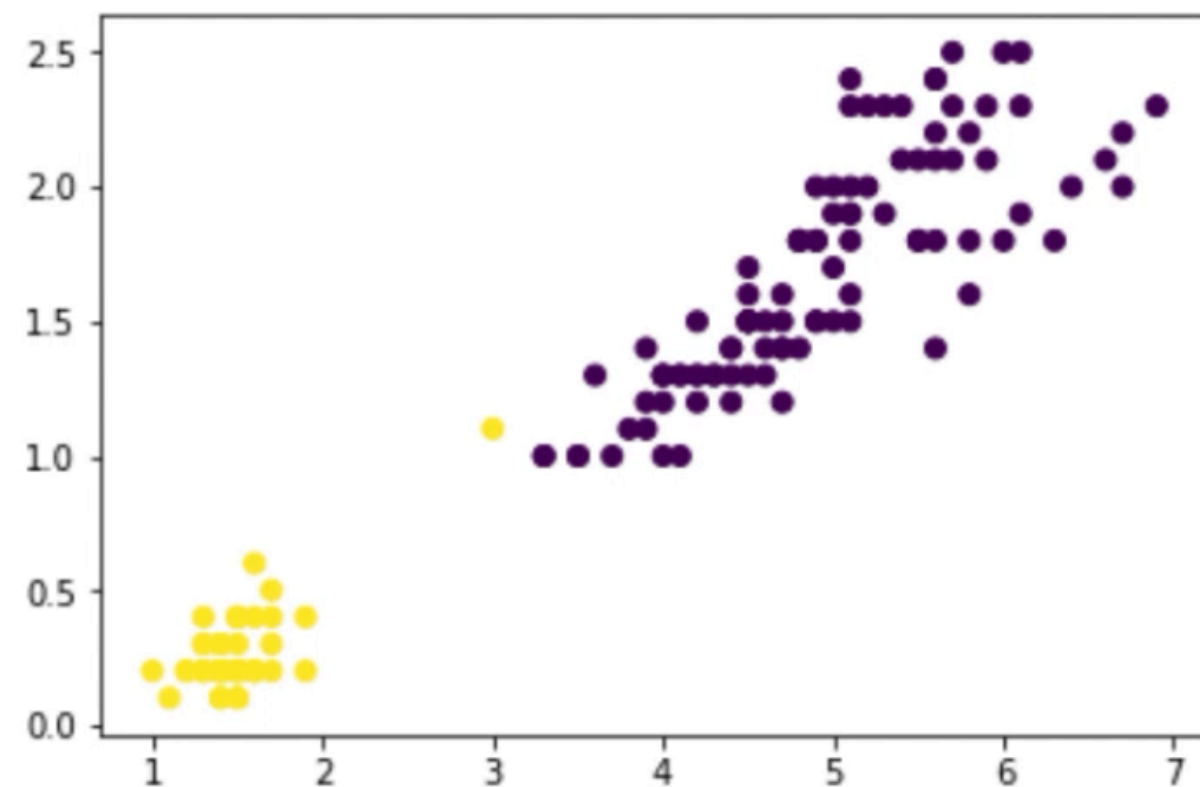


```
In [ ]: import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import datasets
```

```
In [ ]: data = datasets.load_iris().data[:,2:4]  
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [ ]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [ ]: plt.scatter(petal_length, petal_width, c=clusters)
```



The gathered code is...

- reduced
- ordered
- complete





## Task 1: Recovering Code



*How did I  
produce this?*

*Request cell subset that produced the result.*

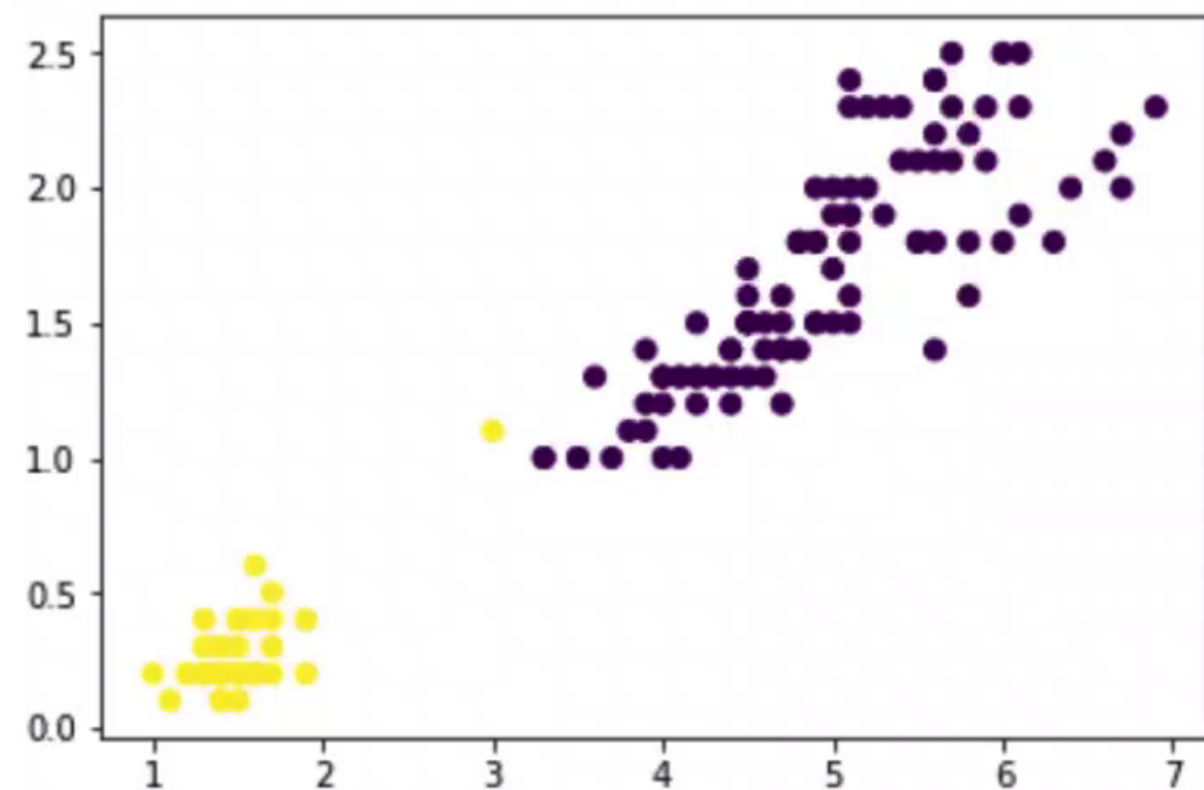
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```

## Task 1: Recovering Code





*Request cell subset that produced the result.*

## Task 2: Comparing Versions

*Didn't I have a better version of this?*



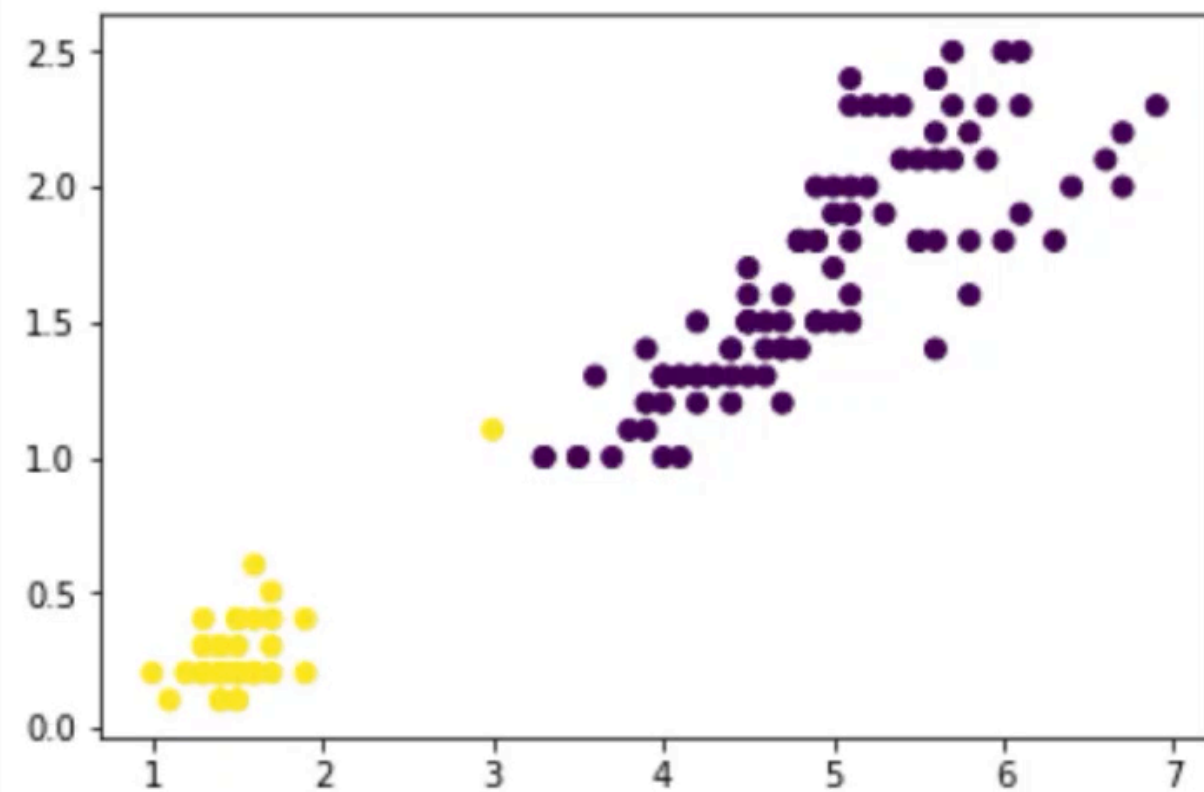
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```

## Task 1: Recovering Code

*Request cell subset that produced the result.*

## Task 2: Comparing Versions

*Didn't I have a better version of this?*

*Open a version browser for a result.*



# CODE GATHERING TOOLS Demo

Gather to: 

Clipboard

Notebook

Revisions

Clear

Current version

Open in notebook

Copy cells

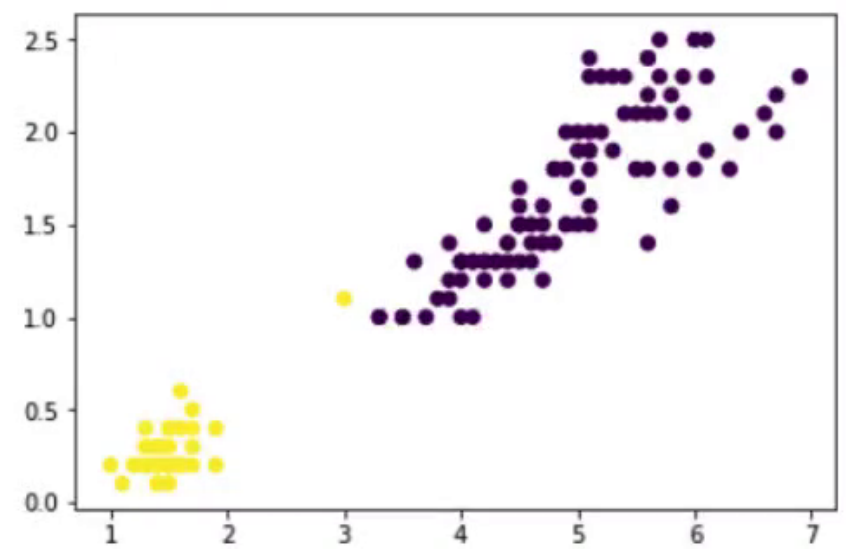
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi

plt.scatter(petal_length, petal_wi

<matplotlib.collections.PathCollec
tion at 0x108eacba8>
```



1 week ago

Open in notebook

Copy cells

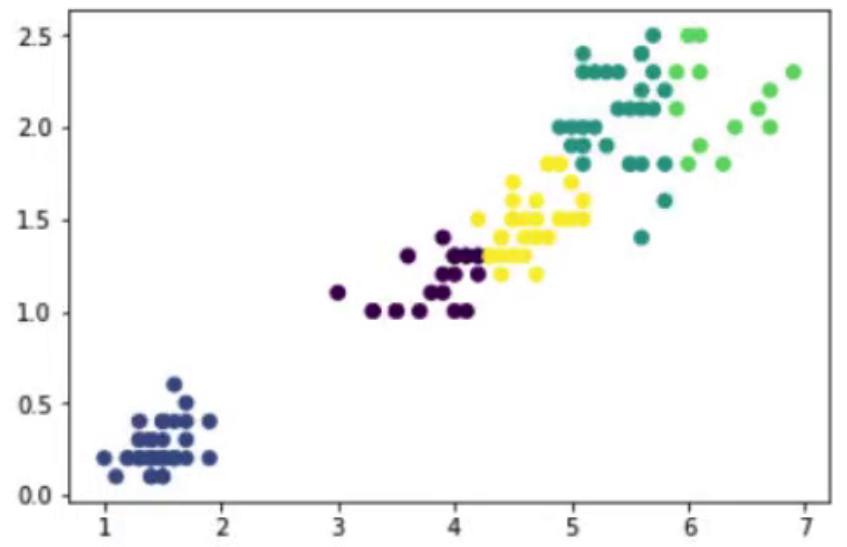
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi
clusters = KMeans(n_clusters=5).fi

plt.scatter(petal_length, petal_wi

<matplotlib.collections.PathCollec
tion at 0x108f4a9e8>
```



## Task 1: Recovering Code



*Request cell subset that produced the result.*

## Task 2: Comparing Versions



*Didn't I have a better version of this?*

*Open a version browser for a result.*

# CODE GATHERING TOOLS Demo

Gather to: Clipboard Notebook Revisions Clear

Current version

Open in notebook Copy cells

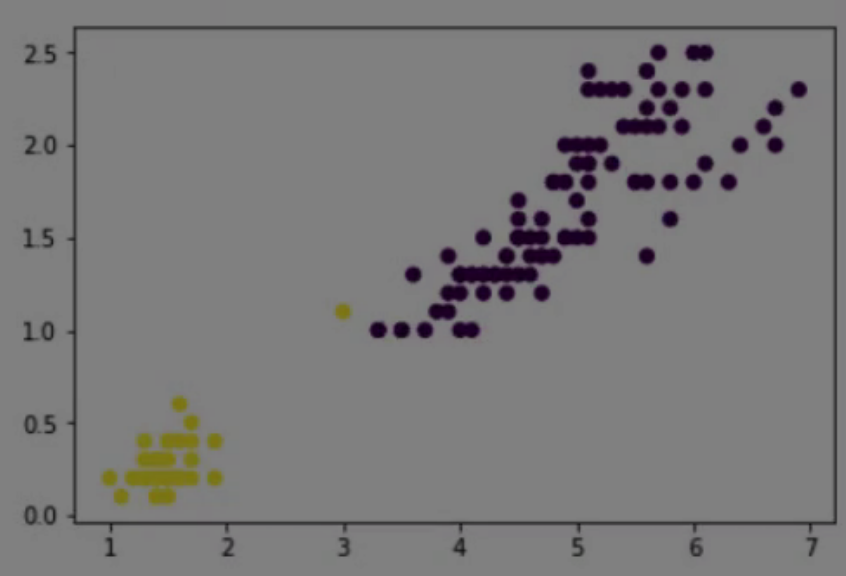
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi

plt.scatter(petal_length, petal_wi
```

<matplotlib.collections.PathCollec  
tion at 0x108eacba8>



1 week ago

Open in notebook Copy cells

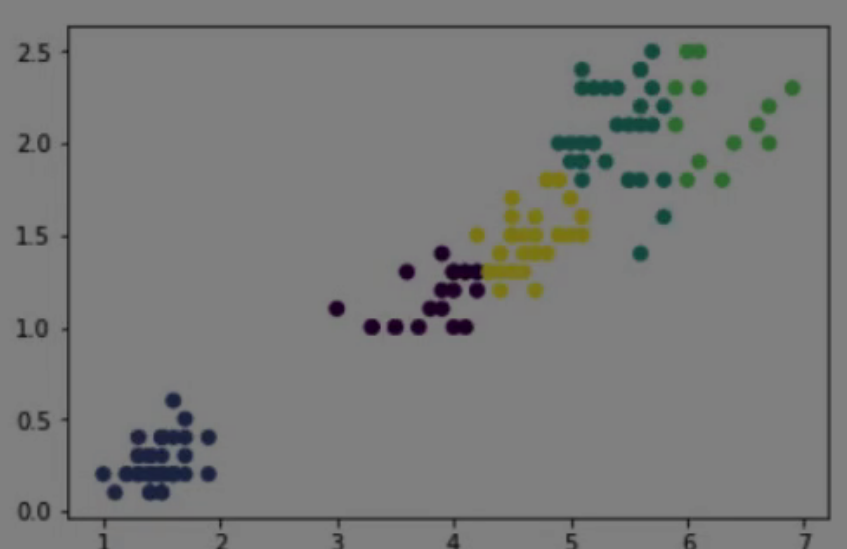
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi
clusters = KMeans(n_clusters=5).fi

plt.scatter(petal_length, petal_wi
```

<matplotlib.collections.PathCollec  
tion at 0x108f4a9e8>



## Task 1: Recovering Code



*Request cell subset that produced the result.*

## Task 2: Comparing Versions



*Didn't I have a better  
version of this?*

*Open a version browser for a result.*



# CODE GATHERING TOOLS Demo

Gather to: Clipboard Notebook Revisions Clear

### Current version

Open in notebook Copy cells

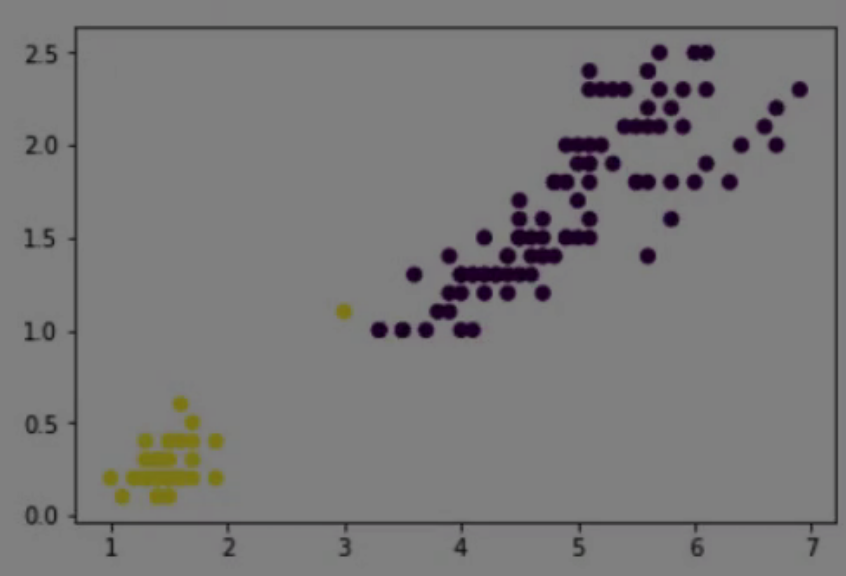
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi

plt.scatter(petal_length, petal_wi

<matplotlib.collections.PathCollec
tion at 0x108eacba8>
```



### 1 week ago

Open in notebook Copy cells

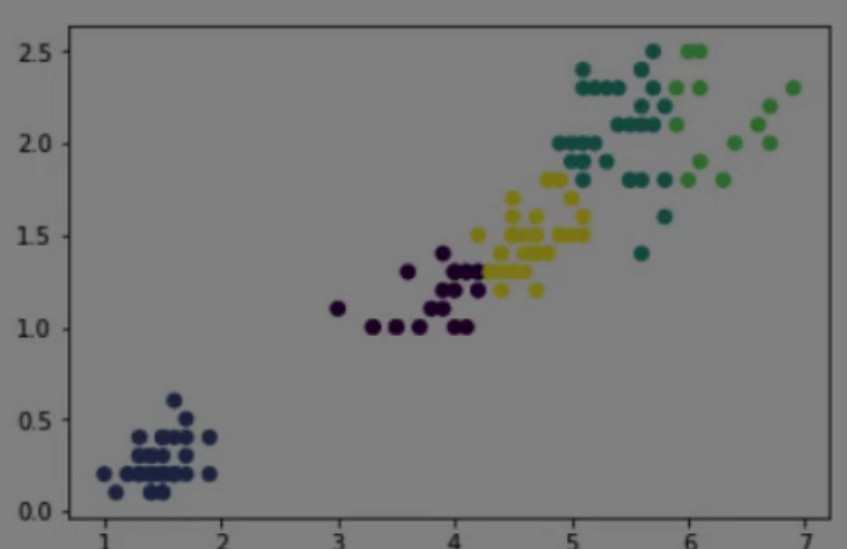
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi
clusters = KMeans(n_clusters=5).fi

plt.scatter(petal_length, petal_wi

<matplotlib.collections.PathCollec
tion at 0x108f4a9e8>
```



## Task 1: Recovering Code



*Request cell subset that produced the result.*

## Task 2: Comparing Versions



*Didn't I have a better version of this?*

*Open a version browser for a result.*

# CODE GATHERING TOOLS Demo

Gather to: Clipboard Notebook Revisions Clear

Current version

Open in notebook Copy cells

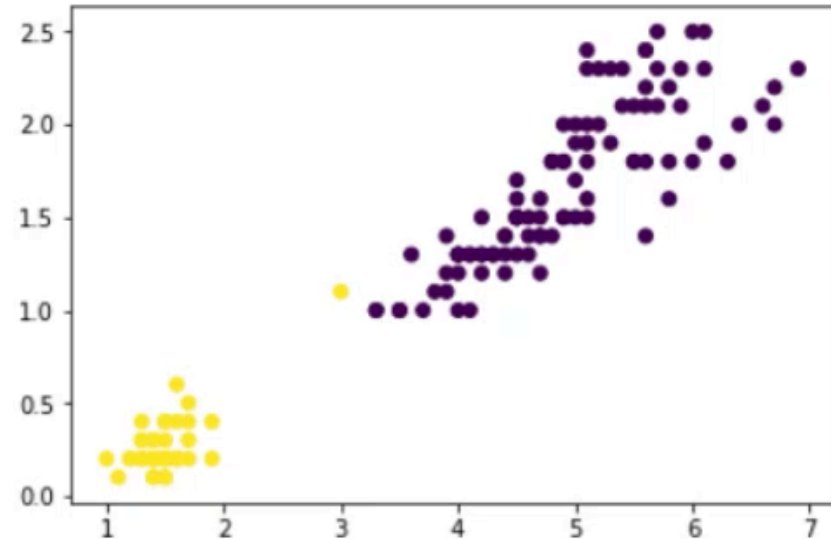
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi

plt.scatter(petal_length, petal_wi
```

```
<matplotlib.collections.PathCollec
tion at 0x108eacba8>
```



1 week ago

Open in notebook Copy cells

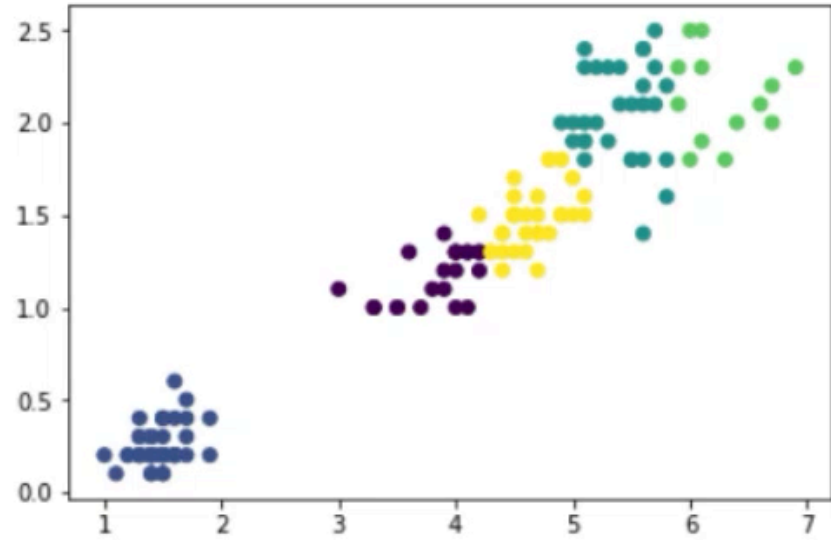
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

data = datasets.load_iris().data[:
petal_length, petal_width = data[:

clusters = KMeans(n_clusters=2).fi
clusters = KMeans(n_clusters=5).fi

plt.scatter(petal_length, petal_wi
```

```
<matplotlib.collections.PathCollec
tion at 0x108f4a9e8>
```



## Task 1: Recovering Code



*Request cell subset that produced the result.*

## Task 2: Comparing Versions




*Didn't I have a better version of this?*

*Open a version browser for a result.*



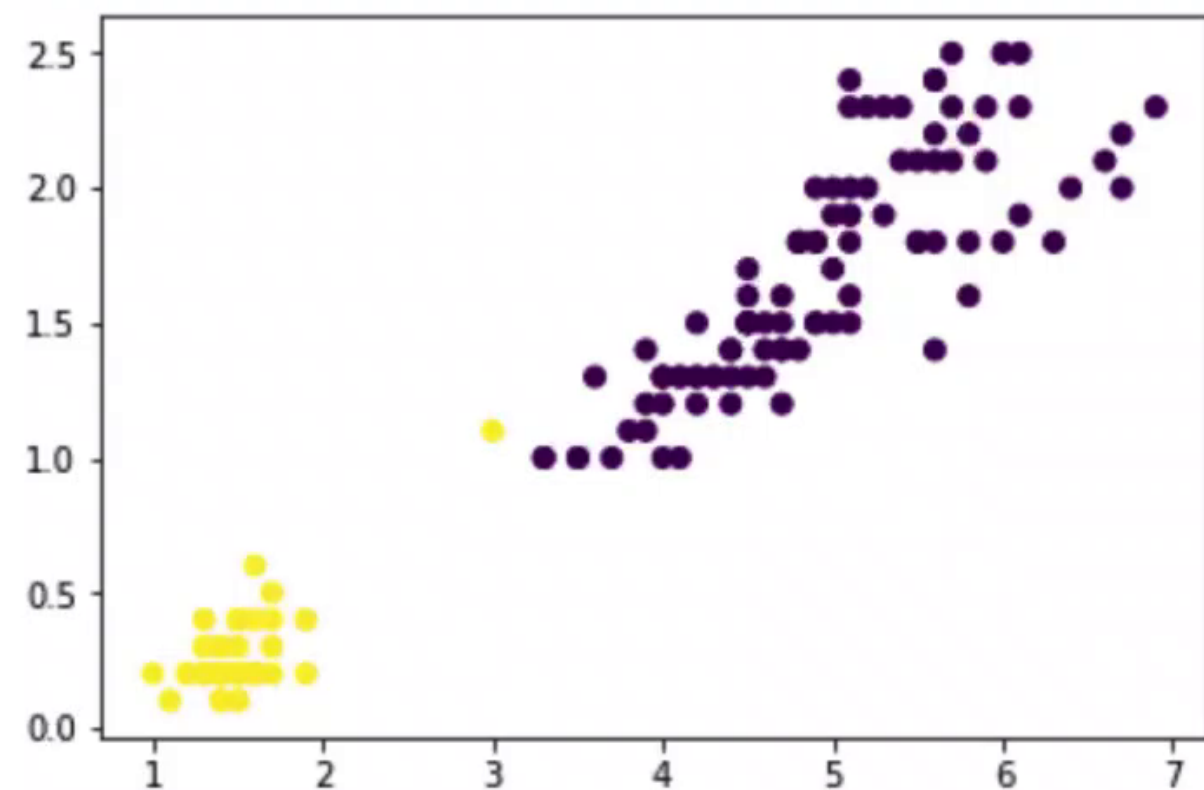
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```

## Task 1: Recovering Code

*Request cell subset that produced the result.*





## Task 2: Comparing Versions

*Open a version browser for a result.*

## Task 3: Cleaning Notebook

*What code can  
I get rid of?*

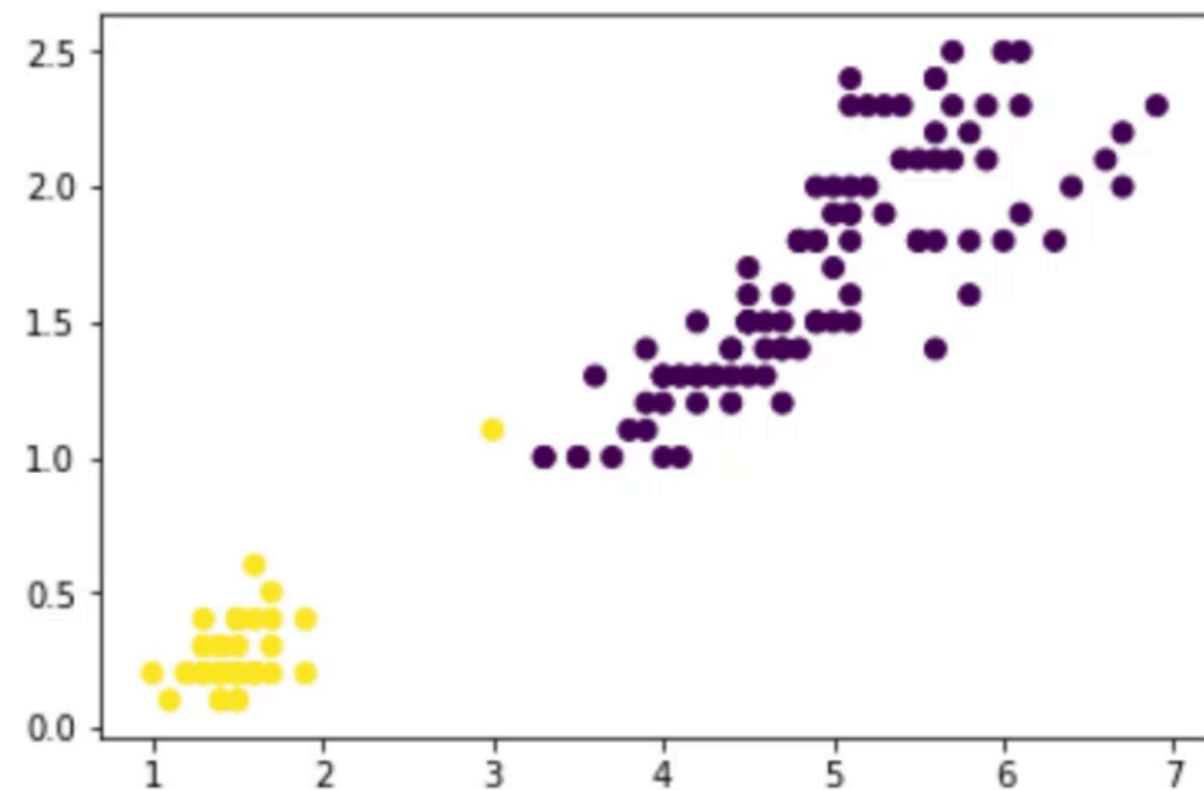
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
Average petal length: 3.758
```

## Task 1: Recovering Code

*Request cell subset that produced the result.*

## Task 2: Comparing Versions

*Open a version browser for a result.*



## Task 3: Cleaning Notebook

*What code can  
I get rid of?*

*... Request cell subset that produced the result.*



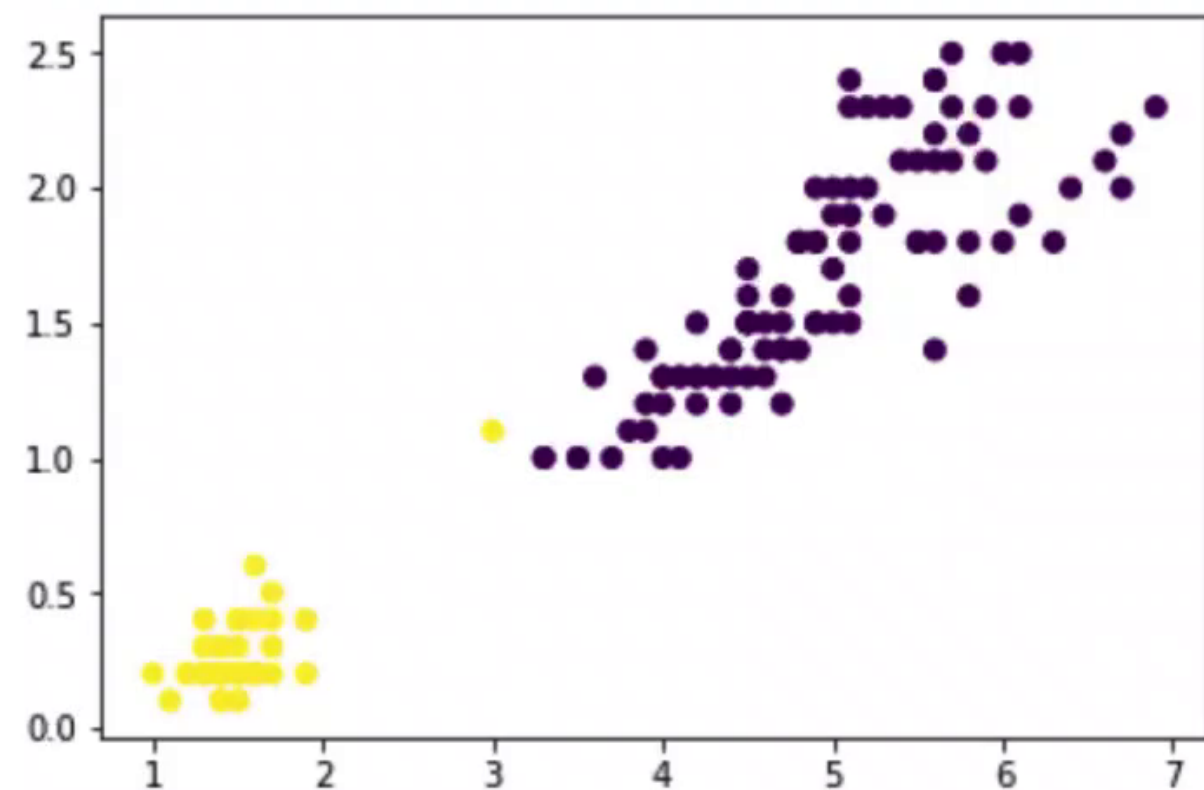
# CODE GATHERING Tools Demo

Gather to:  Clipboard  Notebook  Revisions  Clear

```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
In [11]: plt.scatter(petal_length, petal_width, c=clusters)
```

```
Out[11]: <matplotlib.collections.PathCollection at 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]
```

```
In [12]: petal_length, petal_width = data[:,1], data[:,0]
```

```
In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))
```

```
Average petal length: 3.758
```

## How can tools help analysts manage messes in their notebooks?

### Task 1: Recovering Code



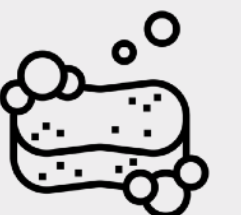
*Request cell subset that produced the result.*

### Task 2: Comparing Versions



*Open a version browser for a result.*

### Task 3: Cleaning Notebook



*... Request cell subset that produced the result.*

## **Post-Hoc Mess Management**

Helping analysts clean and navigate their code whether or not they adopted a strategy to version or organize their code.



# Managing Messes in Computational Notebooks

How can tools help analysts find, recover, and compare code in messy notebooks?

[1] How messes happen

[2] CODE GATHERING TOOLS

[3] Tools in context

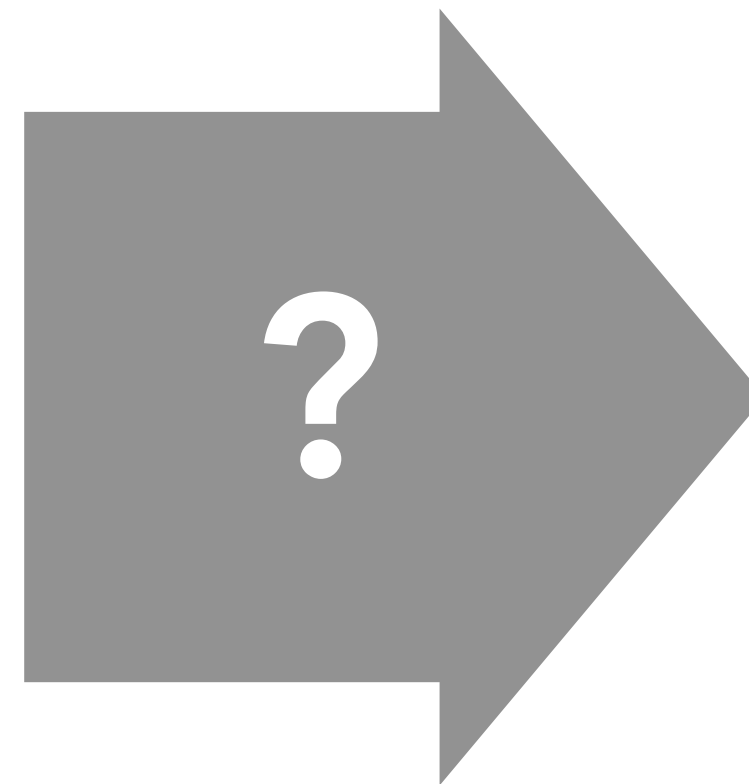
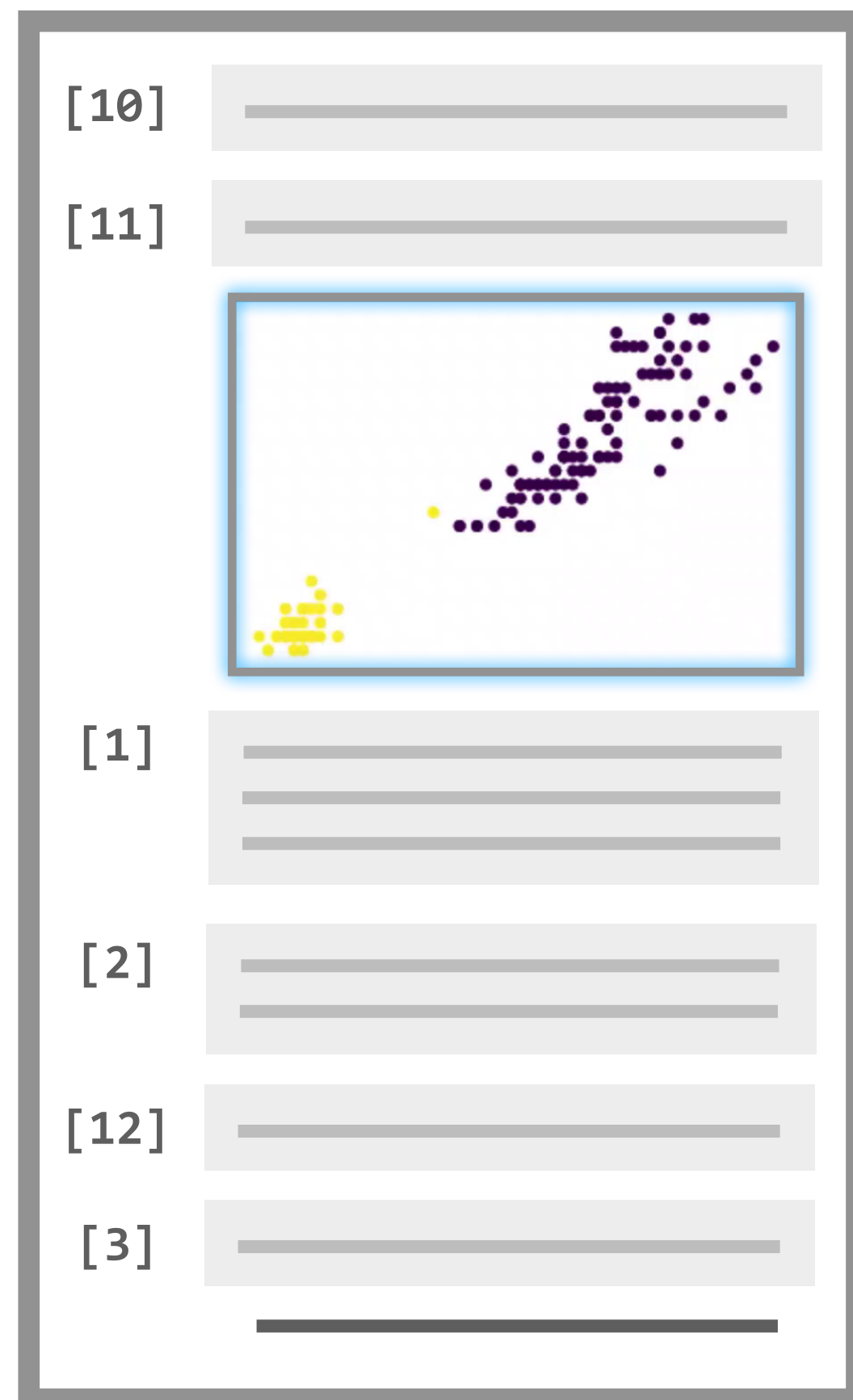
[\*] **Implementation**

[ ] Qualitative usability study

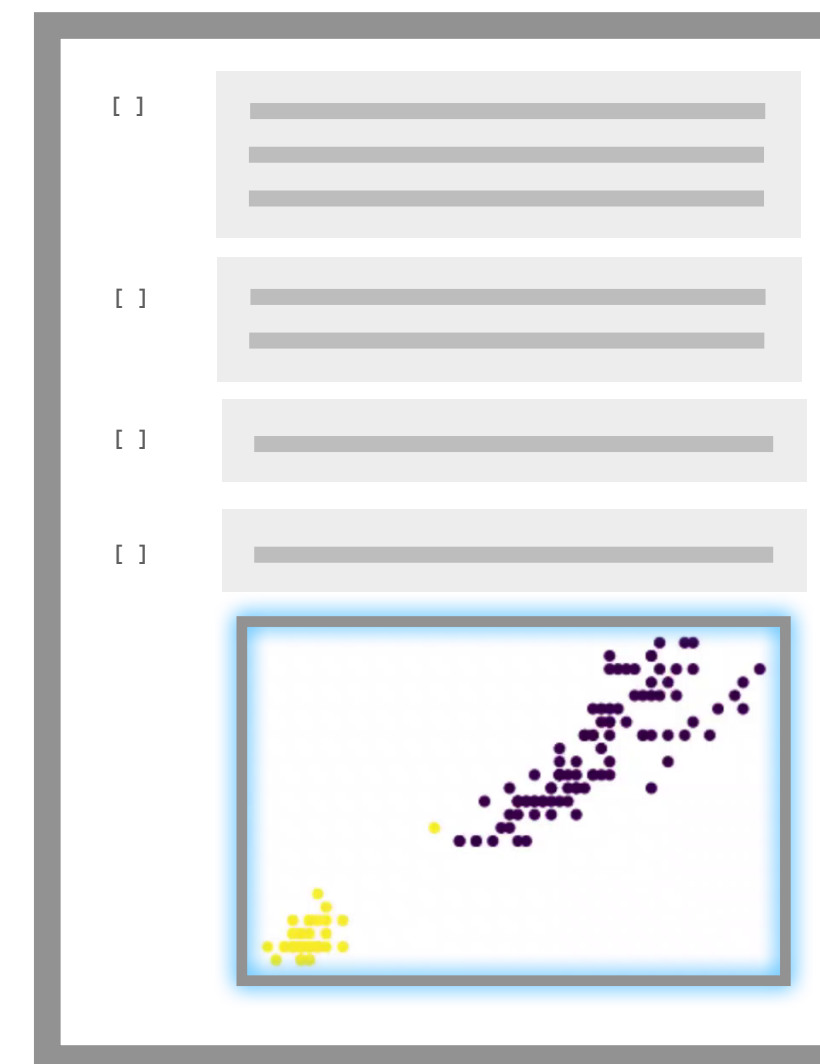
# Implementation: Slicing Notebooks

① Notebook

some cells missing,  
some cells out-of-order



cleaned, ordered  
notebooks



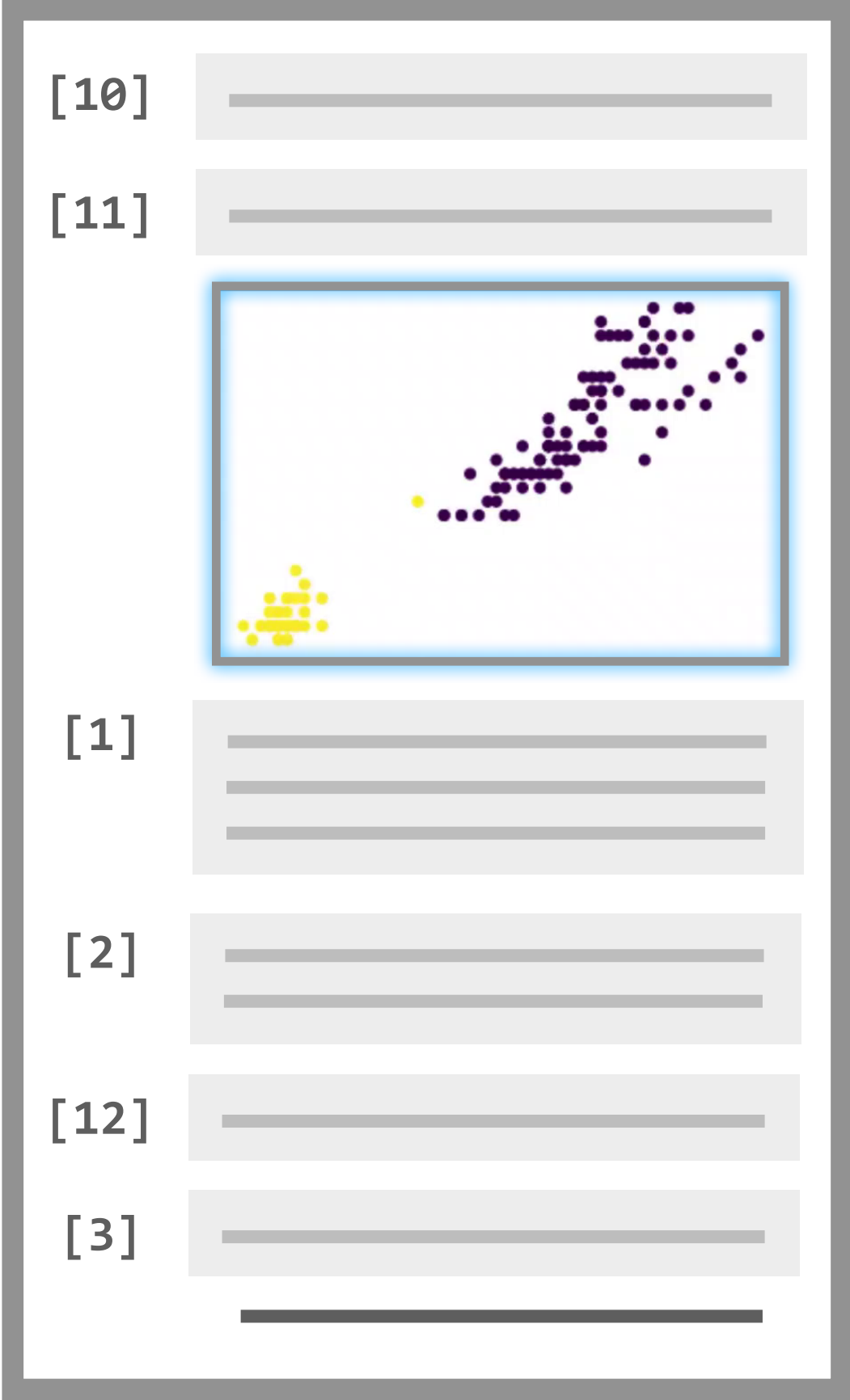
versioned results



# Implementation: Slicing Notebooks

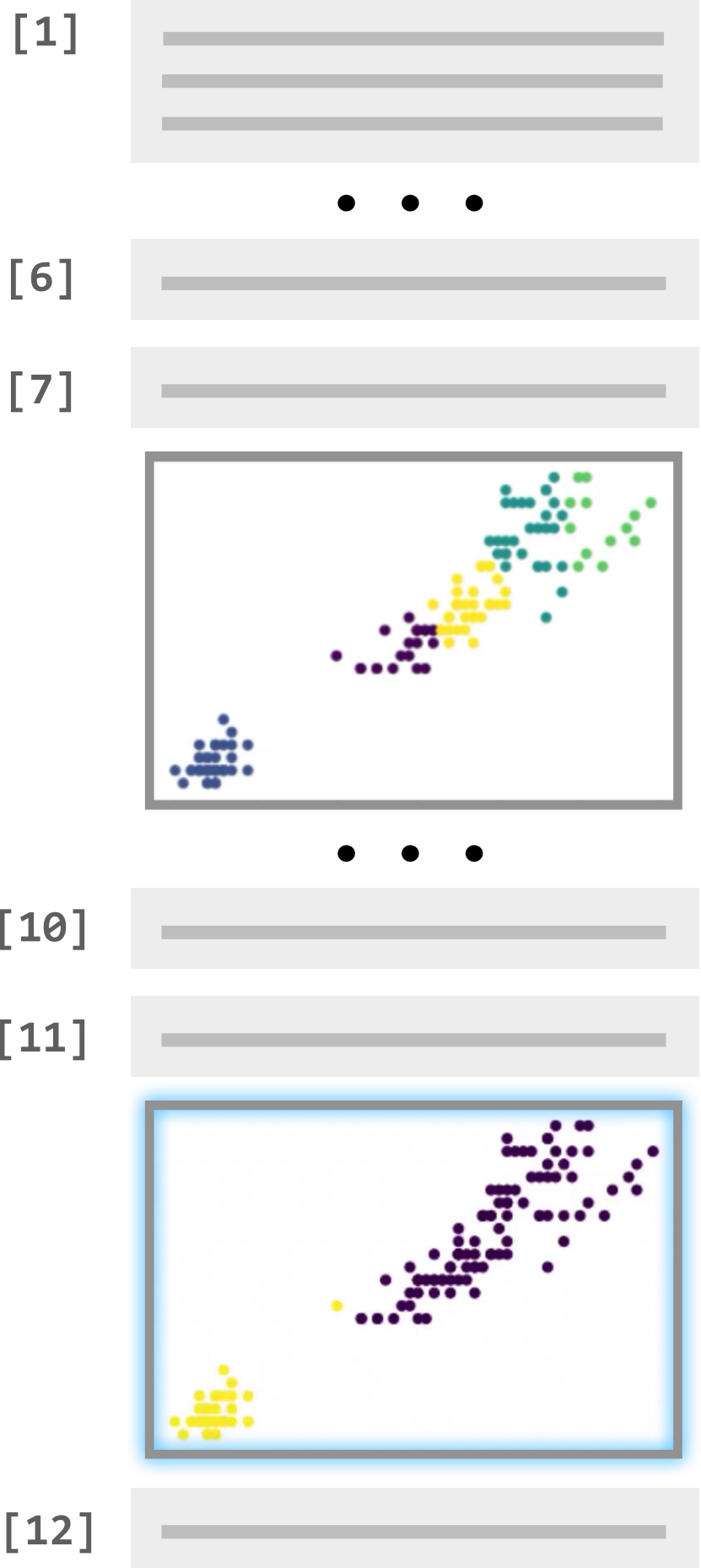
① Notebook

some cells missing,  
some cells out-of-order



② Execution Log

all cells present, in-order



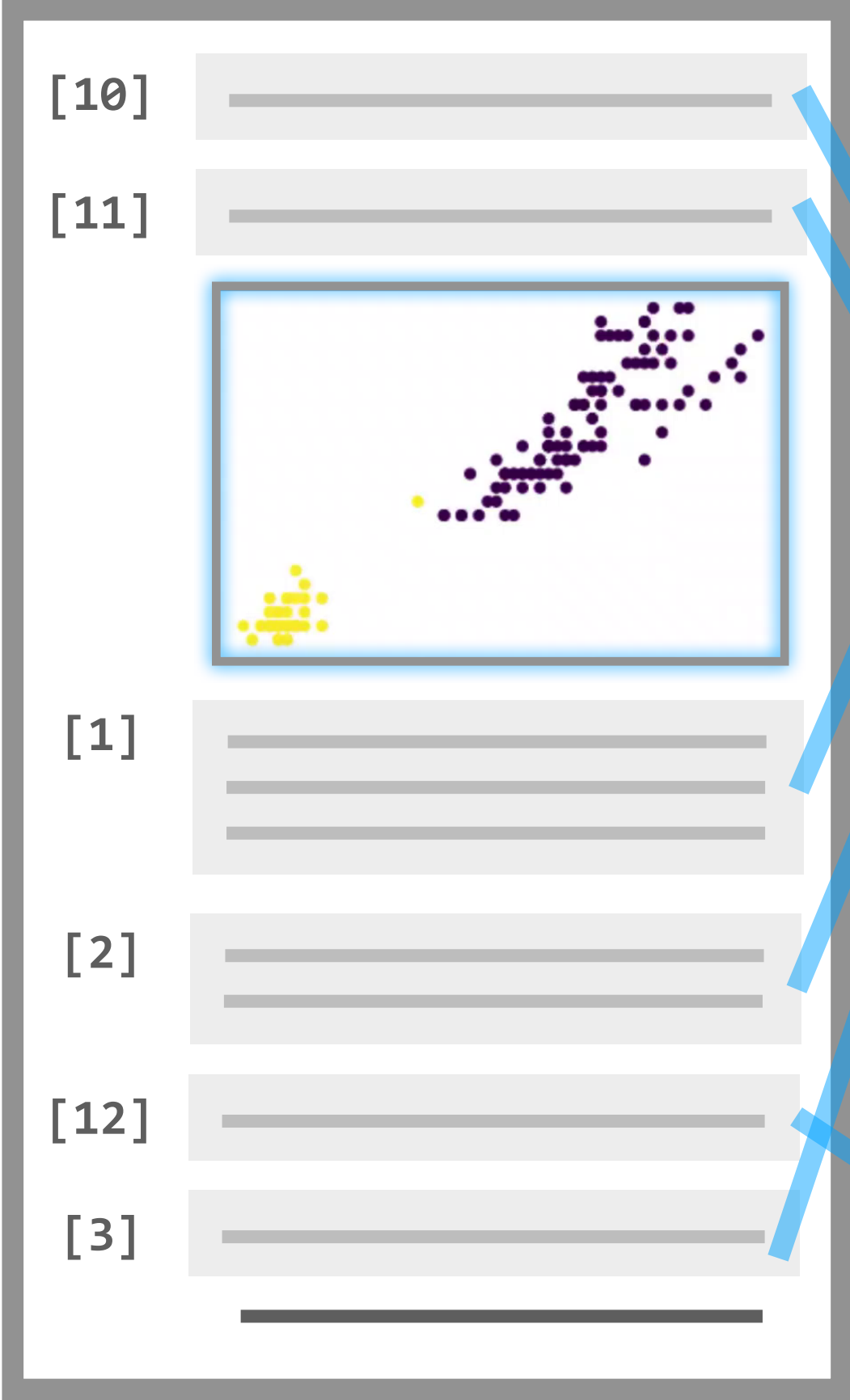
execution time



# Implementation: Slicing Notebooks

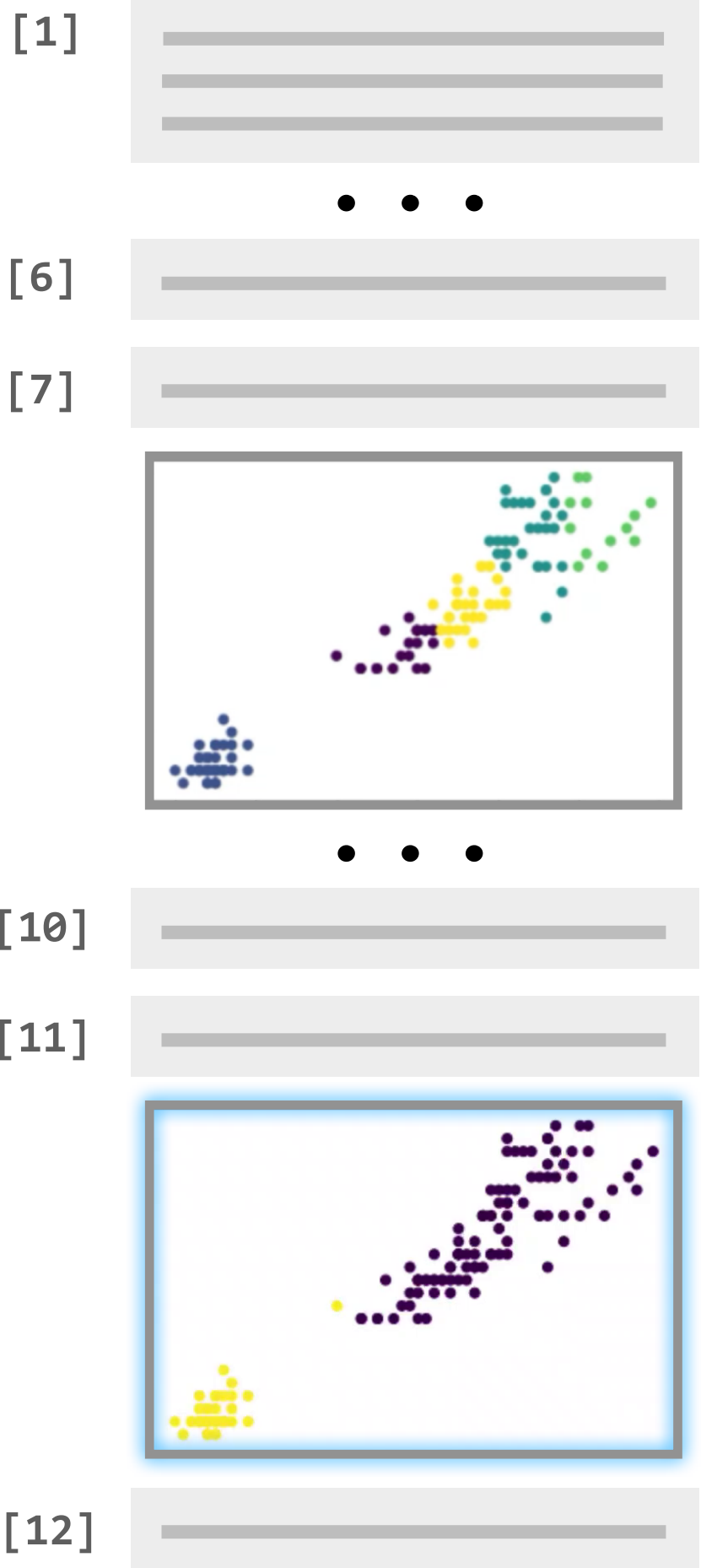
① Notebook

some cells missing,  
some cells out-of-order



② Execution Log

all cells present, in-order

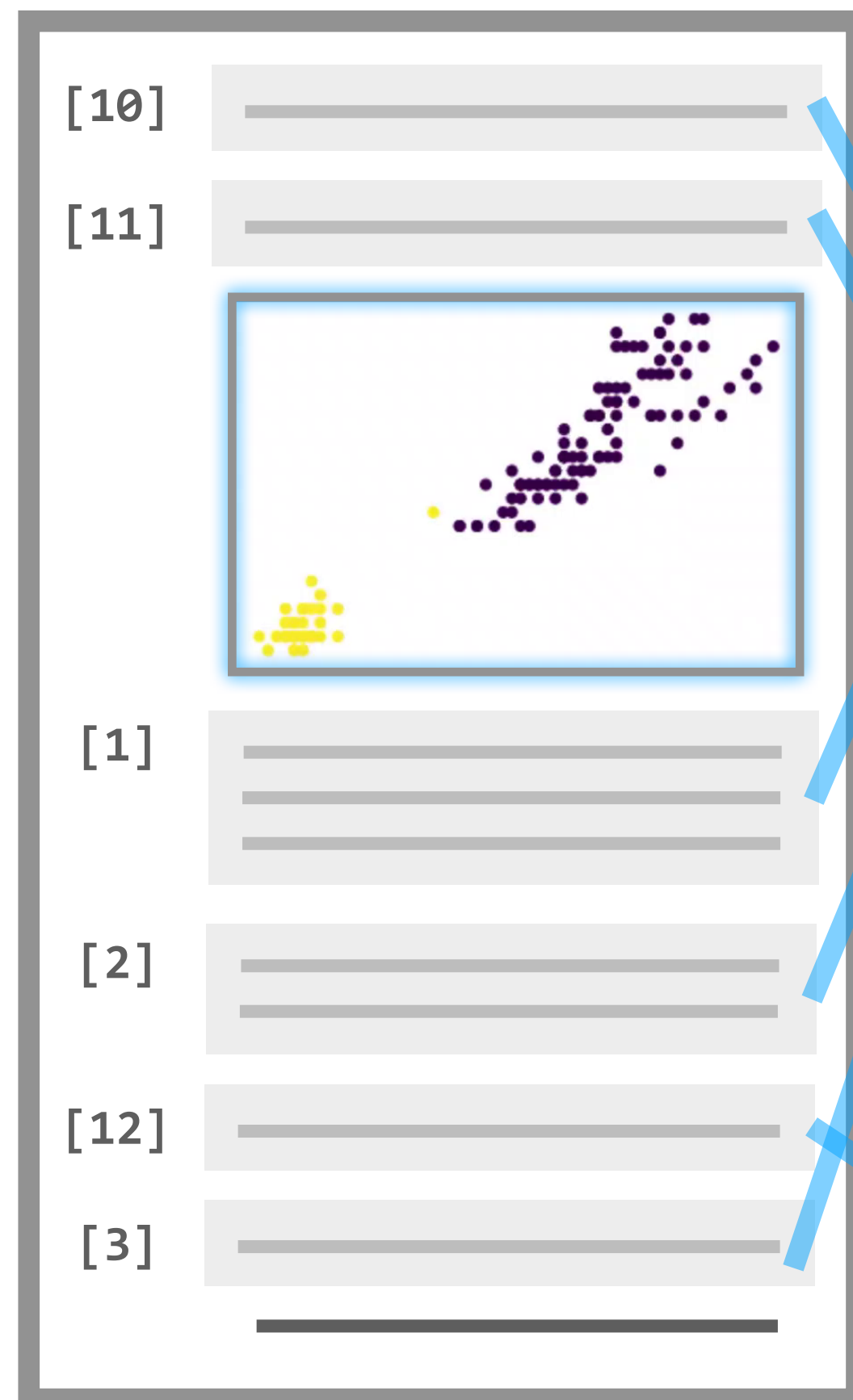


execution time

# Implementation: Slicing Notebooks

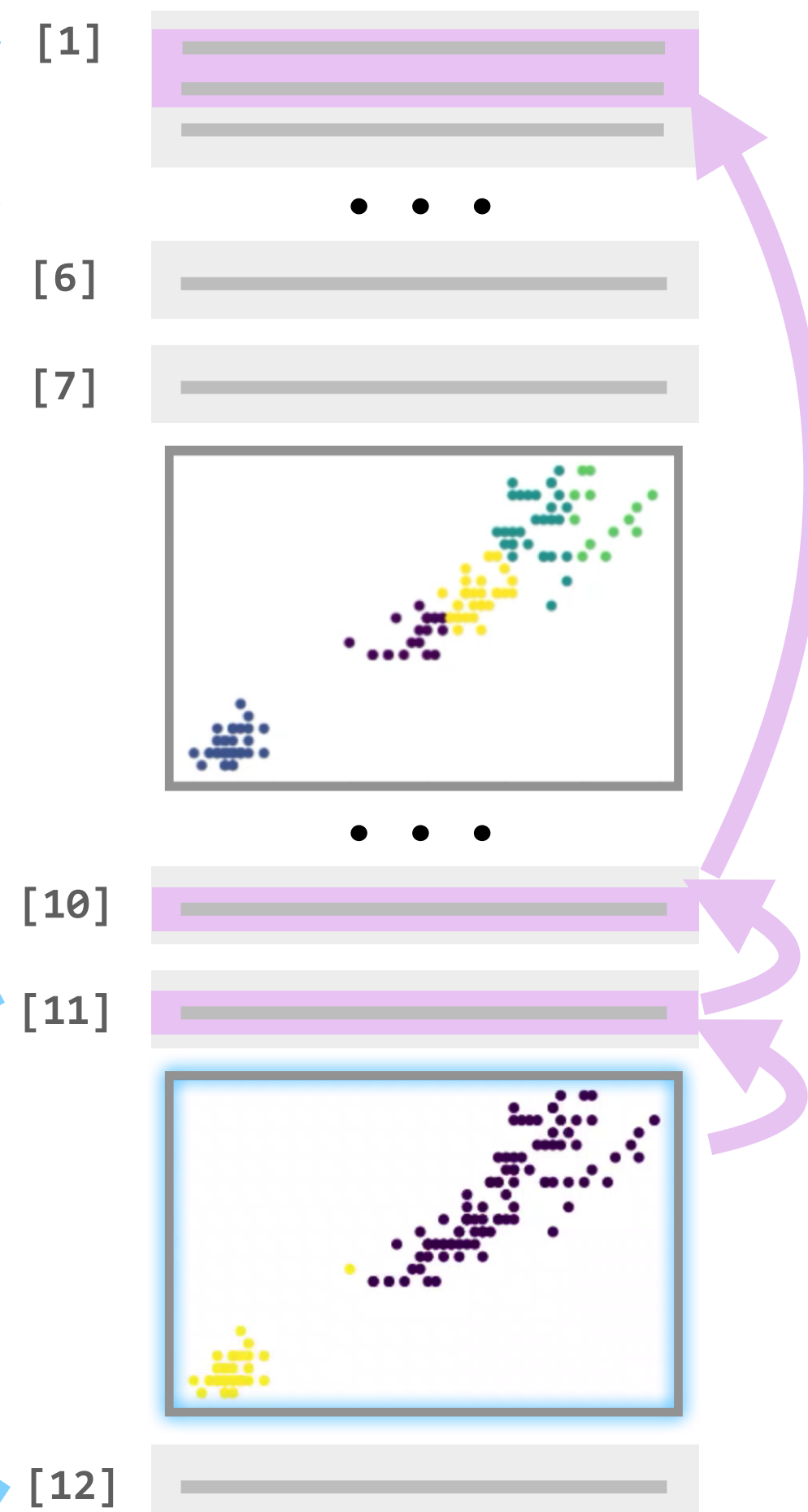
① Notebook

some cells missing,  
some cells out-of-order



② Execution Log

all cells present, in-order



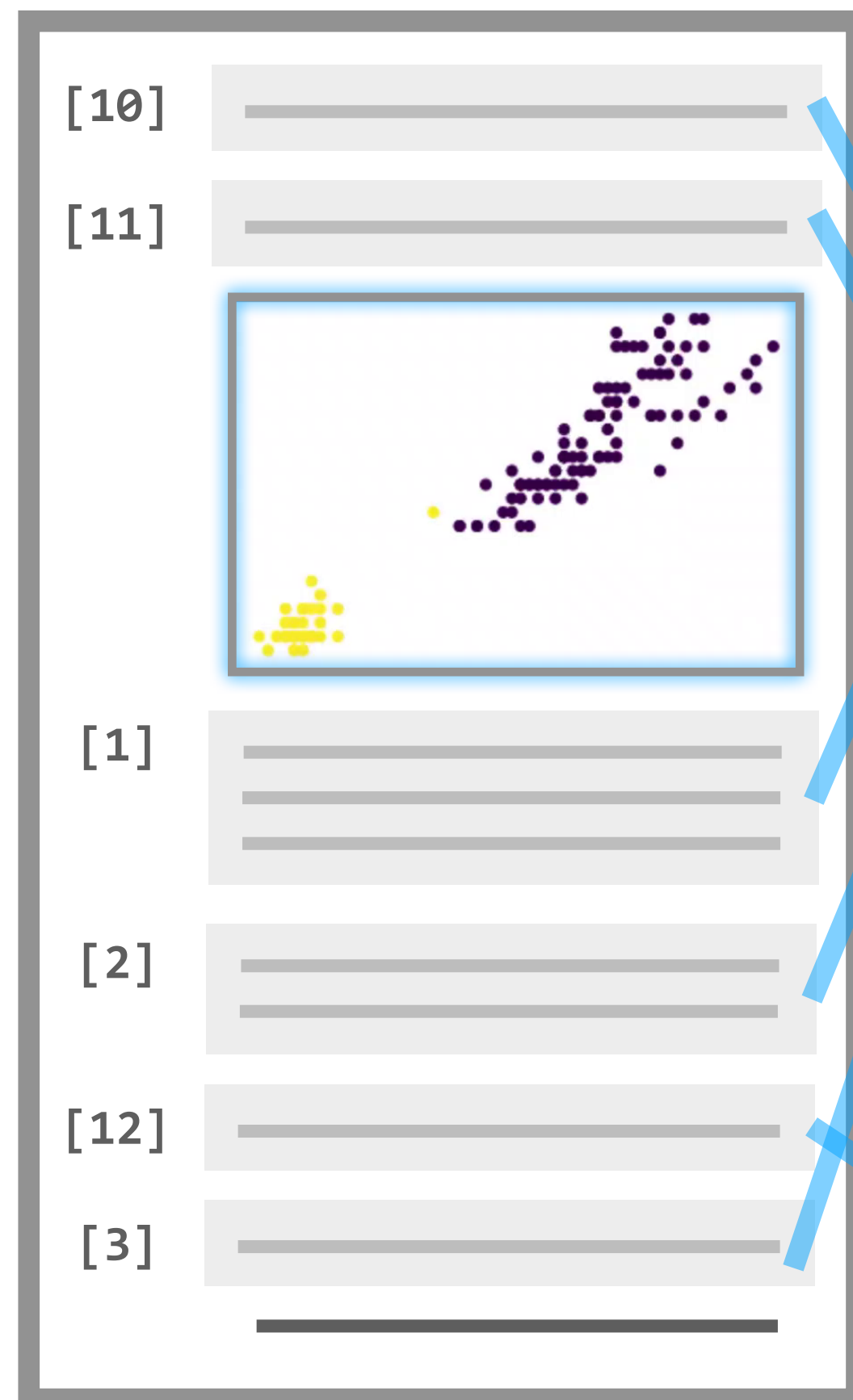
③ Program Slices [Weiser '81]

execution time

# Implementation: Slicing Notebooks

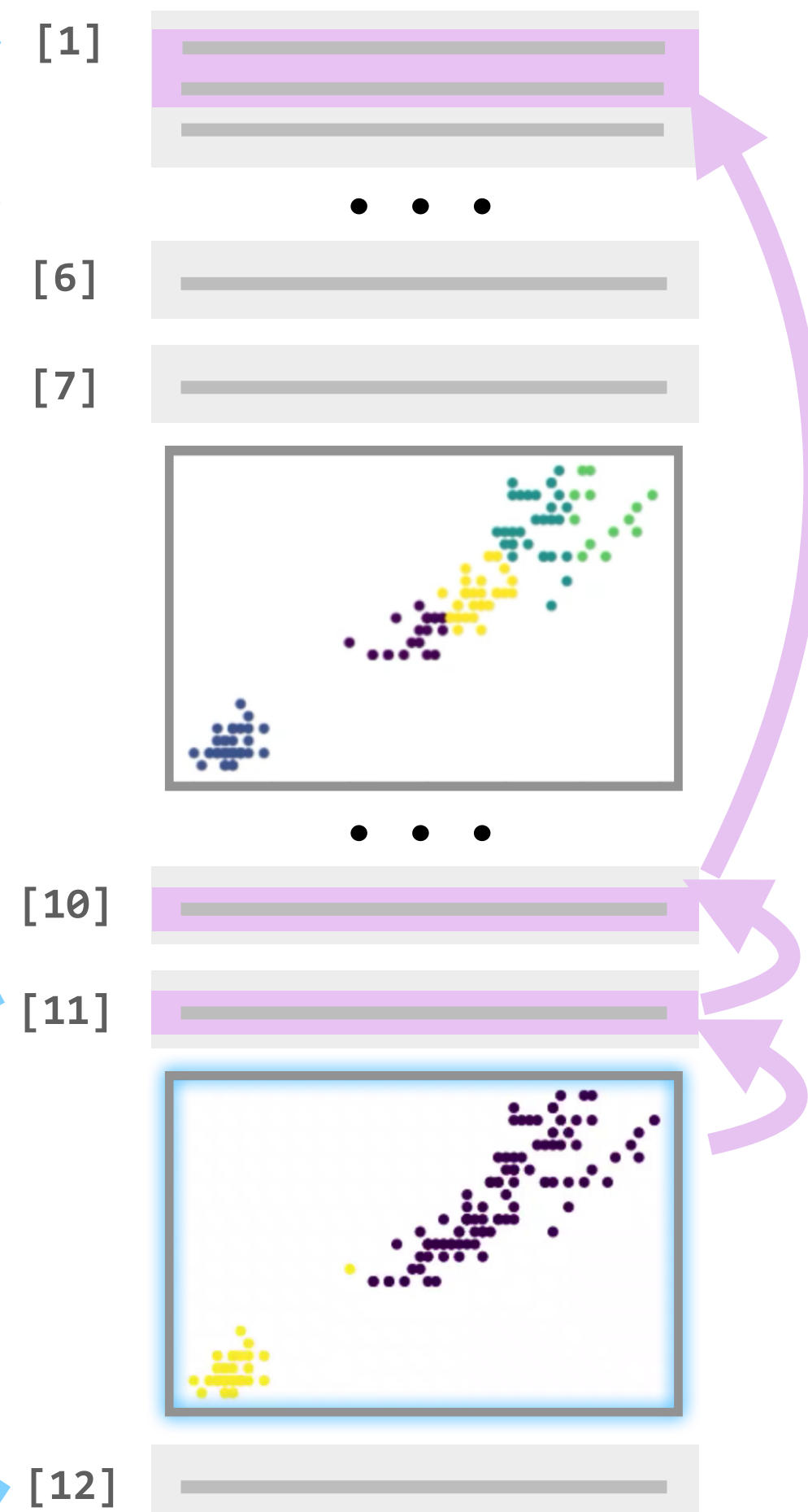
## ① Notebook

some cells missing,  
some cells out-of-order



## ② Execution Log

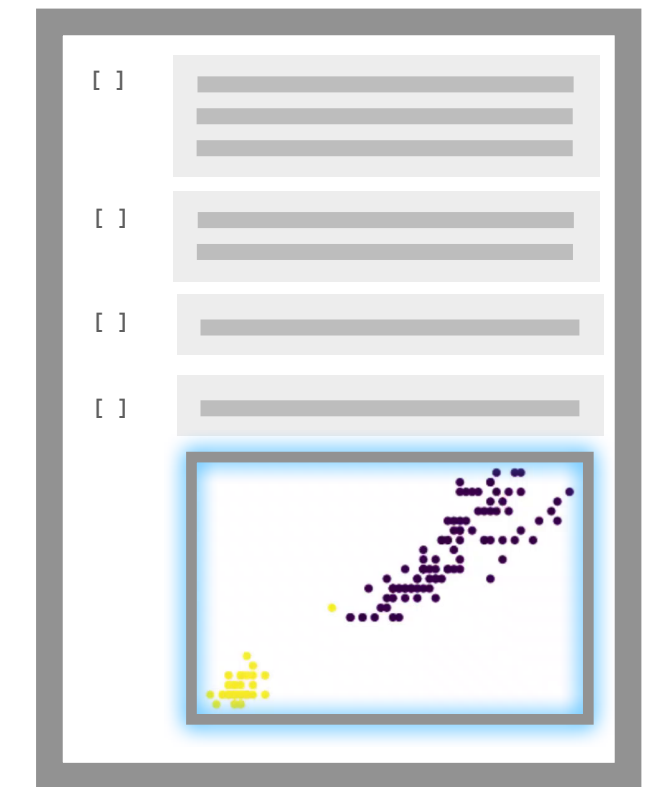
all cells present, in-order



## ③ Program Slices [Weiser '81]

which can be used to make...

cleaned, ordered  
notebooks  
(preserve cell  
boundaries and  
outputs)



versioned  
results  
(slice all cell  
versions)





# Cleaning and Exploring Messy Notebooks

*A Sample of Recent Research*

## output recipes

The screenshot shows a Jupyter Notebook interface. At the top, there's a search bar and a button to reproduce output. Below that is a table with columns: IncidentNum, Category, Descript, and Date. The table contains two rows of data. Below the table, there are two code cells. The first cell, labeled 'step # 0', contains import statements for sys, os, pandas, numpy, matplotlib, and seaborn. The second cell, labeled 'step # 1', contains code to read a CSV file and print its shape.

	IncidentNum	Category	Descript	Date
0	150060275	NON-CRIMINAL	LOST PROPERTY	Mo
1	150098210	ROBBERY	ROBBERY, BODILY FORCE	Su

```
step # 0 v0 Today 7:15 PM
import sys,os
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

step # 1 v9 Today 7:15 PM
d_crime=pd.read_csv('./Police_Incidents.csv')
print d_crime.shape
```

*Interactions for Untangling  
Messy History in a  
Computational Notebook*  
Kery et al., VL/HCC '18

## artifact explorer

The screenshot shows an artifact explorer interface. It displays a list of code cells with their versions and a heatmap visualization. A callout box labeled 'cell version diffs' points to the heatmap. The heatmap shows the differences between different versions of the code cells. The x and y axes of the heatmap are labeled with various code cell identifiers.

NOTEBOOK > CELL 12 > ASSIGN 248

V2 ASSIGN 248 FROM CODE CELL 12, NOTEBOOK #32

```
so = s.sort_values(kind="quicksort", ascending=False)
```

V1 ASSIGN 248 FROM CODE CELL 12, NOTEBOOK #32

```
so = s.sort_values(kind="quicksort")
```

*Towards Effective Foraging  
by Data Scientists to Find  
Past Analysis Choices*  
Kery et al., CHI '19

## cell folding

The screenshot shows a Jupyter Notebook interface. A code cell is selected, and a callout box labeled 'cell folding' points to the cell. The code cell contains a polynomial fit and a plot. The plot shows a set of data points and a fitted curve. The callout box also shows the number of lines in the cell (8 lines).

Imports + Data 8 lines >

```
p = np.poly1d(np.polyfit(x,y,7))
xp = np.linspace(4, 13, 100)

plt.plot(x, y, '.', xp, p(xp), '-')
```

[<matplotlib.lines.Line2D at 0x10eb30400>, <matplotlib.lines.Line2D at 0x10eb30588>]

*Aiding Collaborative Reuse of  
Computational Notebooks with  
Annotated Cell Folding*  
Rule et al., CSCW '18

tabbed browsing  
of cell versions

*Design and Use of  
Computational  
Notebooks*  
Rule, Ph.D. Thesis, '18

# **Evaluating Code Gathering Tools**

**Q1.** What is the meaning of "cleaning"?

**Q2.** How do analysts use code gathering tools during exploratory data analysis?

# A Qualitative Study of Gathering

Participants:  $N = 12$  professional data analysts

Cleaning Task  $\times$  2: Clean a computational notebook, with and without code gathering tools.

Exploration: Rank movies in from a movies dataset.  
Use code gathering tools as you wish.



# Q1. The Meaning of "Cleaning"

Picking a subset of cells [P1-P12]...  
and removing the rest [P8, P10-12].

*"I picked a plot that looked interesting and, if you think of a dependency tree of cells, walked backwards and removed everything that wasn't necessary."*

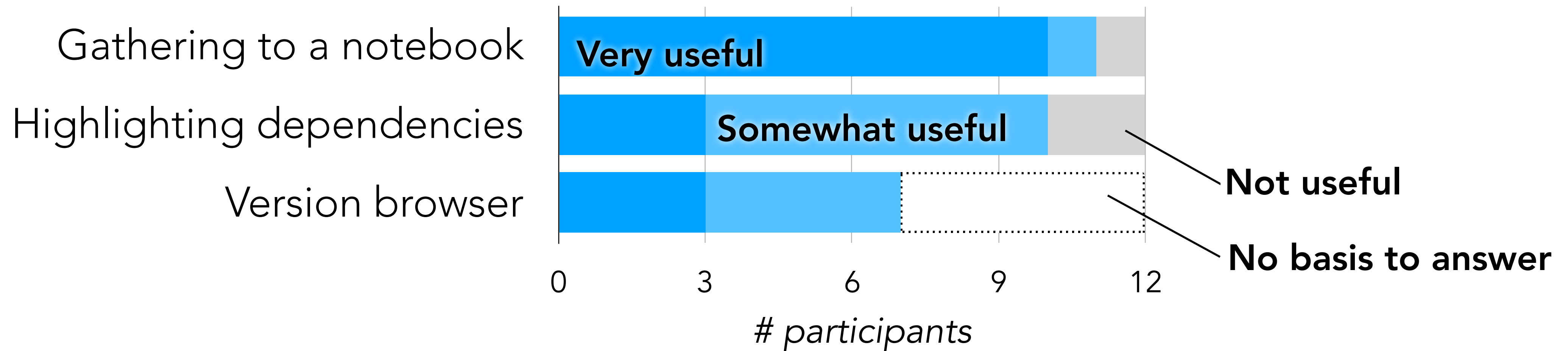
... And many additional stages:

writing documentation [P1, P5, P7, P10, P11]      merging cells [P11]

polishing visualizations [P1, P6]      restructuring code [P3, P4, P6, P12]

integrating with version control [P7]

## Q2. How do analysts use code gathering tools during exploratory data analysis?



Participants described gathering to a notebook as "beautiful" and "amazing": it "hits the nail on the head."

# Some Observed Uses of Gathering Tools

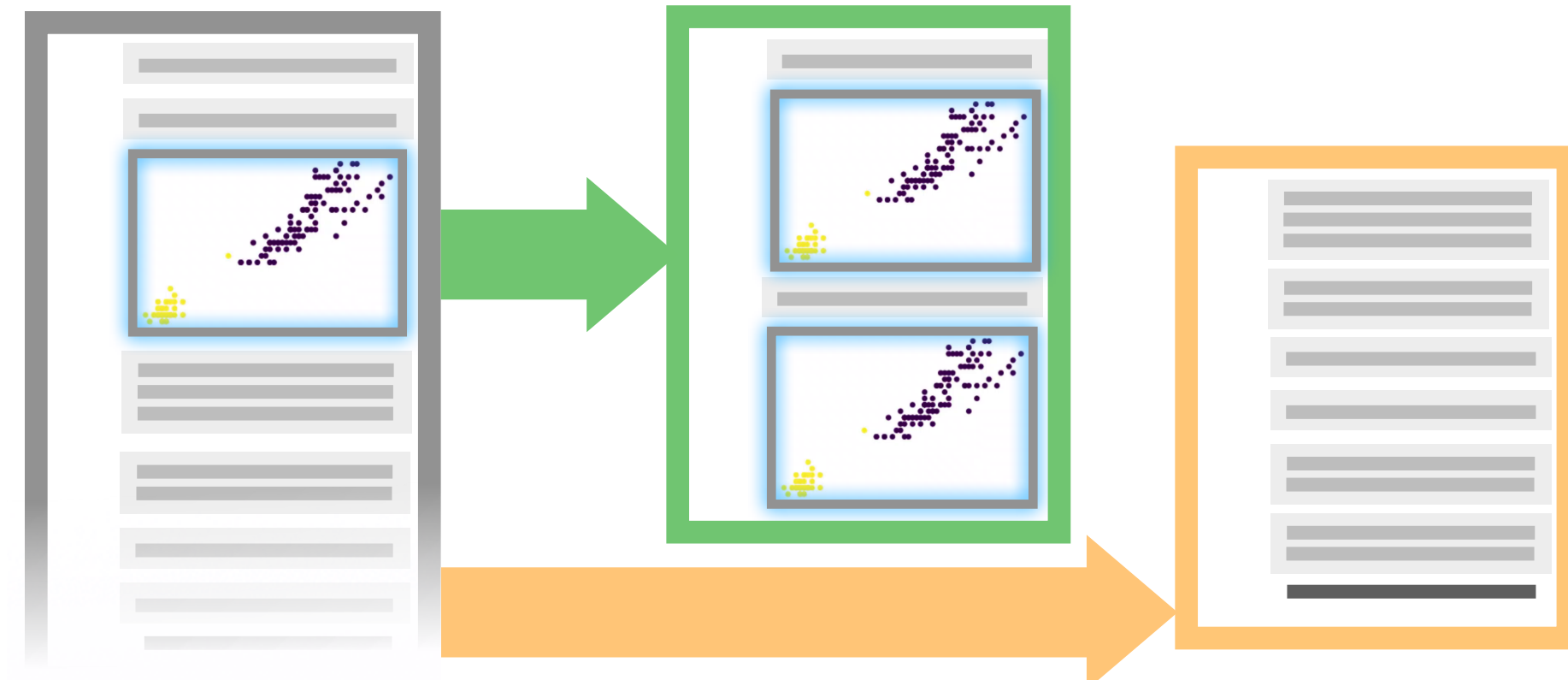
*"Finishing moves"*



Lightweight branching



Gathering for multiple audiences



Creating personal references





# Takeaways from Study

Q1. Gathering covers an important *yet incomplete* set of notebook cleaning tasks.

Q2. Code gathering tools can be picked up quickly and *readily applied to new use cases*.

# \$ jupyter labextension install nbgather

File Edit View Run Kernel Tabs Settings Help

Exploratory Data Analysis.i X

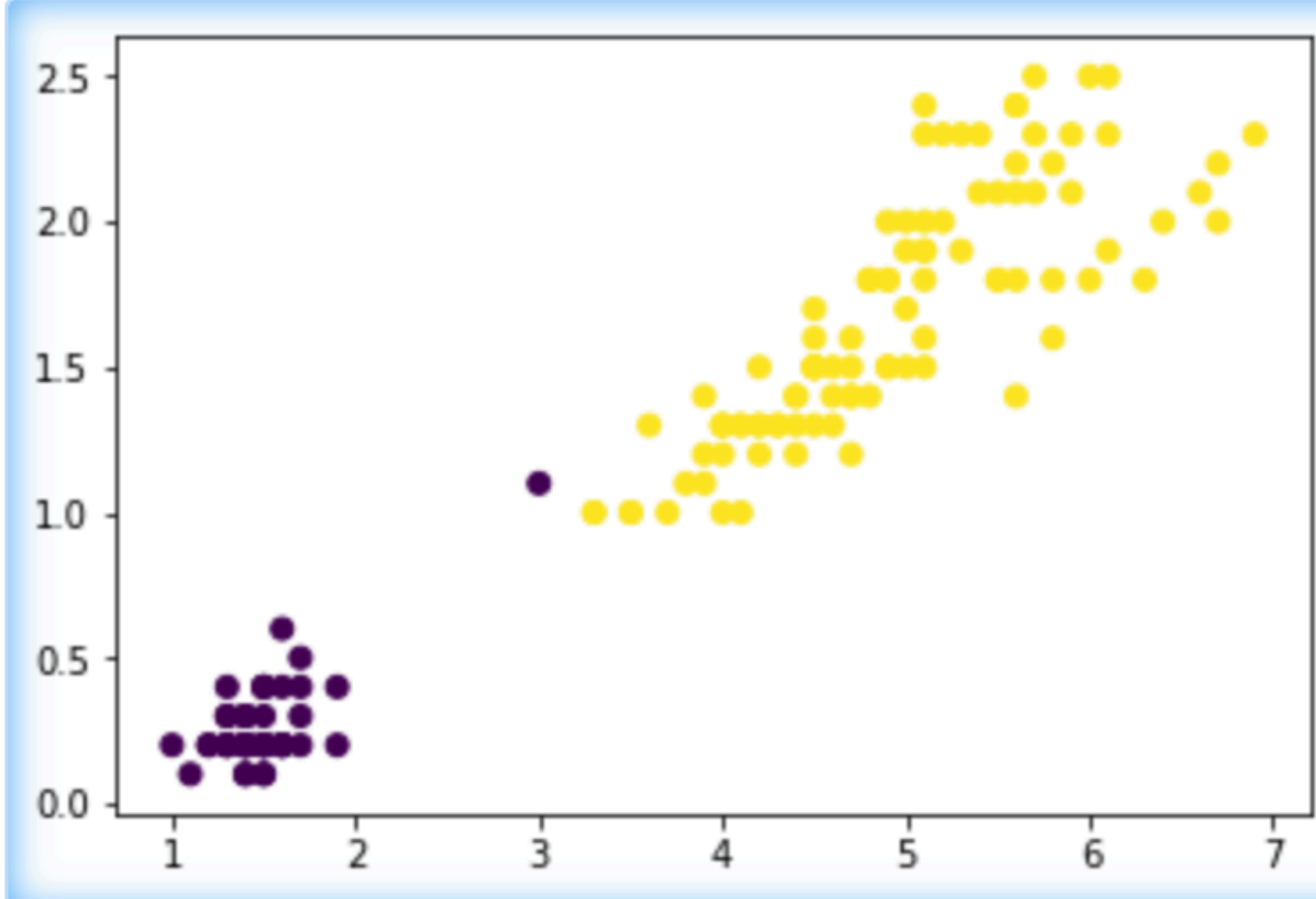
Code

Gather to: Cells Notebook Version Browser Clear Python 3

```
[19]: clusters = KMeans(n_clusters=2).fit(data).labels_
```

```
[20]: scatter(petal_length, petal_width, c=clusters)
```

```
[20]: <matplotlib.collections.PathCollection at 0x1067baf98>
```



Gather

Contributions encouraged:  
[github.com/Microsoft/gather](https://github.com/Microsoft/gather)

```
[13]: from matplotlib.pyplot import scatter
      from sklearn.cluster import KMeans
      from sklearn import datasets
```

```
[8]: data = datasets.load_iris().data[:,2:4]
     petal_length, petal_width = data[:,0], data[:,1]
```

```
[21]: petal_length, petal_width = data[:,1], data[:,0]
```