BIOS-584 Python Programming (Non-Bios Student)

Week 02

Instructor: Tianwen Ma, Ph.D.

Department of Biostatistics and Bioinformatics,

Rollins School of Public Health,

Emory University

Lecture Overview

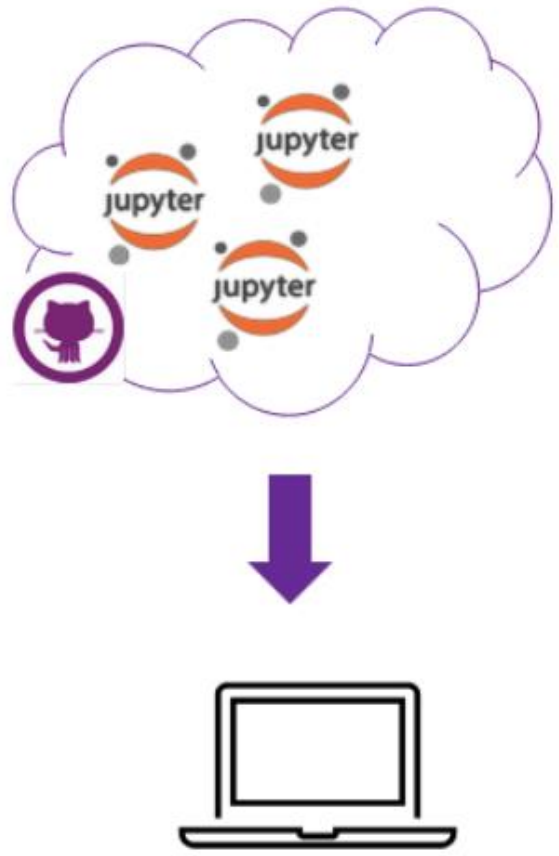
- Recap on GitHub
- Write a Sample Jupyter Notebook
- Variables
- Data Type
  - Lists
- Basic Visualization using matplotlib

Recap on GitHub

- Access: Download a remote GitHub repository to your local computer
  - Cloning
  - Forking
- Edit: Make local changes to the repository and update them to the cloud
  - Add
  - Commit
  - Push

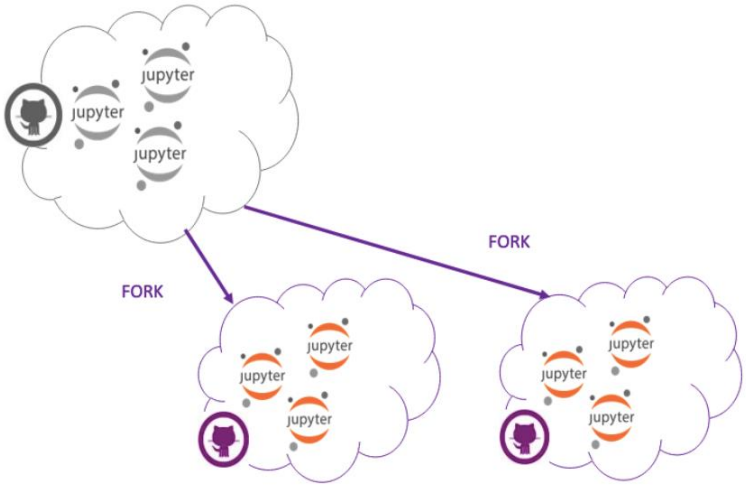
Cloning

- Make a local copy of a repository and download it to your local computer
- Reference the original repository
  - Cannot push changes to it unless granting permission from the owner.



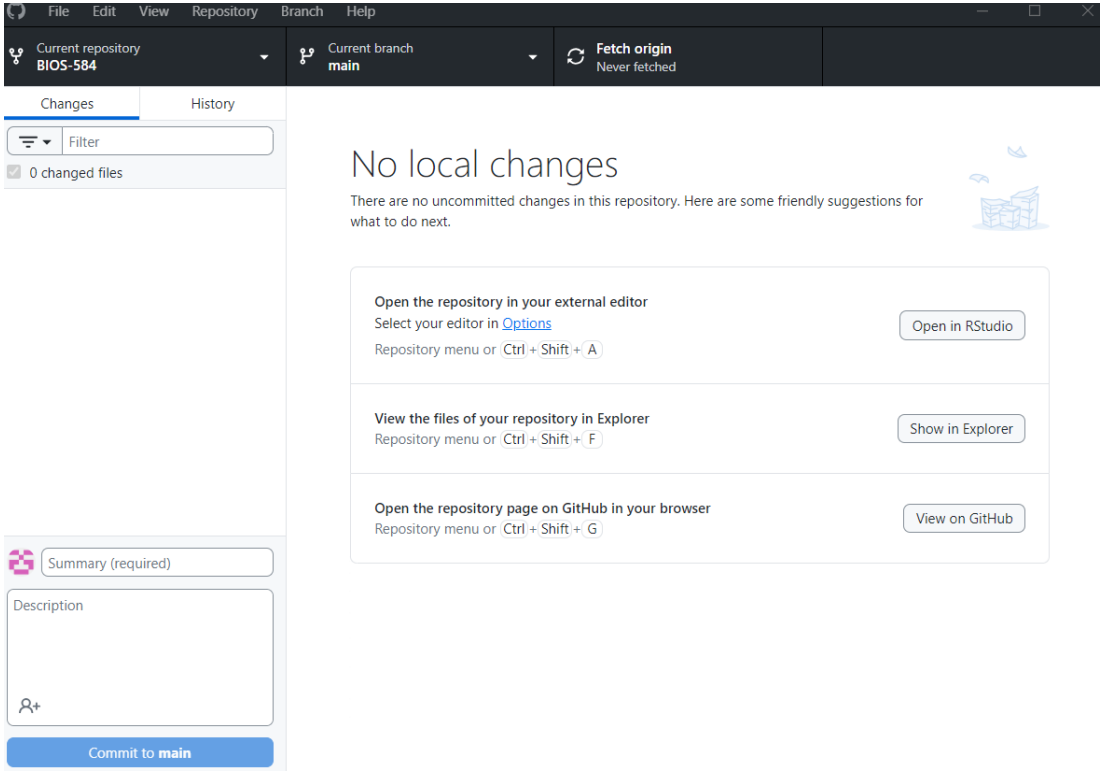
Forking

- Create your own copy of a repository in your remote GitHub page
- Contribute changes to your own copy without interfering with the original one
- Used to create a personal version for custom development



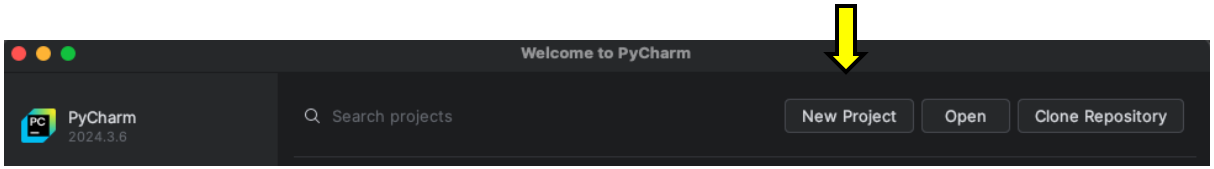
Status of Clone Repo to Desktop

- After successfully cloning repo to GitHub Desktop, you should see an interface below:

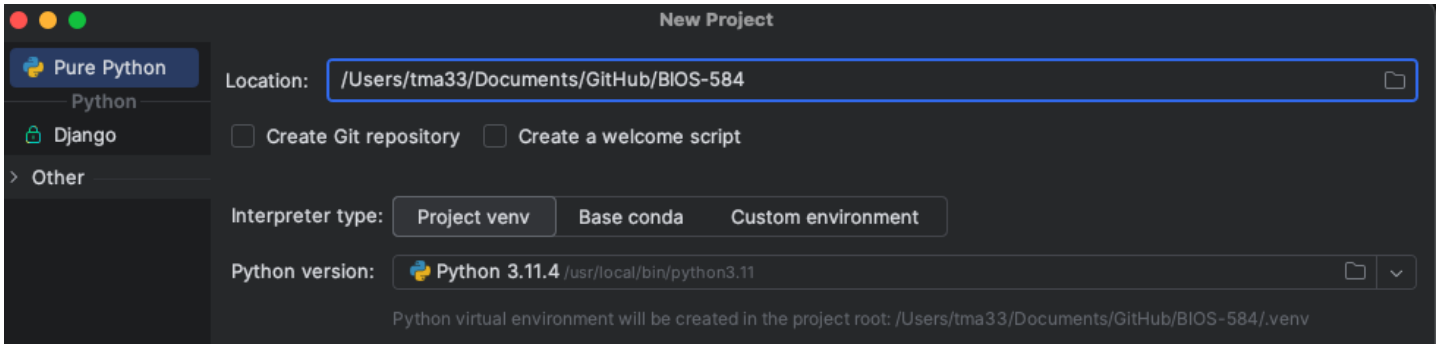


*Clarification

- To facilitate future work, you can create a PyCharm project under that GitHub repo you just created.

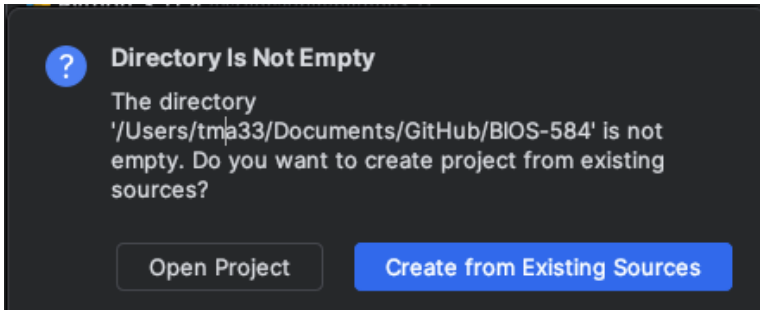


- In Location, type the directory of the local GitHub repo and click “Create” (snapshot not shown)



*Clarification

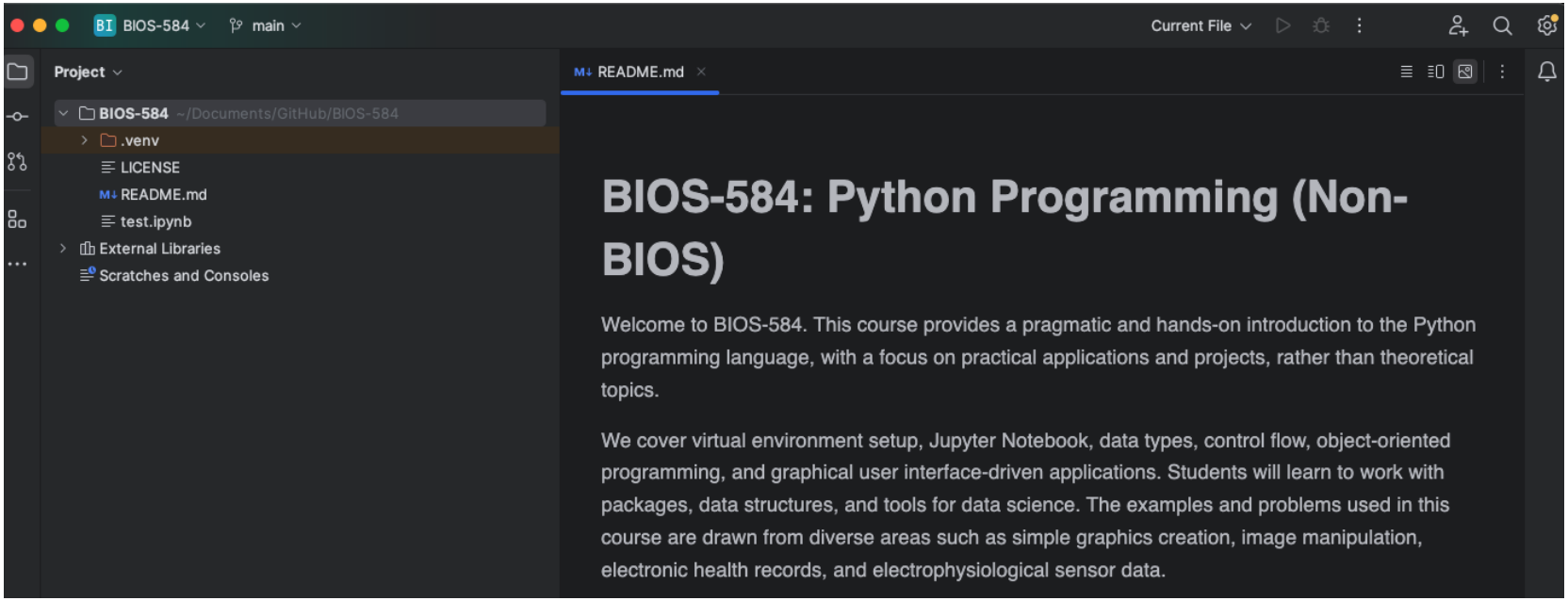
- A warning pops up, and click “Create from Existing Sources”



- It creates a new PyCharm project under the GitHub “BIOS-584” repo with a “.venv” folder.

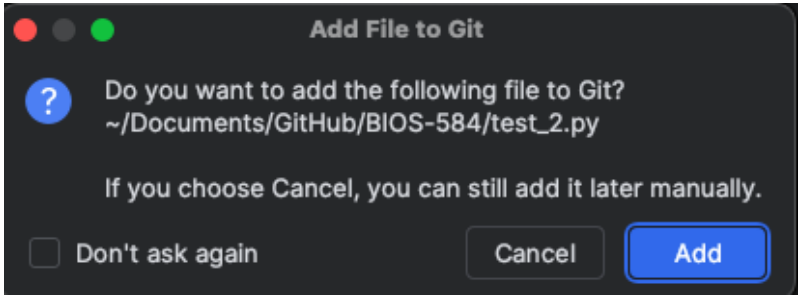
*Clarification

- It looks like this:



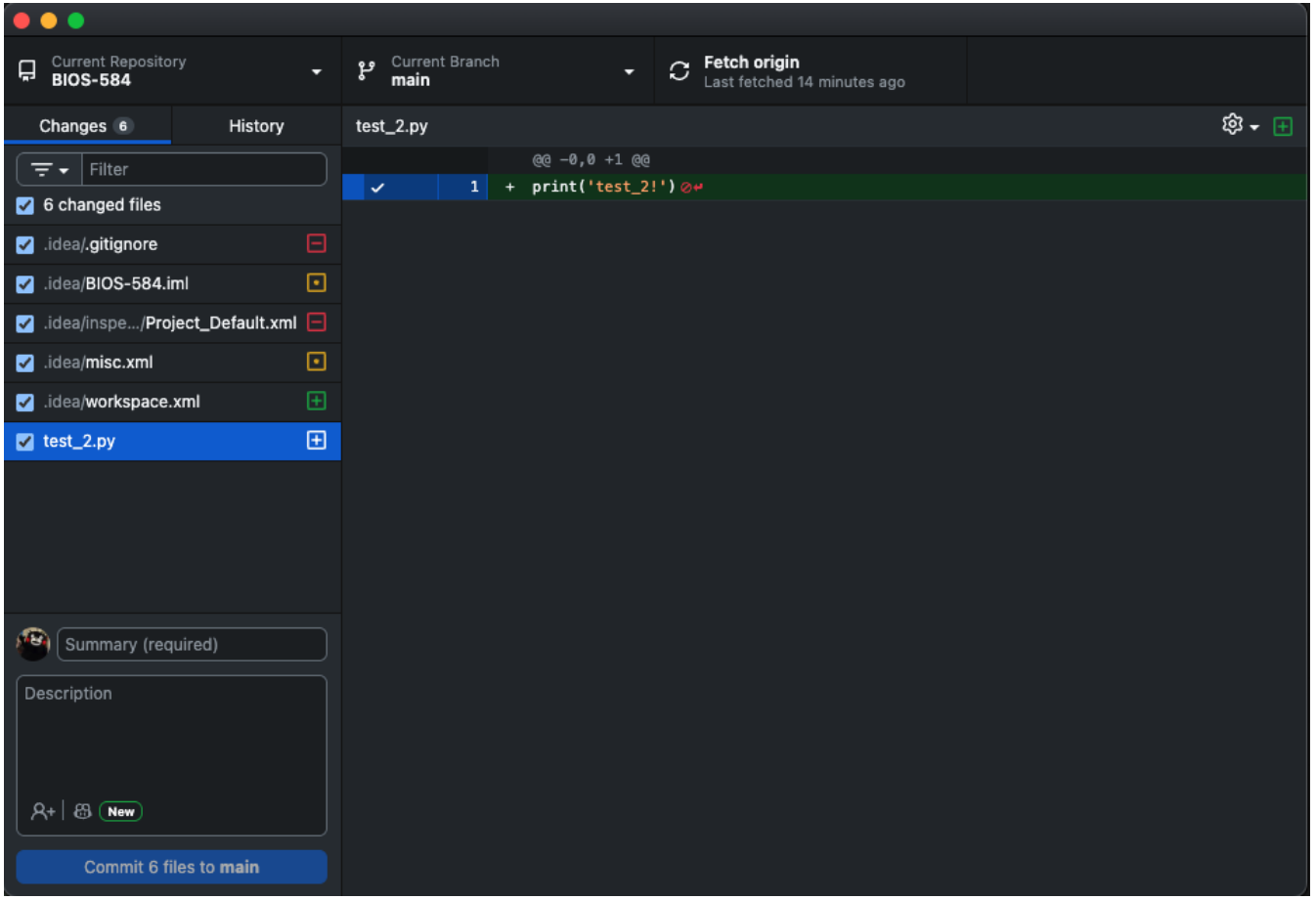
Add Changes

- When you create a new file, PyCharm asks if you want git to track changes in a file. Click “Add”
  - I create another “test_2.py”



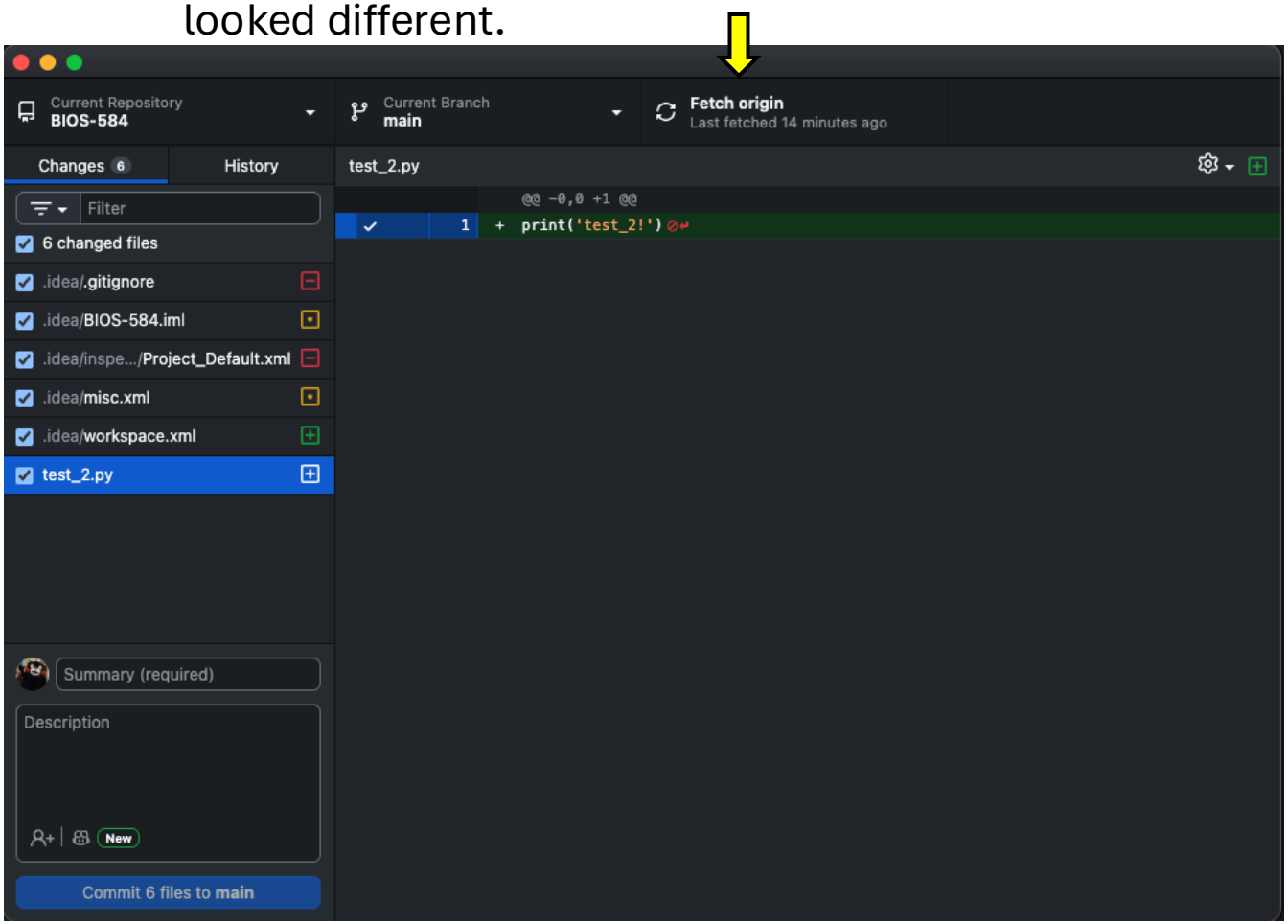
Commit Changes

You see the change here!



Push Changes

After the commitment is done, this will become “Push”. I recycled the same snapshot before I finished committing changes, so it looked different.



Fetch Changes

- You can use “Fetch Origin” to update your local repository with new changes from the online repository

Questions?

Jupyter Notebook

- We covered how to install and create Jupyter notebook with a brief command.
- Let’s write codes and texts!

Writing texts

- You can write texts in Jupyter notebooks using Markdown
- Markdown is a lightweight markup language that allows you to format text easily
- You can use headings, lists, links, images, tables, and more
- You can also use LaTeX to write equations in Jupyter Notebooks.

Writing codes

- You can write code in Jupyter Notebooks using code cells
- It is the same Python code that you would write in a .py file, but with the results displayed below the cell
- You can run the code cells by pressing Shift + Enter or clicking on the Run button
- You can also write comments in the code cells using the # symbol

Let’s take a try!

- Go over week-02-jupyter.ipynb

Questions?

Variables

- A container that stores data values
- Common variables include
  - Integers: whole numbers (int)
  - Floats: numbers with decimals (float)
  - Strings: texts (str)
  - Booleans: True or False (bool)

Variables

- We identify the type using “type()” function
- We can also “print()” function to display the value of a variable

```
1  type(2)
2  type(2.5)
3
4  print(type(2))
5  print(type(2.5))
6  print(type('BIOS-584'))
7  print(type(True))
8
9  # single or double quotation marks are the same here
10 type("BIOS-584")
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
```

```
str
```

Store Variables in Memory

- We store variables in memory using the assignment operator “=”.
- For example, we store the value 3 in the variable x.
- We can access the value of x by typing x.

```
1 x = 3
2 print(x)
```

```
3
```

Store Variables in Memory

- Variable names are case-sensitive.
- Variable name cannot start with a number.
- Variable names cannot contain spaces.
- Variable names cannot contain special characters, except for “_”

Basic Operations

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Exponentiation: **
- Matrix multiplication: @
  - Revisit it when we introduce “numpy” package.

```
1 print(1+1)
2 print(1*2)
3 print(3-2)
4 print(3/2)
5 print(type(3/2))
6 print(3**2)
```

```
2
2
1
1.5
<class 'float'>
9
```

Basic Operations

- Use parentheses for order of operations
- Number and string cannot be combined with “+”

```
1 (1 + 2) * 3

[18]:

9
```

```
[24]:

1 "BIOS" + 584
```

```
---------------------------------------------------------------
TypeError                                 Traceback (mo
Cell In[24], line 1
----> 1 "BIOS" + 584

TypeError: can only concatenate str (not "int") to str
```

Basic Operations

- Concatenation: String + String
- If you do not assign values to variables, + can be ignored.
- If you assign values to variables and use variable names to connect, + is required.

```
1 "BIOS" "-" "584"
```

```
[20]:

'BIOS-584'
```

```
1 x = 'BIOS'
2 y = '584'
3 x "-" y
```

```
  Cell In[22], line 3
    x "-" y
      ^
SyntaxError: invalid syntax
```

```
[23]:
```

```
1 x + "-" + y
```

```
[23]:

'BIOS-584'
```

Data Type (Lists)

- Lists are collections of items
- We can store different types of items in a list
- Lists are always enclosed in square brackets [ ]
- Elements are separated by commas ,
- We access elements in a list using their index (starting from 0)

```
1 # List of numbers
2 list_numbers = [1,2,3,4,5]
3 list_numbers_sq = [1,4,9,16,25]
```

```
1 print(type(list_numbers))
2 print(type(list_numbers_sq))
```

```
<class 'list'>
<class 'list'>
```

```
1 print(list_numbers)
2 print(list_numbers_sq)
```

```
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
```

Lists

- Lists support mixed types and nested lists.
- Use square brackets [ ] to access elements in a list
- If we want to access the first element in a list, we use list[0]
- Lists have their own functions (covered later)

```
# List with strings
# Example: Suppose you ask 5 people about their favorite color. The results:
list_colors = ["red","yellow","yellow", "green","red"]
print(list_colors)
```

```
['red', 'yellow', 'yellow', 'green', 'red']
```

```
# List with mixed types
list_mixed = ["red",1,"yellow",4,5, 3.5]
print(list_mixed)
```

```
['red', 1, 'yellow', 4, 5, 3.5]
```

```
# Lists can be nested too
another_list = [list_mixed, 3, 'h']
print(another_list)
```

```
[['red', 1, 'yellow', 4, 5, 3.5], 3, 'h']
```

```
list_colors[0]
```

```
'red'
```

Data Visualization

- We use matplotlib package to create plots.

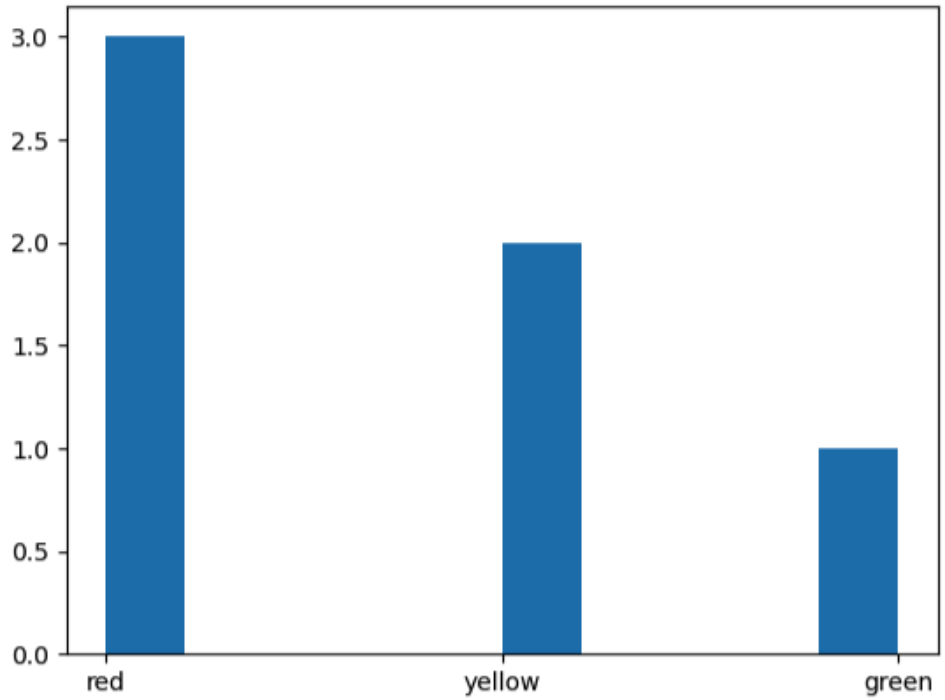
```
1 # The matplotlib library is used to create graphs
2
3 import matplotlib.pyplot as plt
```

- We start with simple histogram using “hist()”
  - We pass a list of parameters to “hist()”
  - We can also customize the plot by adding labels, titles, and changing the colors (more on that later)
  - We print the graph using the “show()” function

Data Visualization

```
1 list_list = list_colors + ['red']
```

```
1 # This creates a histogram with the "list_colors"
2 plt.hist(x = list_list)
3 plt.show()
```



Scatter Plots

- We create scatter plots using “scatter()” function
- It takes two lists as arguments:
  - The first list contains the x-coordinates
  - The second list contains the y-coordinates
- We use the scatterplot to visualize the relationship between two continuous variables
- We use “xlabel()”, “ylabel()”, and “title()” functions to label axes and the title.
- We use “savefig()” to save a copy to your desktop.

Scatter Plots

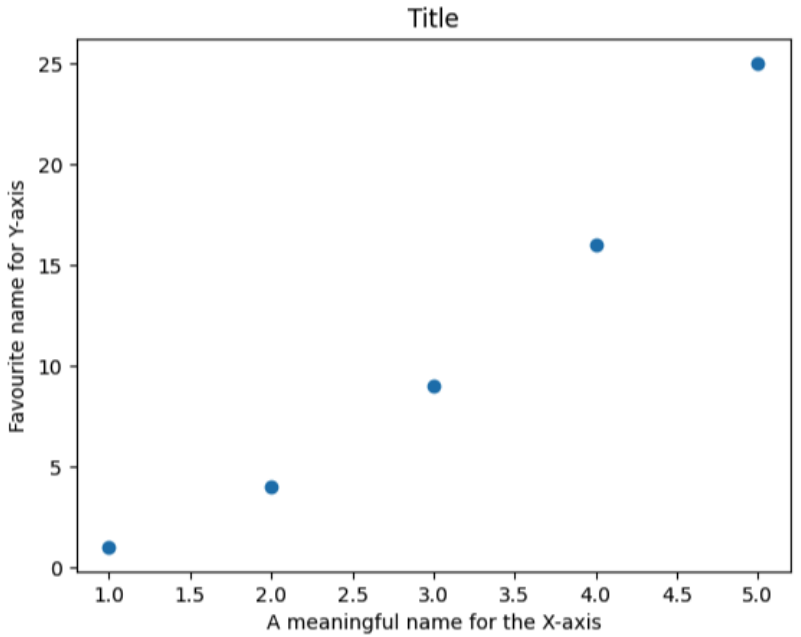
```
1 print(list(list_numbers))
2 print(list_numbers_sq)
```

```
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
```

```
1 type(list_numbers)
```

```
list
```

```
1 plt.scatter(x = list_numbers, y = list_numbers_sq)
2 plt.xlabel("A meaningful name for the X-axis")
3 plt.ylabel("Favourite name for Y-axis")
4 plt.title('Title')
5 plt.savefig('./my_plot.png')
6 plt.show()
```



“./my_plot.png” is a relative directory
that saves a picture to your PyCharm project directory.

Let’s take a try!

- Go over week-02-variables-lists.ipynb