Expanding AI/ML Coursework on Your Campus Using Jupyter Notebooks powered by NRP

Building Hands-on Lessons with Jupyter Notebooks

J. Alex Hurt
University of Missouri

Introduction

- The previous set of slides and examples went over how to go from a standard slide / homework-based Syllabus to a Lesson Plan using Jupyter

- This portion of the tutorial will go more in-depth on building hands-on lessons for topics in AI/ML, including topics such as:
  - General Approach and Methodology
  - Module Layout and Setup
  - Jupyter Cell Types and their usage for instruction
  - Hands-on Example building a Jupyter Notebook Lab for AI/ML

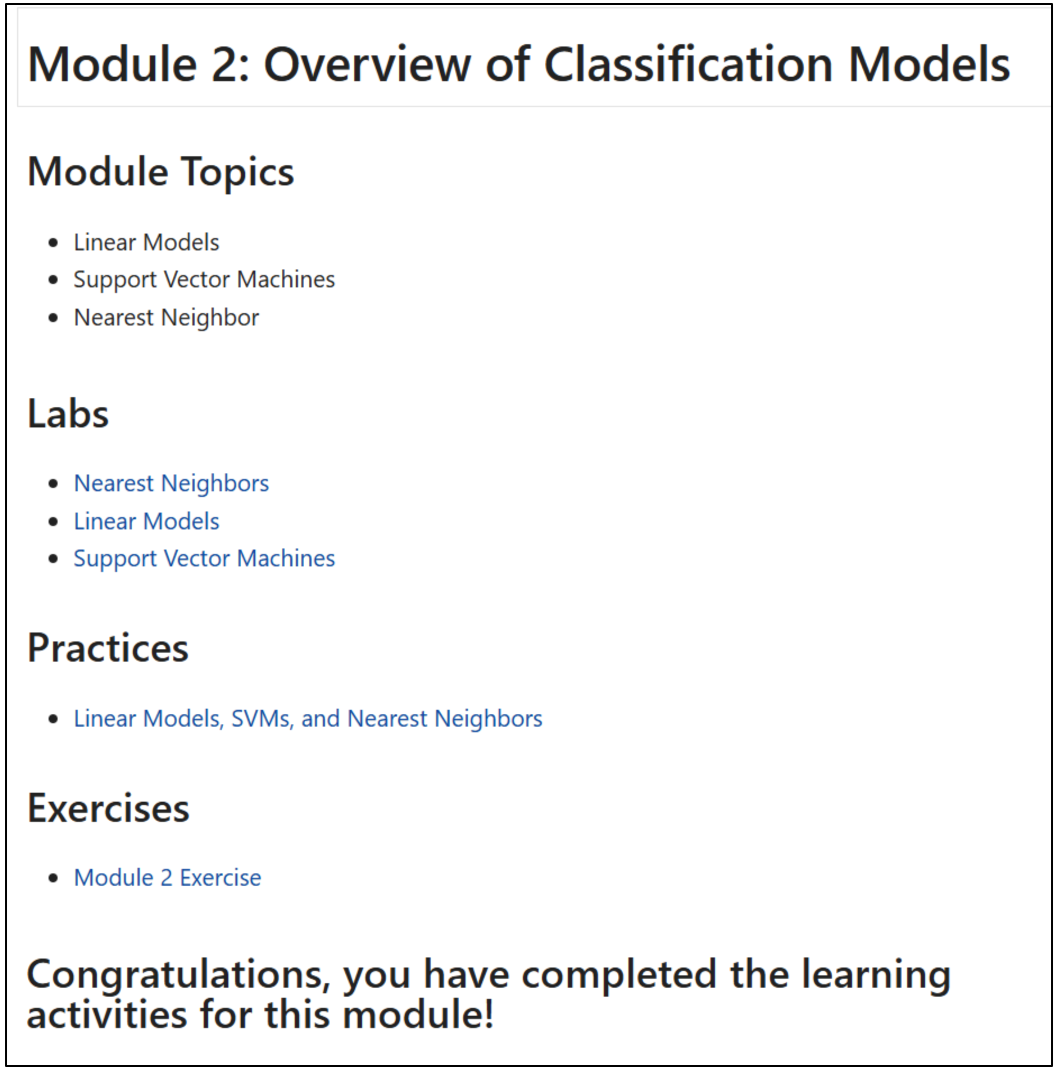
General Approach

- The general approach to building Jupyter notebooks for AI/ML coursework:

Each notebook is a standalone, fully independent example that leans on prior knowledge and exercises to introduce, reinforce, or assess the understanding of a concept, theory, or application of AI/ML

Module Layout and Setup

- Modules can be laid out a variety of ways
- This tutorial will provide the most common one used at MU for AI/ML Coursework
- Each module is broken down into three types of notebooks:
  - **Lab** - Concept / Application Introduction
    - 95% - 100% worked out examples with little to no student involvement / input
  - **Practice** - Concept / Application Reinforcement
    - Alternatively, Concept Reinforcement with Concept / Application Introduction
    - 50% - 80% worked out examples with some student involvement / input
  - **Exercise** - Concept / Application Assessment
    - Alternatively, Concept Assessment with Concept / Application Reinforcement
    - 0% - 10% worked out examples with almost entirely student input



Module 2: Overview of Classification Models

Module Topics

- Linear Models
- Support Vector Machines
- Nearest Neighbor

Labs

- Nearest Neighbors
- Linear Models
- Support Vector Machines

Practices

- Linear Models, SVMs, and Nearest Neighbors

Exercises

- Module 2 Exercise

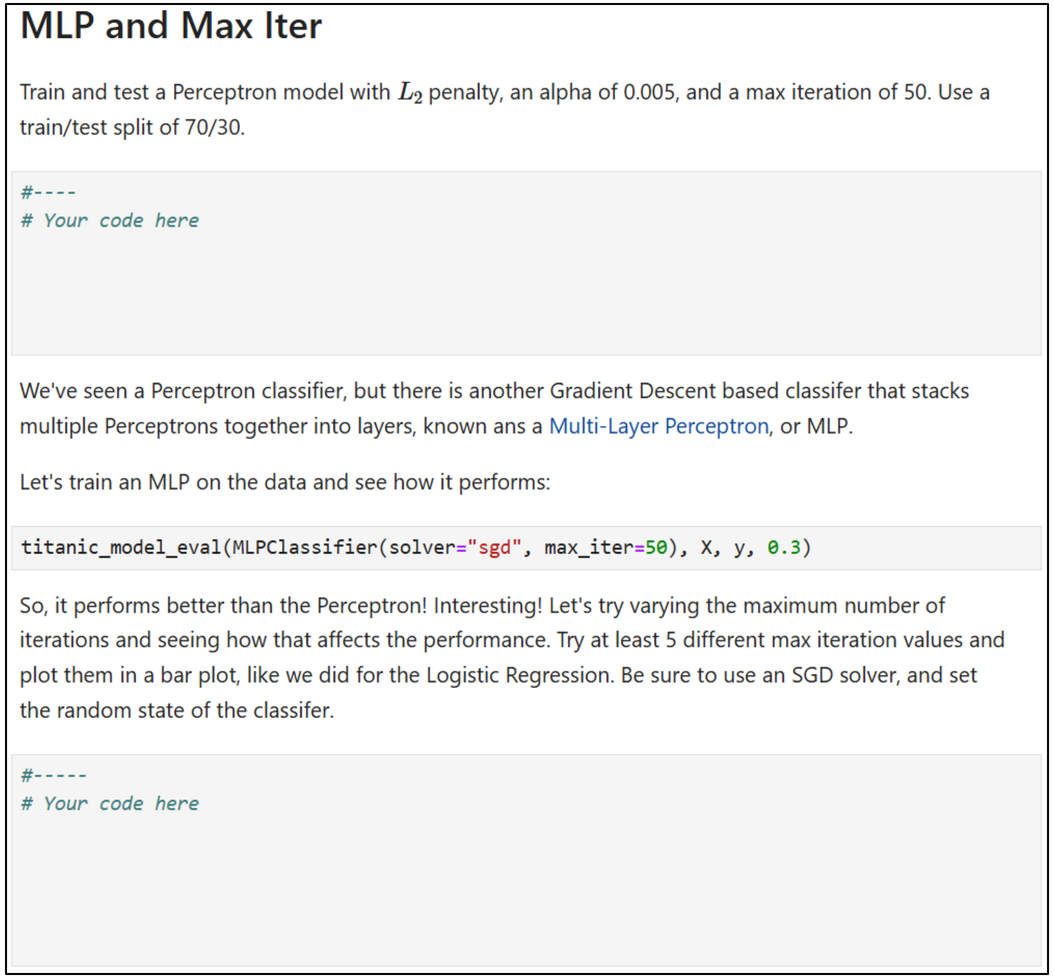
Congratulations, you have completed the learning activities for this module!

Module Labs

- Each lab is a completely or nearly completely filled out example used to **introduce a concept**, in which all concepts are explained clearly using a variety of Code, Markdown, Math, and External Links
- Labs serve to introduce a student to a concept or application
  - K-Nearest Neighbors
  - Differences Between Fully and Semi-Supervised Learning
  - Calculating Performance Metrics of Trained ML Models
- Generally, the only student involvement (if any) is to copy and paste a previous cell and change very little:
  - Change K = 5 to K = 7 for K-Nearest Neighbors Classifier
  - Change K = 5 to K = 10 for K-Fold Cross Validation
- The labs are the “lecture” material, i.e., they are what are walked through in class and what students use as a basis for **Practices** and **Exercises**

Module Practices

- Each practice is a partially filled out example used to **reinforce a concept**, in which some of the concepts are fully worked out using a variety of Code, Markdown, Math, and External Links
  - Students use the same model (reinforcing that model's behavior), but a new dataset or experimental design pattern is introduced
  - Students must create, train, and test model but all preprocessing is fully worked out
- Generally, the student involvement is derived from previous labs and practices within the current module and previous modules
  - Oftentimes, copying code from previous modules and then editing from there
- The practices are the advanced lecture material or small homework assignments
  - Answers to practices are covered in lecture and distributed to students prior to Concept Assessment with **Exercises**
- I generally encourage my students to try new ideas in Practices, letting their curiosity and **intuition** grow
  - Intentionally vague questions that allow students to set parameters: "*Train a Linear ML model*"
  - Require a certain number or style of response without requiring an exact answer: "*Try three more permutations and state the best performer along with your analysis of why that model performed best*"



MLP and Max Iter

Train and test a Perceptron model with L_2 penalty, an alpha of 0.005, and a max iteration of 50. Use a train/test split of 70/30.

```
#----
# Your code here
```

We've seen a Perceptron classifier, but there is another Gradient Descent based classifer that stacks multiple Perceptrons together into layers, known ans a Multi-Layer Perceptron, or MLP.

Let's train an MLP on the data and see how it performs:

```
titanic_model_eval(MLPClassifier(solver="sgd", max_iter=50), X, y, 0.3)
```

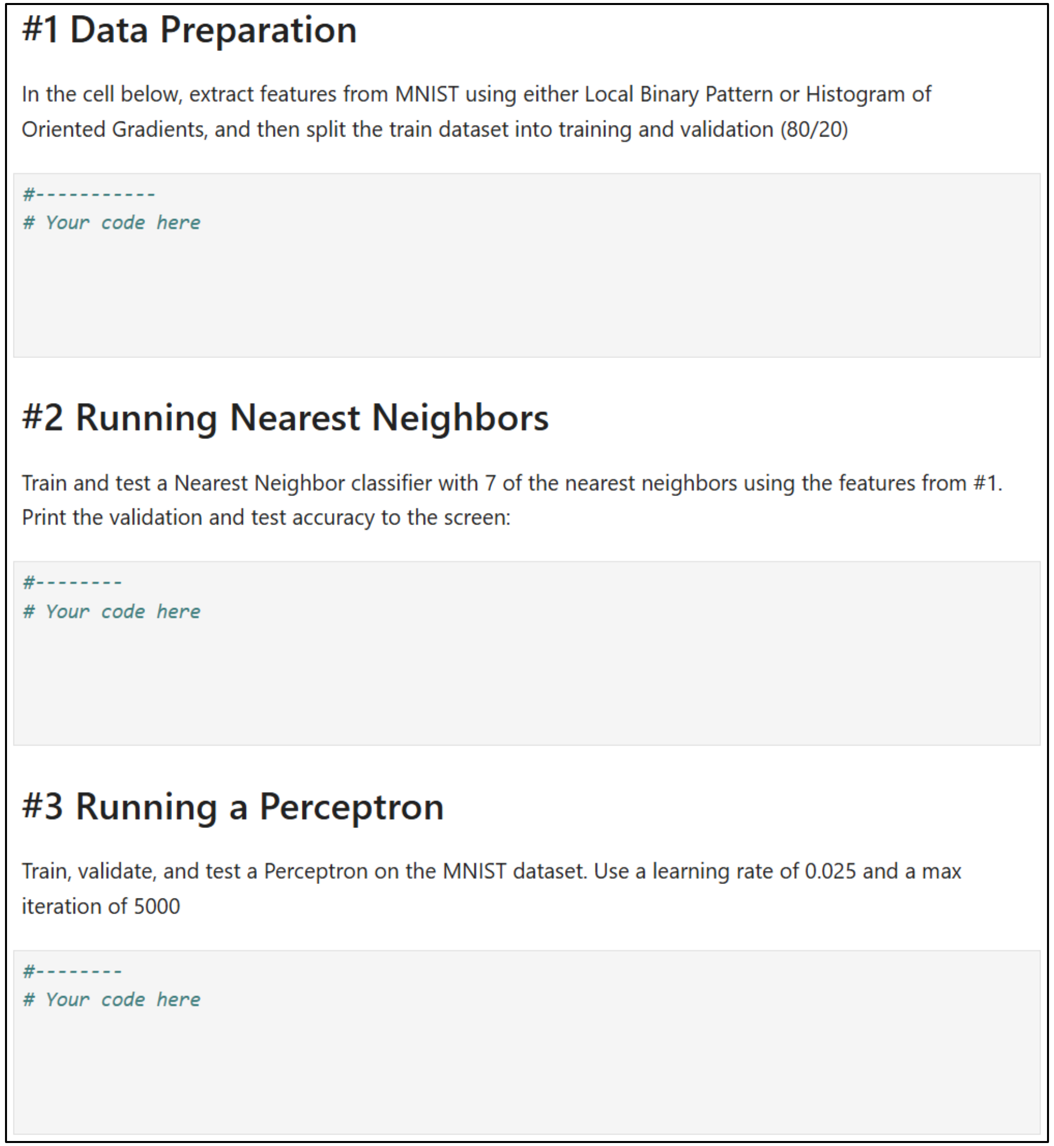
So, it performs better than the Perceptron! Interesting! Let's try varying the maximum number of iterations and seeing how that affects the performance. Try at least 5 different max iteration values and plot them in a bar plot, like we did for the Logistic Regression. Be sure to use an SGD solver, and set the random state of the classifer.

```
#-----
# Your code here
```

Module Layout

Module Exercises

- Each exercise is a nearly empty notebook used to **assess student's understanding** of previously introduced and reinforced concepts
  - Exercises serve to ensure that the concepts and applications being taught in a given module are being understood
- Students are required to do nearly everything in these modules, usually with only the library imports being given to them
  - This ensures that students can go from nothing to fully functional ML workflows using the concepts in a given module
- Difficulty generally varies, with the exercise becoming more difficult and requiring more in-depth understanding as it progresses
- Exercises almost always combine multiple concepts so that previous module's content are continually being reinforced and used
  - If Module 2 is feature selection, then not only will that be assessed in Module 2's exercise, but the exercise for Module 3 or 4 will also use feature selection in combination with the new concepts for that module
- The exercises are the larger homework assignments
  - Answers to exercises are covered in lecture following the due date to ensure students have a functional understanding of a module's concepts before moving to the next



#1 Data Preparation

In the cell below, extract features from MNIST using either Local Binary Pattern or Histogram of Oriented Gradients, and then split the train dataset into training and validation (80/20)

```
#-----------
# Your code here
```

#2 Running Nearest Neighbors

Train and test a Nearest Neighbor classifier with 7 of the nearest neighbors using the features from #1. Print the validation and test accuracy to the screen:

```
#--------
# Your code here
```

#3 Running a Perceptron

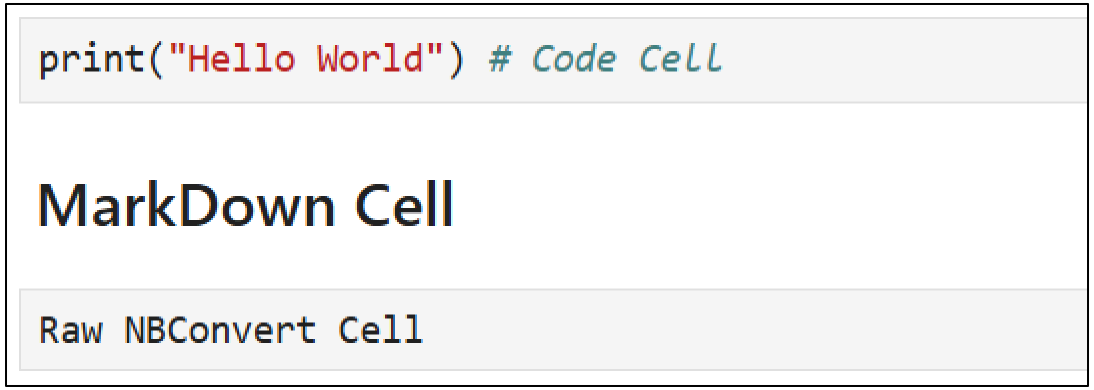
Train, validate, and test a Perceptron on the MNIST dataset. Use a learning rate of 0.025 and a max iteration of 5000

```
#--------
# Your code here
```

Jupyter Cell Types Introduction

- To build the labs, practices, and exercises needed for a given module, I generally utilize multiple types of Jupyter Cells:
  - Code Cells
  - Markdown Cells
  - Raw NBConvert Cells

- Each cell has a usefulness for building instructional materials in AI/ML coursework



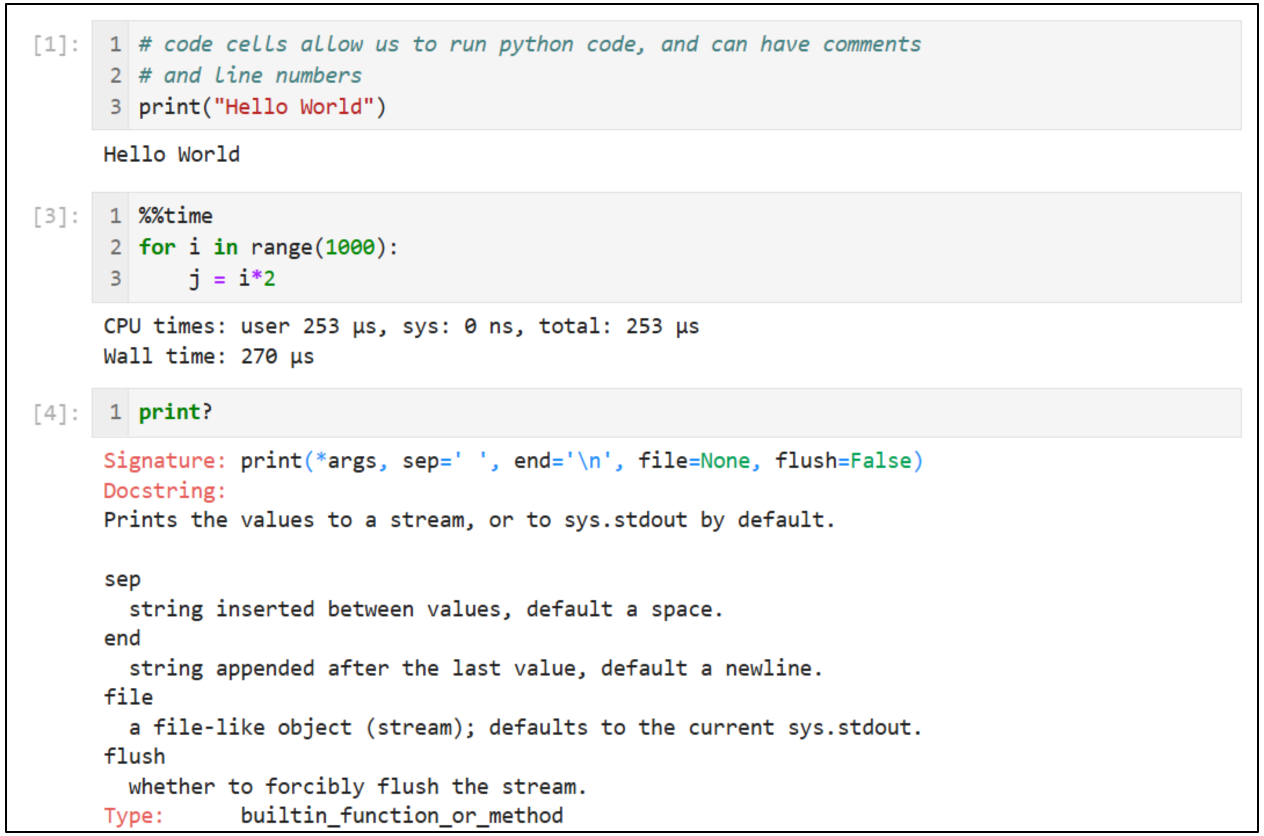
```
print("Hello World") # Code Cell
```

MarkDown Cell

```
Raw NBConvert Cell
```

Jupyter Cell Types: Code

- Code cells are the bread-and-butter of Jupyter Notebook Lessons
- They are the “hands-on” element of Jupyter Notebook lessons
  - **Labs** use code cells for fully worked examples and to guide students through practical usage and application of concepts in Python (or other languages)
  - **Practices** use code cells for the partially worked out examples and for students to fill in the missing pieces
  - **Exercises** use code cells as the answer block for most of the questions
- Code cells have full functionality of Python3, and some particularly helpful features for coursework include:
  - Code Completion
  - Line Numbers
  - Block and Line Commenting
  - IPython Magic: Automatic Runtime, Package Management, **Automatic Documentation**



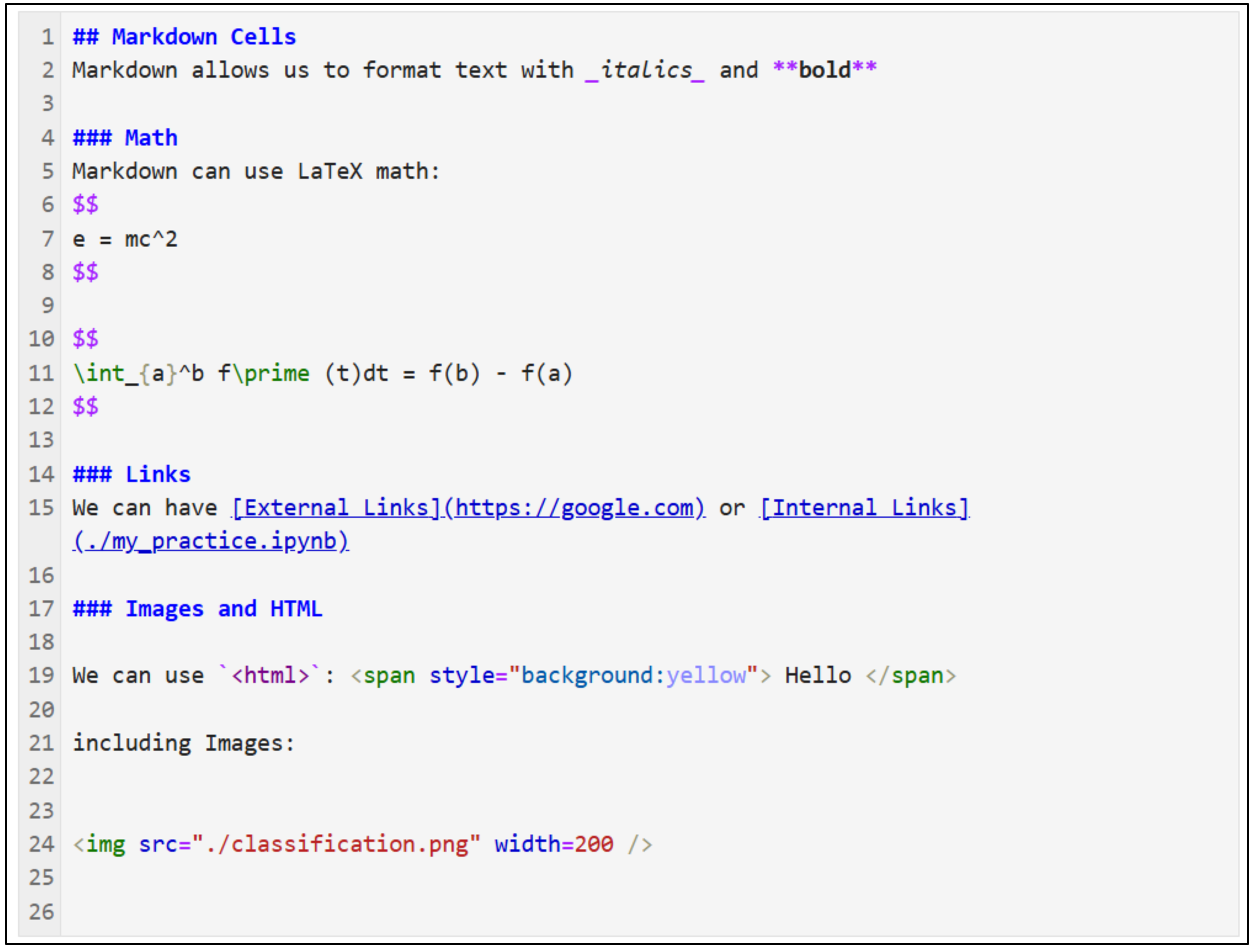
```
[1]: # code cells allow us to run python code, and can have comments
     # and line numbers
     print("Hello World")

Hello World
```

```
[3]: %%time
     for i in range(1000):
         j = i*2

CPU times: user 253 µs, sys: 0 ns, total: 253 µs
Wall time: 270 µs
```

```
[4]: print?

Signature: print(*args, sep=' ', end='\n', file=None, flush=False)
Docstring:
Prints the values to a stream, or to sys.stdout by default.

sep
  string inserted between values, default a space.
end
  string appended after the last value, default a newline.
file
  a file-like object (stream); defaults to the current sys.stdout.
flush
  whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

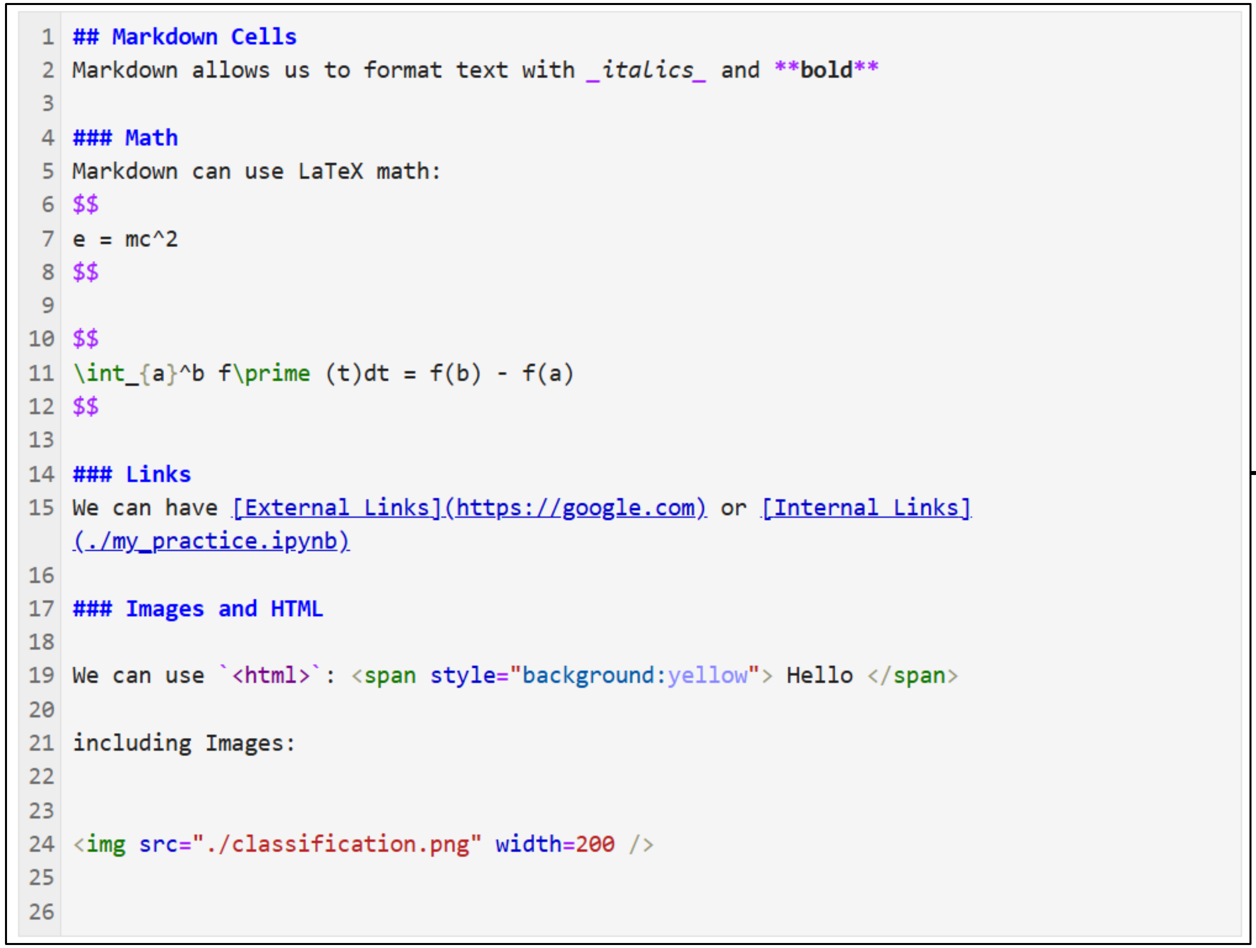
Jupyter Cell Types: Markdown

- Markdown cells are the “explanatory” and textual component of Jupyter Notebook Lessons
- We can use Markdown between and among code cells to split notebooks into sub-regions, add headers and separators, and add additional elements to build **standalone lessons inside of Jupyter**
- Markdown cells have full compatibility with external Markdown renderers, such as GitHub and other documentation. Some particularly helpful features include:
    - Text formatting
    - Math via LaTex
    - Links, both internal and external
    - HTML Support
    - Code Block Inlays

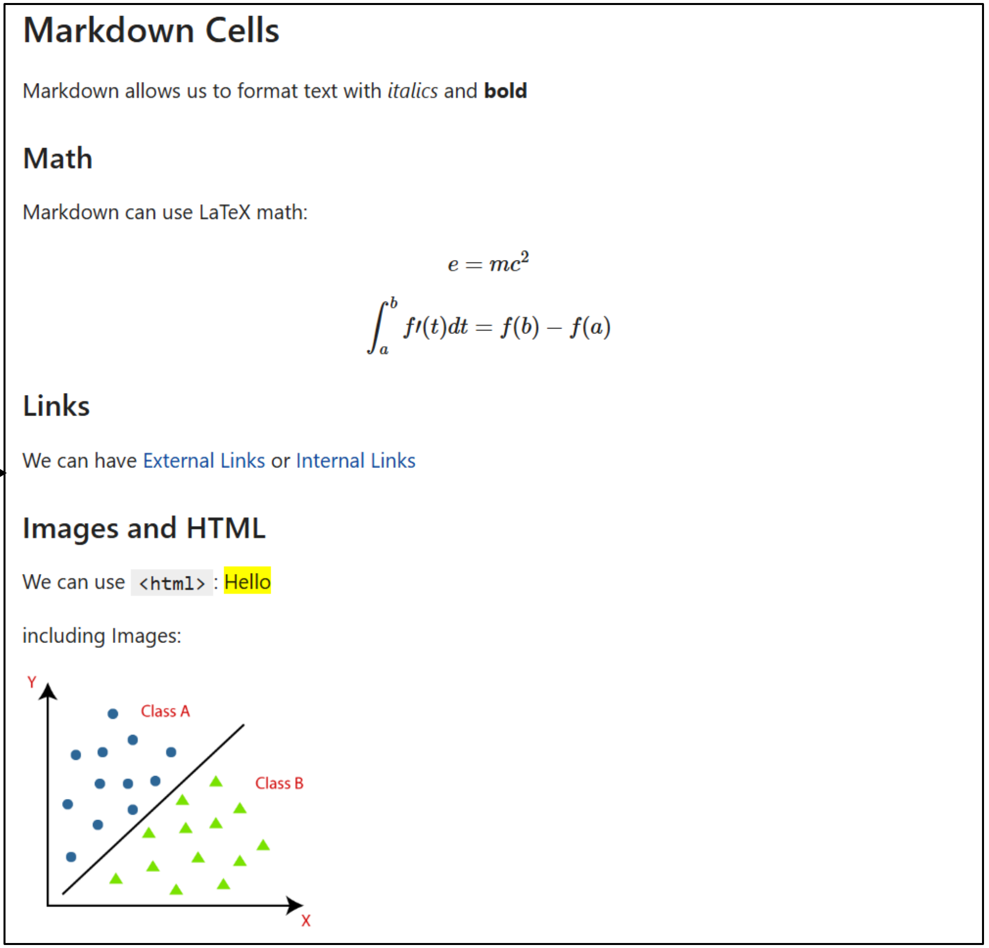


```
## Markdown Cells
Markdown allows us to format text with _italics_ and **bold**

### Math
Markdown can use LaTeX math:
$$
e = mc^2
$$

$$
\int_{a}^b f\prime (t)dt = f(b) - f(a)
$$

### Links
We can have [External Links](https://google.com) or [Internal Links](./my_practice.ipynb)

### Images and HTML

We can use `<html>`: <span style="background:yellow"> Hello </span>

including Images:


<img src="./classification.png" width=200 />


```

Jupyter Cell Types: Markdown



```
## Markdown Cells
Markdown allows us to format text with _italics_ and **bold**

### Math
Markdown can use LaTeX math:
$$
e = mc^2
$$

$$
\int_{a}^b f\prime (t)dt = f(b) - f(a)
$$

### Links
We can have [External Links](https://google.com) or [Internal Links](./my_practice.ipynb)

### Images and HTML

We can use `<html>`: <span style="background:yellow"> Hello </span>

including Images:


<img src="./classification.png" width=200 />


```



Markdown Cells

Markdown allows us to format text with *italics* and **bold**

Math

Markdown can use LaTeX math:

$$e = mc^2$$

$$\int_a^b f\prime(t)dt = f(b) - f(a)$$

Links

We can have External Links or Internal Links

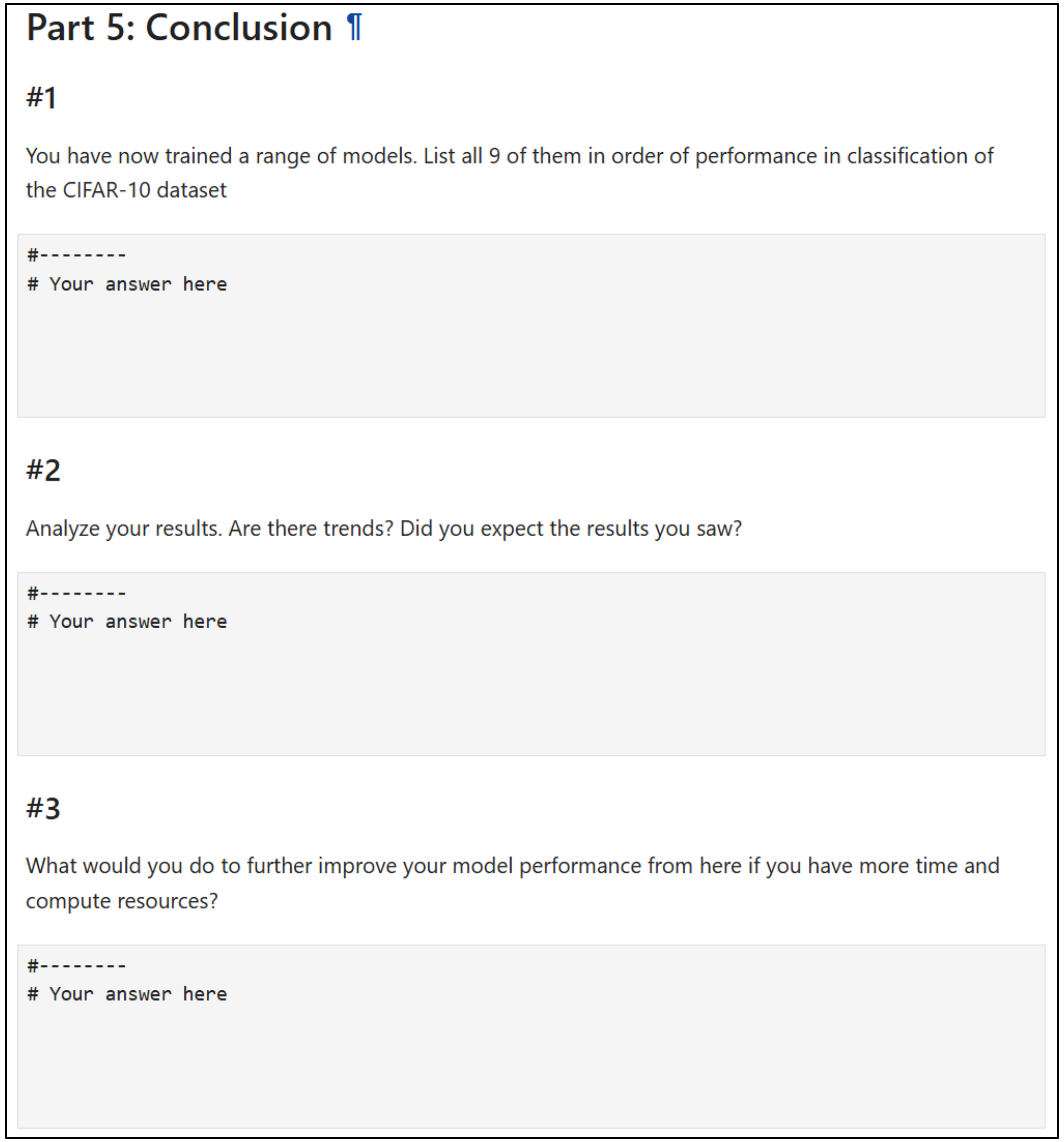
Images and HTML

We can use `<html>`: Hello

including Images:

Jupyter Cell Types: Raw NBConvert

- Raw NBConvert cells are non-formatted, non-code cells
- Raw NBConvert are generally used as “enter your answer that is not code here”
    - Often used after a code cell that produces results to have students use their own words
- Good questions for Raw NBConvert cells are hypothetical questions:
    - How would you go about improving performance if given additional resources?
    - How do you think this model would perform on out-of-distribution data? Why?
- Much less used than cell or markdown cells



Part 5: Conclusion ¶

#1

You have now trained a range of models. List all 9 of them in order of performance in classification of the CIFAR-10 dataset

```
#--------
# Your answer here
```

#2

Analyze your results. Are there trends? Did you expect the results you saw?

```
#--------
# Your answer here
```

#3

What would you do to further improve your model performance from here if you have more time and compute resources?

```
#--------
# Your answer here
```

Thank you!

Questions?

Hands-On Example:

Jupyter Instance: gp-engine.nrp-nautilus.io

Code: github.com/MUAMLL/CourseworkTutorial

Resources

- NRP Homepage: https://nationalresearchplatform.org/
- NRP Documentation: https://ucsd-prp.gitlab.io/
- MU Resources for AI/ML Researchers using NRP: https://github.com/MUAMLL/nautilus
  - Sample Dockerfiles, YAMLs, and Wiki
  - Deploying JupyterHub on NRP: https://github.com/MUAMLL/nautilus/wiki/JupyterHub
- GP-ENGINE Tutorial Resources: https://github.com/MUAMLL/gp-engine-tutorials
  - Step-by-step for getting started using NRP including account creation, PVC setup, and basic pod and job running
- Matrix Chat: https://nationalresearchplatform.org/updates/matrix-chat-for-nautilus-users/
  - Chat-service (similar to slack) for discussion and help using NRP