Machine Learning Engineer Nanodegree Material

Table of Contents

Machine Learning Engineer Nanodegree Material

Machine Learning Engineer Nanodegree Material

Courses and Syllabus

This program is all about advanced machine learning techniques used in the industry. Here is an overview of what you can expect as you move through the classroom.

The content is divided into modules listed below

1. Welcome to the Nanodegree.
2. Software Engineering Fundamentals
3. Machine Learning in Production
4. Machine Learning Deployment Case Studies
5. Build Your Own Machine Learning Portfolio (Capstone) Project.

Software Engineering Fundamentals

First, we'll start by teaching you some best practices for software engineering. You'll learn how to optimize your code, write tests and documentation for a code base, and build a Python package of your own. These skills are valuable in any engineering job and will act as a great foundation for applying machine learning skills in industry.

Machine Learning in Production

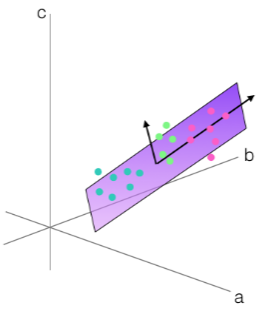
In this section, you'll learn about cloud services and model deployment. Deployment means making a model available for use in a piece of hardware or web application, such as a voice assistant or recommendation engine. In this lesson, you'll see how to analyze housing data and deploy a predictive model in SageMaker. In addition to learning about model deployment, you'll also learn about model serving and updating. You'll learn how to connect a deployed model to a website through an API using AWS services. After deploying the model, you'll update the model to account for changes in the underlying text data—an especially valuable skill in industries that continuously collect data. By the end of

this section, you should have all the skills you need to train and deploy models to solve tasks of your own design!

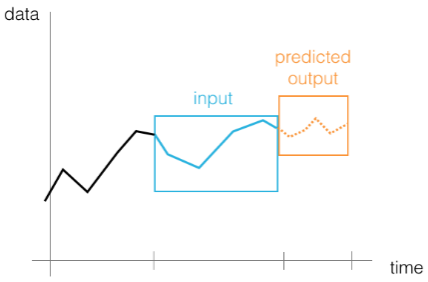
Machine Learning, Deployment Case Studies

In partnership with AWS, we've created a course that examines a variety of machine learning models as they are applied, at-scale, to real-world tasks. You'll learn how to deploy both unsupervised and supervised algorithms and apply them to tasks such as feature engineering and time series forecasting. This content addresses questions such as:

1. How do you decide on the correct machine learning model for a given task?
2. How can you utilize cloud deployment tools such as AWS SageMaker to work with data and improve your machine learning models?



Clustering & Dimensionality Reduction



Time Series Forecasting

Capstone Project

This section has two phases. The first is the Capstone Proposal, during which you will draft a proposal outlining the domain of the problem you would like to solve, and your approach. This is followed by the Capstone Project: here, you will leverage your newly-learned skills to solve the problem—as outlined in your proposal—by applying machine learning algorithms and techniques.

Email Support

If you have a question about enrollment, payment, or a classroom or video glitch, email **machine-support@udacity.com**.

Machine Learning Careers

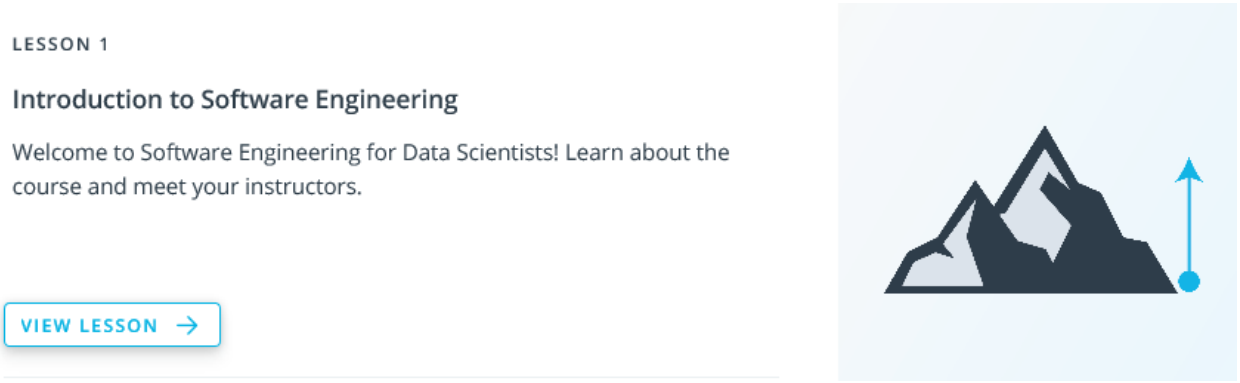
Over the past few years, the demand for machine learning specialists and engineers has soared, with Machine Learning Engineers and Specialists ranking among the top **emerging jobs on LinkedIn**. Recently, machine learning has been adopted by a wide range of industries, including medical diagnostic companies, finance and market prediction firms, and many more. Udacity's Machine Learning Nanodegree program was built in response to this demand to provide access to this growing tech field to a broader audience.

We've seen advances in research and industry practices as more companies look to build machine learning products. **Specifically, there is a growing demand for engineers who are able to deploy machine learning models to global audiences**. Also, machine learning engineers are often expected to work within larger software development teams. To keep up with this advancement and bring the best educational experience to our students, we have included content that includes:

- Best practices for software engineering, and
- The latest deployment skills, including deploying models at scale and updating models in response to changes in the underlying data.

These skills will set you apart from engineers who know only machine learning algorithms, but not the details of machine learning product development, at scale. So, I hope you're excited to learn even more!

Software Engineering Fundamentals



Lesson 1:

Welcome! https://www.youtube.com/watch?v=7kphieW4yl4&feature=emb_logo

How this Course is Organized

- Software Engineering Practices Part 1 covers how to write well documented, modularized code.
- Software Engineering Practices Part 2 discusses testing your code and logging.
- Introduction to Object-Oriented Programming gives you an overview of this programming style and prepares you to write your own Python package.
- Introduction to Web Development covers building a web application data dashboard.

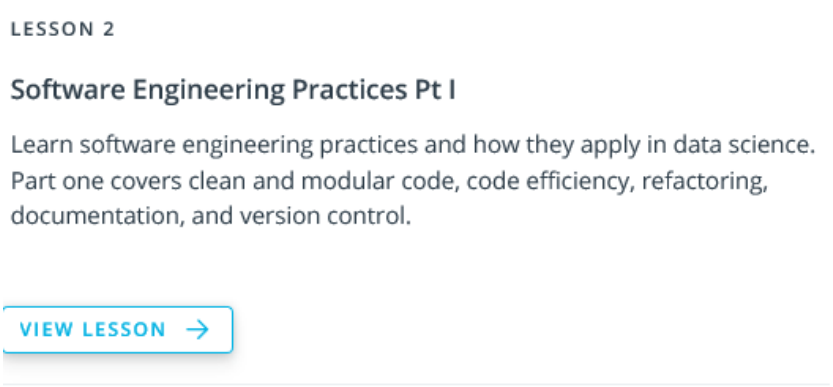
Course Portfolio Exercises

The software engineering course has two portfolio exercises: building a Python package and developing a web data dashboard. These exercises are **NOT** reviewed and are

Machine Learning Engineer Nanodegree Material

NOT required to graduate from the data scientist nanodegree program. In other words, you will not submit either of the portfolio projects to the Udacity review system. Instead, you can use these projects to practice your software engineering skills and then add the projects to your professional portfolio.
Having said that, the skills covered in this course will set you up for success in other Udacity courses with required projects. For example, the data engineering for data scientists course has a required project where you are expected to write clean, concise and well-documented code. You will also have an easier time with that project if you understand the fundamentals of object-oriented programming and a basic understanding of how the backend and frontend of a website works.

https://www.youtube.com/watch?v=v-DB0W_I2n8&feature=emb_logo



Lesson 2:

Intro:

https://www.youtube.com/watch?time_continue=3&v=z7v7oa--W48&feature=emb_logo

Welcome to Software Engineering Practices Part I

In this lesson, you'll learn about the following practices of software engineering and how they apply in data science.

- Writing clean and modular code
- Writing efficient code
- Code refactoring
- Adding meaningful documentation
- Using version control

In the lesson following this one (Part II) you'll also learn these software engineering practices:

- Testing
- Logging
- Code reviews

Clean and Modular code:

https://www.youtube.com/watch?v=RjHV8kRpVbA&feature=emb_logo

- **PRODUCTION CODE:** software running on production servers to handle live users and data of the intended audience. Note this is different from *production quality code*, which describes code that meets expectations in reliability, efficiency, etc., for production. Ideally, all code in production meets these expectations, but this is not always the case.
- **CLEAN:** readable, simple, and concise. A characteristic of production quality code that is crucial for collaboration and maintainability in software development.
- **MODULAR:** logically broken up into functions and modules. Also an important characteristic of production quality code that makes your code more organized, efficient, and reusable.
- **MODULE:** a file. Modules allow code to be reused by encapsulating them into files that can be imported into other files.

Clean Mod Code:

https://www.youtube.com/watch?time_continue=1&v=9bxtHpPvXE0&feature=emb_logo

Writing Clean Code: Meaningful Names

Tip: Use meaningful names

- **Be descriptive and imply type** - E.g. for booleans, you can prefix with is_ or has_ to make it clear it is a condition. You can also use part of speech to imply types, like verbs for functions and nouns for variables.
- **Be consistent but clearly differentiate** - E.g. age_list and age is easier to differentiate than ages and age.
- **Avoid abbreviations and especially single letters** - (Exception: counters and common math variables) Choosing when these exceptions can be made can be determined based on the audience for your code. If you work with other data scientists, certain variables may be common knowledge. While if you work with full stack engineers, it might be necessary to provide more descriptive names in these cases as well.
- **Long names != descriptive names** - You should be descriptive, but only with relevant information. E.g. good functions names describe what they do well without including details about implementation or highly specific uses.

Try testing how effective your names are by asking a fellow programmer to guess the purpose of a function or variable based on its name, without looking at your code. Coming up with meaningful names often requires effort to get right.

Writing Clean Code: Nice Whitespace

Tip: Use whitespace properly

- Organize your code with consistent indentation - the standard is to use 4 spaces for each indent. You can make this a default in your text editor.
- Separate sections with blank lines to keep your code well organized and readable.
- Try to limit your lines to around 79 characters, which is the guideline given in the PEP 8 style guide. In many good text editors, there is a setting to display a subtle line that indicates where the 79 character limit is.

For more guidelines, check out the code layout section of PEP 8 in the notes below.

Writing Clean Code: Nice Whitespace

PEP 8 guidelines for code layout

https://www.youtube.com/watch?v=wNaiahWCwkQ&feature=emb_logo

Writing Modular Code

Tip: DRY (Don't Repeat Yourself)

Don't repeat yourself! Modularization allows you to reuse parts of your code. Generalize and consolidate repeated code in functions or loops.

Tip: Abstract out logic to improve readability

Abstracting out code into a function not only makes it less repetitive, but also improves readability with descriptive function names. Although your code can become more readable when you abstract out logic into functions, it is possible to over-engineer this and have way too many modules, so use your judgement.

Tip: Minimize the number of entities (functions, classes, modules, etc.)

There are tradeoffs to having function calls instead of inline logic. If you have broken up your code into an unnecessary amount of functions and modules, you'll have to jump around everywhere if you want to view the implementation details for something that may be too small to be worth it. Creating more modules doesn't necessarily result in effective modularization.

Tip: Functions should do one thing

Each function you write should be focused on doing one thing. If a function is doing multiple things, it becomes more difficult to generalize and reuse. Generally, if there's an "and" in your function name, consider refactoring.

Tip: Arbitrary variable names can be more effective in certain functions

Arbitrary variable names in general functions can actually make the code more readable.

Tip: Try to use fewer than three arguments per function

Try to use no more than three arguments when possible. This is not a hard rule and there are times it is more appropriate to use many parameters. But in many cases, it's more effective to use fewer arguments. Remember we are modularizing to simplify our code and make it more efficient to work with. If your function has a lot of parameters, you may want to rethink how you are splitting this up.

https://www.youtube.com/watch?time_continue=23&v=qN6EOyNlSnk&feature=emb_logo

Efficient Code

Knowing how to write code that runs efficiently is another essential skill in software development. Optimizing code to be more efficient can mean making it:

- Execute faster
- Take up less space in memory/storage

The project you're working on would determine which of these is more important to optimize for your company or product. When we are performing lots of different transformations on large amounts of data, this can make orders of magnitudes of difference in performance.

https://www.youtube.com/watch?time_continue=13&v=LbtxY7xetBw&feature=emb_logo

- Notebook: Wine_Quality_Sol

Optimizing - Common Books

Resources:

- **What makes sets faster than lists**

https://www.youtube.com/watch?time_continue=85&v=WF9n_19V08g&feature=emb_logo

- Notebook: Holiday gifts

Documentation

- **DOCUMENTATION:** additional text or illustrated information that comes with or is embedded in the code of software.
- Helpful for clarifying complex parts of code, making your code easier to navigate, and quickly conveying how and why different components of your program are used.
- Several types of documentation can be added at different levels of your program:
- **In-line Comments** - line level
- **Docstrings** - module and function level
- **Project Documentation** - project level

https://www.youtube.com/watch?time_continue=10&v=M45B2VbPgjo&feature=emb_logo

Docstrings

https://www.youtube.com/watch?time_continue=11&v=_gapemxsRJY&feature=emb_logo

Project Documentation

Project documentation is essential for getting others to understand why and how your code is relevant to them, whether they are potentials users of your project or developers who may contribute to your code. A great first step in project documentation is your README file. It will often be the first interaction most users will have with your project.

Whether it's an application or a package, your project should absolutely come with a README file. At a minimum, this should explain what it does, list its dependencies, and provide sufficiently detailed instructions on how to use it. You

want to make it as simple as possible for others to understand the purpose of your project, and quickly get something working.

Translating all your ideas and thoughts formally on paper can be a little difficult, but you'll get better over time and makes a significant difference in helping others realize the value of your project. Writing this documentation can also help you improve the design of your code, as you're forced to think through your design decisions more thoroughly. This also allows future contributors to know how to follow your original intentions.

A full Udacity course on this topic can be found **here**.

Here are a few READMEs from some popular projects:

- **Bootstrap**
- **Scikit-learn**
- **Stack Overflow Blog**

Version Control in Data Science

If you need a refresher on using git for version control, check out the course linked in the extracurriculars. If you're ready, let's see how git is used in real data science scenarios!

https://www.youtube.com/watch?time_continue=40&v=EQzrLC88Bzk&feature=emb_logo

Scenario #1

Let's walk through the git commands that go along with each step in the scenario you just observed in the video above.

STEP 1: You have a local version of this repository on your laptop, and to get the latest stable version, you pull from the develop branch.

Switch to the develop branch

```
git checkout develop
```

Pull latest changes in the develop branch

```
git pull
```

STEP 2: When you start working on this demographic feature, you create a new branch for this called demographic, and start working on your code in this branch.

Create and switch to new branch called demographic from develop branch

```
git checkout -b demographic
```

Work on this new feature and commit as you go

```
git commit -m 'added gender recommendations'
```

```
git commit -m 'added location specific recommendations'
```

```
...
```

STEP 3: However, in the middle of your work, you need to work on another feature. So you commit your changes on this demographic branch, and switch back to the develop branch.

Commit changes before switching

```
git commit -m 'refactored demographic gender and location recommendations '
```

Switch to the develop branch

```
git checkout develop
```

STEP 4: From this stable develop branch, you create another branch for a new feature called friend_groups.

Create and switch to new branch called friend_groups from develop branch

```
git checkout -b friend_groups
```

STEP 5: After you finish your work on the friend_groups branch, you commit your changes, switch back to the development branch, merge it back to the develop branch, and push this to the remote repository's develop branch.

Commit changes before switching

```
git commit -m 'finalized friend_groups recommendations '
```

Switch to the develop branch

```
git checkout develop
```

Merge friend_groups branch to develop

```
git merge --no-ff friends_groups
```

Push to remote repository

```
git push origin develop
```

STEP 6: Now, you can switch back to the demographic branch to continue your progress on that feature.

Switch to the demographic branch

```
git checkout demographic
```

https://www.youtube.com/watch?v=C92YcuwjZOs&feature=emb_logo

Scenario #2

Let's walk through the git commands that go along with each step in the scenario you just observed in the video above.

Step 1: You check your commit history, seeing messages of the changes you made and how well it performed.

View log history

```
git log
```

Step 2: The model at this commit seemed to score the highest, so you decide to take a look.

Checkout a commit

```
git checkout bc90f2cbc9dc4e802b46e7a153aa106dc9a88560
```

After inspecting your code, you realize what modifications made this perform well, and use those for your model.

Step 3: Now, you're pretty confident merging this back into the development branch, and pushing the updated recommendation engine.

Switch to develop branch

```
git checkout develop
```

Merge friend_groups branch to develop

```
git merge --no-ff friend_groups
```

Push changes to remote repository

```
git push origin develop
```

https://www.youtube.com/watch?time_continue=17&v=w1iHWpwOkMg&feature=emb_logo

Scenario #3

Let's walk through the git commands that go along with each step in the scenario you just observed in the video above.

Step 1: Andrew commits his changes to the documentation branch, switches to the development branch, and pulls down the latest changes from the cloud on this development branch, including the change I merged previously for the friends group feature.

Commit changes on documentation branch

git commit -m "standardized all docstrings in process.py"

Switch to develop branch

git checkout develop

Pull latest changes on develop down

git pull

Step 2: Then, Andrew merges his documentation branch on the develop branch on his local repository, and then pushes his changes up to update the develop branch on the remote repository.

Merge documentation branch to develop

git merge --no-ff documentation

Push changes up to remote repository

git push origin develop

Step 3: After the team reviewed both of your work, they merge the updates from the development branch to the master branch. Now they push the changes to the master branch on the remote repository. These changes are now in production.

Merge develop to master

git merge --no-ff develop

Push changes up to remote repository

git push origin master

Resources

There's a great article on a successful git branching strategy that you should really read **here**.

Note on Merge Conflicts

For the most part, git makes merging changes between branches really simple. However, there are some cases where git will be confused on how to combine two changes, and asks you for help. This is called a merge conflict.

Mostly commonly, this happens when two branches modify the same file.

For example, in this situation, let's say I deleted a line that Andrew modified on his branch. Git wouldn't know whether to delete the line or modify it. Here, you need to tell git which change to take, and some tools even allow you to edit the change manually. If it isn't straightforward, you may have to consult with the developer of the other branch to handle a merge conflict.

You can learn more about merge conflicts and methods to handle them **here**.

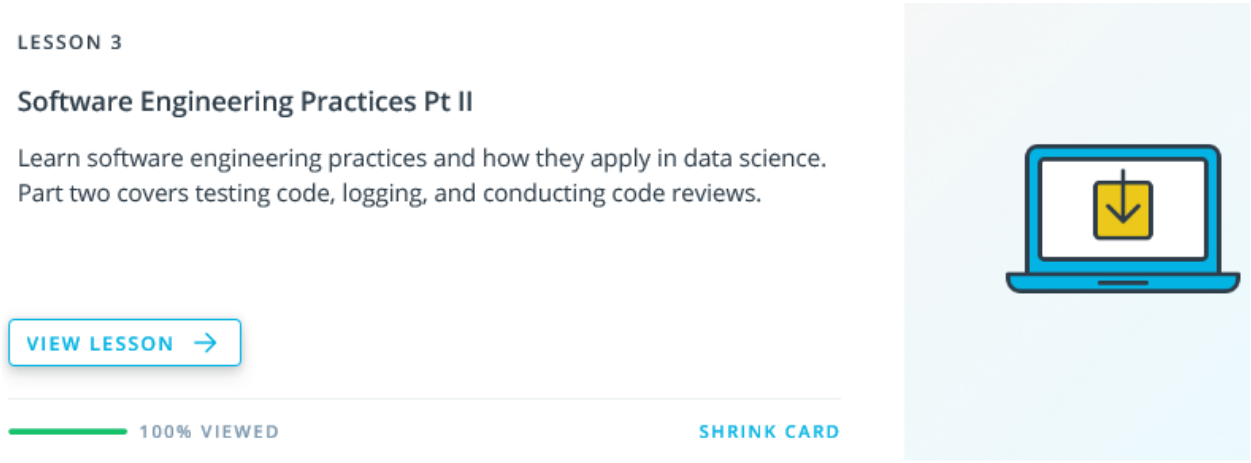
https://www.youtube.com/watch?time_continue=67&v=36DOnNzvT4A&feature=emb_logo

Model Versioning

In the previous example, you may have noticed that each commit was documented with a score for that model. This is one simple way to help you keep track of model versions. Version control in data science can be tricky, because there are many pieces involved that can be hard to track, such as large amounts of data, model versions, seeds, hyperparameters, etc.

Here are some resources for useful ways and tools for managing versions of models and large data. These are here for you to explore, but are not necessary to know now as you start your journey as a data scientist. On the job, you'll always be learning new skills, and many of them will be specific to the processes set in your company.

- How to Version Control Your Production Machine Learning Models
- Versioning Data Science



Lesson 3: Software Engineer Practices pt 2

Welcome to Software Engineering Practices Part II

In part 2 of software engineering practices, you'll learn about the following practices of software engineering and how they apply in data science.

- Testing
- Logging
- Code reviews

Testing

Testing your code is essential before deployment. It helps you catch errors and faulty conclusions before they make any major impact. Today, employers are looking for data scientists with the skills to properly prepare their code for an industry setting, which includes testing their code.

https://www.youtube.com/watch?time_continue=18&v=IkLUUHt_jis&feature=emb_logo

Testing and Data Science

- Problems that could occur in data science aren't always easily detectable; you might have values being encoded incorrectly, features being used inappropriately, unexpected data breaking assumptions
- To catch these errors, you have to check for the quality and accuracy of your *analysis* in addition to the quality of your *code*. Proper testing is necessary to avoid unexpected surprises and have confidence in your results.
- **TEST DRIVEN DEVELOPMENT:** a development process where you write tests for tasks before you even write the code to implement those tasks.
- **UNIT TEST:** a type of test that covers a "unit" of code, usually a single function, independently from the rest of the program.

Resources:

- Four Ways Data Science Goes Wrong and How Test Driven Data Analysis Can Help: **Blog Post**
- Ned Batchelder: Getting Started Testing: **Slide Deck** and **Presentation Video**

https://www.youtube.com/watch?time_continue=21&v=AsnstNEMv1c&feature=emb_logo

Unit Tests

We want to test our functions in a way that is repeatable and automated. Ideally, we'd run a test program that runs all our unit tests and cleanly lets us know which ones failed and which ones succeeded. Fortunately, there are great tools available in Python that we can use to create effective unit tests!

Unit Test Advantages and Disadvantages

The advantage of unit tests is that they are isolated from the rest of your program, and thus, no dependencies are involved. They don't require access to databases, APIs, or other external sources of information. However, passing unit tests isn't always enough to prove that our program is working successfully. To show that all the parts of our program work with each other properly, communicating and transferring data between them correctly, we use integration tests. In this lesson, we'll focus on unit tests; however, when you start building larger programs, you will want to use integration tests as well.

You can read about integration testing and how integration tests relate to unit tests **here**. That article contains other very useful links as well.

https://www.youtube.com/watch?v=wb9jggHEvgI&feature=emb_logo

Unit Testing Tools

To install pytest, run pip install -U pytest in your terminal. You can see more information on getting started **here**.

- Create a test file starting with test_
- Define unit test functions that start with test_ inside the test file
- Enter pytest into your terminal in the directory of your test file and it will detect these tests for you!

test_ is the default - if you wish to change this, you can learn how to in this **pytest configuration**

In the test output, periods represent successful unit tests and F's represent failed unit tests. Since all you see is what test functions failed, it's wise to have only one assert statement per test. Otherwise, you wouldn't know exactly how many tests failed, and which tests failed.

Your tests won't be stopped by failed assert statements, but it will stop if you have syntax errors.

https://www.youtube.com/watch?time_continue=8&v=8bKhOyFbX_Y&feature=emb_logo

Test Driven Development and Data Science

- **TEST DRIVEN DEVELOPMENT:** writing tests before you write the code that's being tested. Your test would fail at first, and you'll know you've finished implementing a task when this test passes.
- Tests can check for all the different scenarios and edge cases you can think of, before even starting to write your function. This way, when you do start implementing your function, you can run this test to get immediate feedback on whether it works or not in all the ways you can think of, as you tweak your function.
- When refactoring or adding to your code, tests help you rest assured that the rest of your code didn't break while you were making those changes. Tests also helps ensure that your function behavior is repeatable, regardless of external parameters, such as hardware and time.

Test driven development for data science is relatively new and has a lot of experimentation and breakthroughs appearing, which you can learn more about in the resources below.

- **Data Science TDD**
- **TDD for Data Science**
- **TDD is Essential for Good Data Science Here's Why**
- **Testing Your Code** (general python TDD)

https://www.youtube.com/watch?v=M-eskssLcQM&feature=emb_logo

Logging

Logging is valuable for understanding the events that occur while running your program. For example, if you run your model over night and see that it's producing ridiculous results the next day, log messages can really help you understand more about the context in which this occurred. Lets learn about the qualities that make a log message effective.

https://www.youtube.com/watch?time_continue=9&v=9qKQdRoIMbU&feature=emb_logo

Log Messages

Logging is the process of recording messages to describe events that have occurred while running your software. Let's take a look at a few examples, and learn tips for writing good log messages.

Tip: Be professional and clear

```
Bad: Hmmm... this isn't working???
Bad: idk.... :(
Good: Couldn't parse file.
```

Tip: Be concise and use normal capitalization

```
Bad: Start Product Recommendation Process
Bad: We have completed the steps necessary and will now proceed with the recommendati
on process for the records in our product database.
Good: Generating product recommendations.
```

Tip: Choose the appropriate level for logging

DEBUG - level you would use for anything that happens in the program.
ERROR - level to record any error that occurs
INFO - level to record all actions that are user-driven or system specific, such as regularly scheduled operations

Tip: Provide any useful information

```
Bad: Failed to read location data
Good: Failed to read location data: store_id 8324971
```

Code Reviews

Code reviews benefit everyone in a team to promote best programming practices and prepare code for production. Let's go over what to look for in a code review and some tips on how to conduct one.

- **Code Review**
- **Code Review Best Practices**

https://www.youtube.com/watch?v=zAy1ffMFA-k&feature=emb_logo

Questions to Ask Yourself When Conducting a Code Review

First, let's look over some of the questions we may ask ourselves while reviewing code. These are simply from the concepts we've covered in these last two lessons!

Is the code clean and modular?

- Can I understand the code easily?
- Does it use meaningful names and whitespace?

- Is there duplicated code?
- Can you provide another layer of abstraction?
- Is each function and module necessary?
- Is each function or module too long?

Is the code efficient?

- Are there loops or other steps we can vectorize?
- Can we use better data structures to optimize any steps?
- Can we shorten the number of calculations needed for any steps?
- Can we use generators or multiprocessing to optimize any steps?

Is documentation effective?

- Are in-line comments concise and meaningful?
- Is there complex code that's missing documentation?
- Do function use effective docstrings?
- Is the necessary project documentation provided?

Is the code well tested?

- Does the code high test coverage?
- Do tests check for interesting cases?
- Are the tests readable?
- Can the tests be made more efficient?

Is the logging effective?

- Are log messages clear, concise, and professional?
- Do they include all relevant and useful information?
- Do they use the appropriate logging level?

Tips for Conducting a Code Review

Now that we know what we are looking for, let's go over some tips on how to actually write your code review. When your coworker finishes up some code that they want to

merge to the team's code base, they might send it to you for review. You provide feedback and suggestions, and then they may make changes and send it back to you. When you are happy with the code, you approve and it gets merged to the team's code base.

As you may have noticed, with code reviews you are now dealing with people, not just computers. So it's important to be thoughtful of their ideas and efforts. You are in a team and there will be differences in preferences. The goal of code review isn't to make all code follow your personal preferences, but a standard of quality for the whole team.

Tip: Use a code linter

This isn't really a tip for code review, but can save you lots of time from code review! Using a Python code linter like **pylint** can automatically check for coding standards and PEP 8 guidelines for you! It's also a good idea to agree on a style guide as a team to handle disagreements on code style, whether that's an existing style guide or one you create together incrementally as a team.

Tip: Explain issues and make suggestions

Rather than commanding people to change their code a specific way because it's better, it will go a long way to explain to them the consequences of the current code and *suggest* changes to improve it. They will be much more receptive to your feedback if they understand your thought process and are accepting recommendations, rather than following commands. They also may have done it a certain way intentionally, and framing it as a suggestion promotes a constructive discussion, rather than opposition.

```
BAD: Make model evaluation code its own module - too repetitive.

BETTER: Make the model evaluation code its own module. This will simplify models.py t
o be less repetitive and focus primarily on building models.

GOOD: How about we consider making the model evaluation code its own module? This wou
ld simplify models.py to only include code for building models. Organizing these eval
uations methods into separate functions would also allow us to reuse them with differ
ent models without repeating code.
```

Tip: Keep your comments objective

Try to avoid using the words "I" and "you" in your comments. You want to avoid comments that sound personal to bring the attention of the review to the code and not to themselves.

```
BAD: I wouldn't groupby genre twice like you did here... Just compute it once and use
that for your aggregations.

BAD: You create this groupby dataframe twice here. Just compute it once, save it as g
roupby_genre and then use that to get your average prices and views.

GOOD: Can we group by genre at the beginning of the function and then save that as a
groupby object? We could then reference that object to get the average prices and vie
ws without computing groupby twice.
```

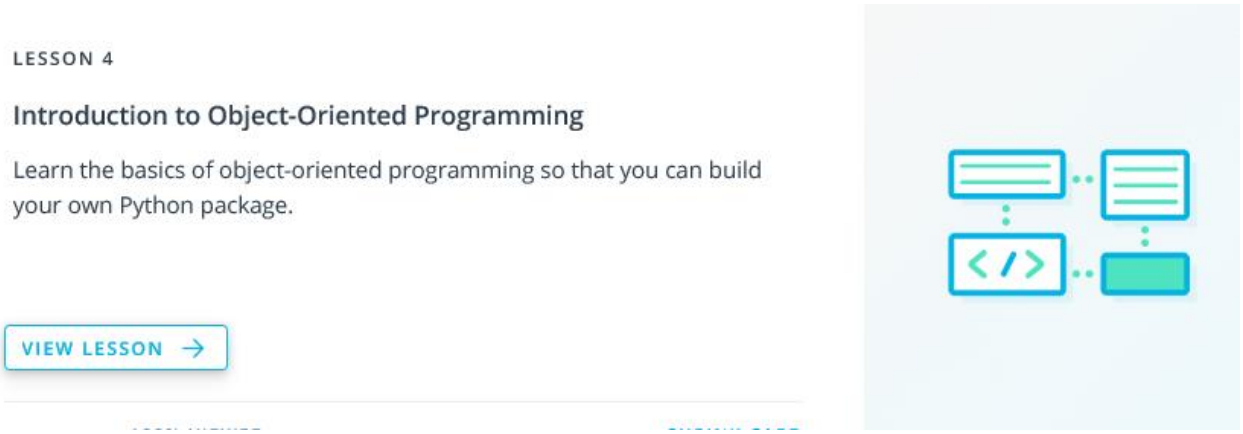
Tip: Provide code examples

When providing a code review, you can save the author time and make it easy for them to act on your feedback by writing out your code suggestions. This shows you are willing to spend some extra time to review their code and help them out. It can also just be much quicker for you to demonstrate concepts through code rather than explanations.

Let's say you were reviewing code that included the following lines:

```
first_names = []
last_names = []

for name in enumerate(df.name):
    first, last = name.split(' ')
    first_names.append(first)
    last_names.append(last)

df['first_name'] = first_names
df['last_names'] = last_names
BAD: You can do this all in one step by using the pandas str.split method.
GOOD: We can actually simplify this step to the line below using the pandas str.split
method. Found this on this stack overflow post: https://stackoverflow.com/questions/1
4745022/how-to-split-a-column-into-two-columns
```

```
df['first_name'], df['last_name'] = df['name'].str.split(' ', 1).str
```



Lesson 4: Introduction to OOP

Lesson Outline

- Object-oriented programming syntax
- procedural vs object-oriented programming
- classes, objects, methods and attributes
- coding a class
- magic methods
- inheritance
- Using object-oriented programming to make a Python package
- making a package
- tour of scikit-learn source code
- putting your package on PyPi

Why Object-Oriented Programming?

Object-oriented programming has a few benefits over procedural programming, which is the programming style you most likely first learned. As you'll see in this lesson,

- object-oriented programming allows you to create large, modular programs that can easily expand over time;

- object-oriented programs hide the implementation from the end-user.

Consider Python packages like **Scikit-learn**, **pandas**, and **NumPy**. These are all Python packages built with object-oriented programming. Scikit-learn, for example, is a relatively large and complex package built with object-oriented programming. This package has expanded over the years with new functionality and new algorithms. When you train a machine learning algorithm with Scikit-learn, you don't have to know anything about how the algorithms work or how they were coded. You can focus directly on the modeling.

Here's an example taken from the **Scikit-learn website**:

```
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
```

How does Scikit-learn train the SVM model? You don't need to know because the implementation is hidden with object-oriented programming. If the implementation changes, you as a user of Scikit-learn might not ever find out. Whether or not you SHOULD understand how SVM works is a different question.

In this lesson, you'll practice the fundamentals of object-oriented programming. By the end of the lesson, you'll have built a Python package using object-oriented programming.

Lesson Files

This lesson uses classroom workspaces that contain all of the files and functionality you will need. You can also find the files in the **data scientist nanodegree term 2 GitHub repo**.

Procedural vs Object-Oriented Programming

Objects are defined by characteristics and actions

Here is a reminder of what is a characteristic and what is an action.

https://www.youtube.com/watch?time_continue=32&v=psXD_J8FnCQ&feature=emb_logo

Objects are defined by their characteristics and their actions

Characteristics and Actions in English Grammar

Another way to think about characteristics and actions is in terms of English grammar. A characteristic would be a noun. On the other hand, an action would be a verb.

Let's pick something from the real-world: a dog. A few characteristics could be the dog's weight, color, breed, and height. These are all nouns. What actions would a dog take? A dog can bark, run, bite and eat. These are all verbs.

Object-Oriented Programming (OOP) Vocabulary

- class - a blueprint consisting of methods and attributes
- object - an *instance* of a class. It can help to think of objects as something in the real world like a yellow pencil, a small dog, a blue shirt, etc. However, as you'll see later in the lesson, objects can be more abstract.
- attribute - a descriptor or characteristic. Examples would be color, length, size, etc. These attributes can take on specific values like blue, 3 inches, large, etc.
- method - an action that a class or object could take
- OOP - a commonly used abbreviation for object-oriented programming
- encapsulation - one of the fundamental ideas behind object-oriented programming is called encapsulation: you can combine functions and data all into a single entity. In object-oriented programming, this single entity is called a class. Encapsulation allows you to hide implementation details much like how the scikit-learn package hides the implementation of machine learning algorithms.

  In English, you might hear an attribute described as a *property*, *description*, *feature*, *quality*, *trait*, or *characteristic*. All of these are saying the same thing.

  Here is a reminder of how a class, object, attributes and methods relate to each other.

https://www.youtube.com/watch?v=yvVMJt09HuA&feature=emb_logo

Function vs Method

*In the video above at 1:44, the dialogue mistakenly calls **init** a function rather than a method. Why is **init** not a function?*

A function and a method look very similar. They both use the `def` keyword. They also have inputs and return outputs. The difference is that a method is inside of a class whereas a function is outside of a class.

What is self?

If you instantiate two objects, how does Python differentiate between these two objects?

```
shirt_one = Shirt('red', 'S', 'short-sleeve', 15)
short_two = Shirt('yellow', 'M', 'long-sleeve', 20)
```

That's where `self` comes into play. If you call the `change_price` method on shirt_one, how does Python know to change the price of shirt_one and not of shirt_two?

```
shirt_one.change_price(12)
```

Behind the scenes, Python is calling the `change_price` method:

```
    def change_price(self, new_price):

        self.price = new_price
```

`Self` tells Python where to look in the computer's memory for the shirt_one object. And then Python changes the price of the shirt_one object. When you call the `change_price` method, `shirt_one.change_price(12)`, `self` is implicitly passed in.

The word `self` is just a convention. You could actually use any other name as long as you are consistent; however, you should always use `self` rather than some other word or else you might confuse people.

https://www.youtube.com/watch?v=Y8ZVw1LHI8E&feature=emb_logo

Set and Get methods

The last part of the video mentioned that accessing attributes in Python can be somewhat different than in other programming languages like Java and C++. This section goes into further detail.

The shirt class has a method to change the price of the shirt: `shirt_one.change_price(20)`. In Python, you can also change the values of an attribute with the following syntax:

```
shirt_one.price = 10
shirt_one.price = 20
shirt_one.color = 'red'
shirt_one.size = 'M'
shirt_one.style = 'long_sleeve'
```

This code accesses and changes the price, color, size and style attributes directly. Accessing attributes directly would be frowned upon in many other languages **but not in Python**. Instead, the general object-oriented programming convention is to use methods to access attributes or change attribute values. These methods are called set and get methods or setter and getter methods.
A get method is for obtaining an attribute value. A set method is for changing an attribute value. If you were writing a Shirt class, the code could look like this:

```
class Shirt:

    def __init__(self, shirt_color, shirt_size, shirt_style, shirt_price):
        self._price = shirt_price

    def get_price(self):
      return self._price

    def set_price(self, new_price):
      self._price = new_price
```

Instantiating and using an object might look like this:

```
shirt_one = Shirt('yellow', 'M', 'long-sleeve', 15)
```

```
print(shirt_one.get_price())
shirt_one.set_price(10)
```

In the class definition, the underscore in front of price is a somewhat controversial Python convention. In other languages like C++ or Java, price could be explicitly labeled as a private variable. This would prohibit an object from accessing the price attribute directly like `shirt_one._price = 15`. However, Python does not distinguish between private and public variables like other languages. Therefore, there is some controversy about using the underscore convention as well as get and set methods in Python. Why use get and set methods in Python when Python wasn't designed to use them?
At the same time, you'll find that some Python programmers develop object-oriented programs using get and set methods anyway. Following the Python convention, the underscore in front of price is to let a programmer know that price should only be accessed with get and set methods rather than accessing price directly with `shirt_one._price`. However, a programmer could still access _price directly because there is nothing in the Python language to prevent the direct access.
To reiterate, a programmer could technically still do something like `shirt_one._price = 10`, and the code would work. But accessing price directly, in this case, would not be following the intent of how the Shirt class was designed.

One of the benefits of set and get methods is that, as previously mentioned in the course, you can hide the implementation from your user. Maybe originally a variable was coded as a list and later became a dictionary. With set and get methods, you could easily change how that variable gets accessed. Without set and get methods, you'd have to go to every place in the code that accessed the variable directly and change the code.

You can read more about get and set methods in Python on this **Python Tutorial site**.

A Note about Attributes

There are some drawbacks to accessing attributes directly versus writing a method for accessing attributes.

In terms of object-oriented programming, the rules in Python are a bit looser than in other programming languages. As previously mentioned, in some languages, like C++, you can explicitly state whether or not an object should be allowed to change or access an attribute's values directly. Python does not have this option.

Why might it be better to change a value with a method instead of directly? Changing values via a method gives you more flexibility in the long-term. What if the units of measurement change, like the store was originally meant to work in US dollars and now has to handle Euros? Here's an example:

Example Dollars versus Euros

If you've changed attribute values directly, you'll have to go through your code and find all the places where US dollars were used, like:

```
shirt_one.price = 10 # US dollars
```

and then manually change to Euros

```
shirt_one.price = 8 # Euros
```

If you had used a method, then you would only have to change the method to convert from dollars to Euros.

```
def change_price(self, new_price):
    self.price = new_price * 0.81 # convert dollars to Euros

shirt_one.change_price(10)
```

For the purposes of this introduction to object-oriented programming, you will not need to worry about updating attributes directly versus with a method; however, if you decide to further your studies of object-oriented programming, especially in another language such as C++ or Java, you'll have to take this into consideration.

Modularized Code

Thus far in the lesson, all of the code has been in Jupyter Notebooks. For example, in the previous exercise, a code cell loaded the Shirt class, which gave you access to the Shirt class throughout the rest of the notebook; however, if you were developing a software program, you would want to modularize this code.

You would put the Shirt class into its own Python script called, say, shirt.py. And then in another Python script, you would import the Shirt class with a line like: `from shirt import Shirt`.
For now, as you get used to OOP syntax, you'll be completing exercises in Jupyter notebooks. But midway through the lesson, you'll modularize object-oriented code into separate files.

https://www.youtube.com/watch?v=NcgDIWm6iBA&feature=emb_logo

Commenting Object-Oriented Code

Did you notice anything special about the answer key in the previous exercise? The Pants class and the SalesPerson class contained docstrings! A docstring is a type of comment that describes how a Python module, function, class or method works. Docstrings, therefore, are not unique to object-oriented programming. This section of the course is merely reminding you to use docstrings and to comment your code. It's not just going to help you understand and maintain your code. It will also make you a better job candidate.

From this point on, please always comment your code. Use both in-line comments and document level comments as appropriate.

Check out this **link** to read more about docstrings.

Docstrings and Object-Oriented Code

Below is an example of a class with docstrings and a few things to keep in mind:

- Make sure to indent your docstrings correctly or the code will not run. A docstring should be indented one indentation underneath the class or method being described.
- You don't have to define 'self' in your method docstrings. It's understood that any method will have self as the first method input.

```
class Pants:
    """The Pants class represents an article of clothing sold in a store
    """

    def __init__(self, color, waist_size, length, price):
        """Method for initializing a Pants object

        Args:
            color (str)
            waist_size (int)
            length (int)
            price (float)

        Attributes:
            color (str): color of a pants object
            waist_size (str): waist size of a pants object
            length (str): length of a pants object
            price (float): price of a pants object
        """

        self.color = color
        self.waist_size = waist_size
        self.length = length
        self.price = price

    def change_price(self, new_price):
        """The change_price method changes the price attribute of a pants object

        Args:
            new_price (float): the new price of the pants object
```

```
        Returns: None

        """
        self.price = new_price

    def discount(self, percentage):
        """The discount method outputs a discounted price of a pants object

        Args:
            percentage (float): a decimal representing the amount to discount

        Returns:
            float: the discounted price
        """
        return self.price * (1 - percentage)
```

Gaussian Class

Resources for Review

The example in the next part of the lesson assumes you are familiar with Gaussian and binomial distributions.

Here are a few formulas that might be helpful:

Gaussian Distribution Formulas
probability density function

https://www.youtube.com/watch?v=TVzNdFYyJIU&feature=emb_logo

Machine Learning Engineer Nanodegree Material

Gaussian Distribution Formulas

probability density function

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

μ is the mean
σ is the standard deviation
σ^2 is the variance

Binomial Distribution Formulas

mean

$$\mu = n * p$$

In other words, a fair coin has a probability of a positive outcome (heads) p = 0.5. If you flip a coin 20 times, the mean would be 20 * 0.5 = 10; you'd expect to get 10 heads.

variance

$$\sigma^2 = n * p * (1 - p)$$

Continuing with the coin example, n would be the number of coin tosses and p would be the probability of getting heads.

standard deviation

$$\sigma = \sqrt{n * p * (1 - p)}$$

or in other words, the standard deviation is the square root of the variance.

probability density function

$$f(k, n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{(n-k)}$$

e328bf96-8fb8-42f8-8f44-722a57

How the Gaussian Class Works

https://www.youtube.com/watch?time_continue=6&v=N-5I0d1zJHI&feature=emb_logo

Magic Methods

https://www.youtube.com/watch?v=9dEsv1aNUEE&feature=emb_logo

https://www.youtube.com/watch?v=oDuXThOqans&feature=emb_logo

Inheritance Code

In the following video, you'll see how to code inheritance using Python.
https://www.youtube.com/watch?v=uWT-HIHBjv0&feature=emb_logo

Inheritance Gaussian Class

https://www.youtube.com/watch?time_continue=11&v=XS4LQn1VA3U&feature=emb_logo

Advanced OOP Topics

Inheritance is the last object-oriented programming topic in the lesson. Thus far you've been exposed to:

- classes and objects
- attributes and methods
- magic methods
- inheritance

Classes, object, attributes, methods, and inheritance are common to all object-oriented programming languages.

Knowing these topics is enough to start writing object-oriented software. What you've learned so far is all you need to know to complete this OOP lesson. However, these are only the fundamentals of object-oriented programming.

Here is a list of resources for advanced Python object-oriented programming topics.

- **class methods, instance methods, and static methods** - these are different types of methods that can be accessed at the class or object level
- **class attributes vs instance attributes** - you can also define attributes at the class level or at the instance level
- **multiple inheritance, mixins** - A class can inherit from multiple parent classes
- **Python decorators** - Decorators are a short-hand way for using functions inside other functions

Organizing into Modules

https://www.youtube.com/watch?time_continue=1&v=AARS10U5bbo&feature=emb_logo

Windows vs. macOS vs. Linux

Linux, which our Udacity classroom workspaces use, is an operating system like Windows or macOS. One important difference is that Linux is free and open source whereas Windows is owned by Microsoft and macOS by Apple.

Throughout the lesson, you can do all of your work in a classroom workspace. These workspaces provide interfaces that connect to **virtual machines in the cloud**. However, if you want to run this code locally on your computer, the commands to use might be slightly different.
If you are using macOS, you can open an application called "Terminal" and essentially use the same commands that you use in the workspace. That is because **Linux and MacOS are related**.
If you are using Windows, the analogous application is called console. The console commands can be somewhat different than the terminal commands. A search engine like Google is your best friend for finding the right commands in a Windows environment.

The classroom workspace has one major benefit. You can do whatever you want to the workspace including installing Python packages. And if something goes wrong, you can reset the workspace and start with a clean slate; however, always download your code files or commit your code to github or gitlab before resetting a workspace. Otherwise, you will lose your code!

Making a Package

In the previous section, the Distribution and Gaussian code was refactored into individual modules. A Python module is just a Python file containing code.

In this next section, you'll convert the Distributions code into a Python package. A package is a collection of Python modules. Although the previous code might already seem like it was a Python package because it contained multiple files, a Python package

also needs an `__init__.py` file. In this section, you'll learn how to create this `__init__.py` file and then pip install the package into your local Python installation

https://www.youtube.com/watch?v=Hj2OBr1CGZM&feature=emb_logo

What is pip?

Pip is a **Python package manager** that helps with installing and uninstalling Python packages. You might have used pip to install packages using the command line: `pip install numpy`. When you execute a command like `pip install numpy`, pip will download the package from a Python package repository called **PyPi**. However, for this next exercise, you'll use pip to install a Python package from a local folder on your computer. The last part of the lesson will focus on uploading packages to PyPi so that you can share your package with the world.
You can complete this entire lesson within the classroom using the provided workspaces; however, if you want to develop a package locally on your computer, you should consider setting up a virtual environment. That way if you install your package on your computer, the package won't install into your main Python installation. Before starting the next exercise, the next part of the lesson will discuss what virtual environments are and how to use them.

Object-Oriented Programming and Python Packages

A Python package does not need to use object-oriented programming. You could simply have a Python module with a set of functions. However, most if not all of the popular Python packages take advantage of object-oriented programming for a few reasons:

1. Object-oriented programs are relatively easy to expand especially because of inheritance
2. Object-oriented programs obscure functionality from the user. Consider scipy packages. You don't need to know how the actual code works in order to use its classes and methods.

Virtual Environments

Python Environments

In the next part of the lesson, you'll be given a workspace where you can upload files into a Python package and pip install the package. If you decide to install your package on your local computer, you'll want to create a virtual environment. A virtual environment is a silo-ed Python installation apart from your main Python installation. That way you can install packages and delete the virtual environment without affecting your main Python installation

Let's talk about two different Python environment managers: conda and venv. You can create virtual environments with either one. Below you'll read about each of these environment managers including some advantages and disadvantages. If you've taken other data science, machine learning or artificial intelligence courses at Udacity, you're probably already familiar with **conda**.

Conda

Conda does two things: manages packages and manages environments.

As a package manager, conda makes it easy to install Python packages especially for data science. For instance, typing `conda install numpy` will install the numpy package. As an environment manager, conda allows you to create silo-ed Python installations. With an environment manager, you can install packages on your computer without affecting your main Python installation.

The command line code looks something like this:

```
conda create --name environmentname
source activate environmentname
conda install numpy
```

Pip and Venv

There are other environmental managers and package managers besides conda. For example, venv is an environment manager that comes pre-installed with Python 3. Pip is a package manager.

Pip can only manage Python packages whereas conda is a language agnostic package manager. In fact, conda was invented because pip could not handle data science packages that depended on libraries outside of Python. If you look at the **history of conda**, you'll find that the software engineers behind conda needed a way to manage data science packages (NumPy, Matplotlib, etc.) that relied on libraries outside of Python.
Conda manages environments AND packages. Pip only manages packages.

To use venv and pip, the commands look something like this:

```
python3 -m venv environmentname
source environmentname/bin/activate
pip install numpy
```

Which to Choose

Whether you choose to create environments with venv or conda will depend on your use case. Conda is very helpful for data science projects, but conda can make generic Python software development a bit more confusing; that's the case for this project.

If you create a conda environment, activate the environment, and then pip install the distributions package, you'll find that the system installs your package **globally rather than in your local conda environment**. However, if you create the conda environment and install pip simultaneously, you'll find that pip behaves as expected installing packages into your local environment:

```
conda create --name environmentname pip
```

On the other hand, using pip with venv works as expected. Pip and venv tend to be used for generic software development projects including web development. For this lesson

on creating packages, you can use conda or venv if you want to develop locally on your computer and install your package.

The video below shows how to use venv, which is what we recommend for this project.
Instructions for venv

Here are instructions about how to set up virtual environments on a macOS, Linux, or Windows machine using the terminal: **instructions link**.
These are a few notes for understanding the tutorial:

- If you are using Python 2.7.9 or later (including Python 3), the Python installation should already come with the Python package manager called pip. There is no need to install it.
- `env` is the name of the environment you want to create. You can call `env` anything you want.
- Python 3 comes with a virtual environment package pre-installed. So instead of typing `python3 -m virtualenv env`, you can type `python3 -m venv env` to create a virtual environment.

Once you've activated a virtual environment, you can then use terminal commands to go into the directory where your Python library is stored. And then you can run `pip install .`. In the next section, you can practice pip installing and/or creating virtual environments in the classroom workspace. You'll see that creating a virtual environment actually creates a new folder containing a Python installation. Deleting this folder will remove the virtual environment.
Note that if you install packages on the workspace and run into issues, you can always reset the workspace; however, you will lose all of your work. So be sure to download any files you want to keep before resetting a workspace.
https://www.youtube.com/watch?v=f7rzxUiHOJ0&feature=emb_logo

Binomial Class

https://www.youtube.com/watch?v=O-4qRh74rkI&feature=emb_logo

Binomial Class Exercise

In this next video, you'll get an overview of the binomial class exercise
https://www.youtube.com/watch?time_continue=137&v=xTamXY6Z9Kg&feature=emb_logo

Scikit-learn Source Code

https://www.youtube.com/watch?v=4_qkqMsbthg&feature=emb_logoz

Contributing to a GitHub project

Here are a few links about how to contribute to a github project:

- Beginner's Guide to Contributing to a Github Project
- Contributing to a Github Project

Advanced Python OOP Topics

Here are a few links to more advanced OOP topics that appear in the Scikit-learn package:

- Decorators
- Mixins

Putting Code on PyPi

https://www.youtube.com/watch?v=4uosDOKn5LI&feature=emb_logo

PyPi vs. Test PyPi

Note that pypi.org and test.pypy.org are two different websites. You'll need to register separately at each website. If you only register at pypi.org, you will not be able to upload to the test.pypy.org repository.
Also, remember that your package name must be unique. If you use a package name that is already taken, you will get an error when trying to upload the package.

Summary of the Terminal Commands Used in the Video

```
cd binomial_package_files
python setup.py sdist
pip install twine

# commands to upload to the pypi test repository
twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

```
pip install --index-url https://test.pypi.org/simple/ dsnd-probability

# command to upload to the pypi repository
twine upload dist/*
pip install dsnd-probability
```

More PyPi Resources

Tutorial on distributing packages

This link has a good tutorial on distributing Python packages including more configuration options for your setup.py file: **tutorial on distributing packages**. You'll notice that the python command to run the setup.py is slightly different with

```
python3 setup.py sdist bdist_wheel
```

This command will still output a folder called `dist`. The difference is that you will get both a .tar.gz file and a .whl file. The .tar.gz file is called a *source archive* whereas the .whl file is a *built distribution*. The .whl file is a newer type of installation file for Python packages. When you pip install a package, pip will first look for a whl file (wheel file) and if there isn't one, will then look for the tar.gz file.
A tar.gz file, ie an sdist, contains the files needed to **compile** and install a Python package. A whl file, ie a *built distribution*, only needs to be copied to the proper place for installation. Behind the scenes, pip installing a whl file has fewer steps than a tar.gz file.
Other than this command, the rest of the steps for uploading to PyPi are the same.

Other Links

If you'd like to learn more about PyPi, here are a couple of resources:

- **Overview of PyPi**
- **MIT License**

  Summary: https://www.youtube.com/watch?v=DStO1hBKtHQ&feature=emb_logo

Portfolio Exercise: Upload a Package to PyPi

Personal portfolios are an excellent way to demonstrate your knowledge and creativity. In fact, they are little by little becoming a must-have for people working in the tech industry. In this portfolio building exercise, you will build a Python package and upload the package to PyPi.

NOTE that a portfolio exercise like this is NOT reviewed. So you will not submit your work on this, and you do not need to complete this assignment in order to graduate.

Getting Started

Next, you'll find a blank classroom workspace where you can do your work. The benefits of using the workspace over your own computer are two fold:

- you shouldn't run into set-up issues
- if you install something on the workspace that causes issues, you can always reset everything

The workspace is the same Ubuntu Linux environment you've been using to do the exercises. You are also welcome to develop your package locally on your own computer if you prefer.

For local development, if you are developing on macOS, you can use the exact same terminal commands. On a windows machine, the commands are slightly different and you'll need to use the command prompt. This link contains a **comparison of MS-DOS vs Linux commands**.

If at some point you decide to reset your workspace, make sure you have saved your work somewhere else like in a GitHub repository; otherwise, all of your work will be deleted. You can connect to a remote GitHub repo from within the workspace terminal. Here is a reminder of how to do that: **adding an existing repository to GitHub using**

the command line. Another option is to download files or folders directly from the workspace. You right click on the file/folder name and click "Download".
In the workspace, you'll see that the files and folders have already been set up for you in terms of the structure, but the files themselves are empty.

Every time you type in the workspace, your work is automatically saved. But you should still save and commit your work to GitHub as well.

Instructions

Build a Python package and upload the package to PyPi. We encourage you to use object-oriented programming and tackle a problem that interests you. The Introduction to Object-Oriented Programming lesson has all of the information needed to create a package and upload the package. As mentioned in the lesson material, you'll need to use a unique name to upload the package to PyPi since each package needs a unique name; hence 'dsnd-probability' will not work since that package name is already taken.

Here are a few ideas for packages:

- create a package that does basic matrix algebra such as addition, subtraction, multiplication, matrix inversion, etc.
- make a calculus package that implements algorithms such as Newton's method
- create a package built on top of Matplotlib or Seaborn that creates well-formatted, well labeled visualizations
- create an object-oriented package to play a game of tic-tac-toe between two human players.
- or make a number guessing game where the computer randomly chooses an integer and then tells a human player if a guess is higher or lower than the number
  Feel free to come up with your own ideas. The main goal is to develop and demonstrate your skills in object-oriented programming and uploading packages to PyPi.

Reminders

- include a README file detailing the files in your package and how to install the package.

- Comment your code - use docstrings and inline comments where appropriate.
- Refactor code when possible - if you find your functions are getting too long, then refactor your code!
- Use object-oriented programming whenever it makes sense to do so.
- You're encouraged to write unit tests! The coding exercises in this lesson contained unit tests, so you can use those tests as a model for your package.
- Use GitHub for version control, and commit your work often.
  As a reminder, your package should be placed in a folder with the following folders and files:

- a folder with the name of your package that contains:
- the Python code that makes up your package
- a `README.md` file
- an `__init__.py`
- `license.txt`
- `setup.cfg`
- setup.py file

Hints and Helpful Links

Because this exercise requires writing and organizing code in a specific way, you might have to rewatch some of the lecture videos especially the "Putting Code on PyPi" concept.

Before you upload your code to PyPi, you should first pip install the package locally to make sure everything works as expected. The "Making a Package" and "Virtual Environments" lesson concepts should be helpful. You should also consider writing unit tests to test the functionality of your package. In the object-oriented programming lesson workspace, there were unit tests inside the 4a_binomial_package folder that you can use to help you get started. Those were in a file called test.py.

The object-oriented programming lesson included a complete, working package called dsnd-probability. We encourage you to code a project from scratch; however, if you get

stuck, use the dsnd-probability package code as a template. It already contains all of the necessary files you'll need for creating a package. It's also a simple example of object-oriented code. You can use these files, including the setup.py and setup.cfg files, to help structure your own code.

Those files are located in the "Exercise: Upload to PyPi" section inside the "5_exercise_upload_to_pypi" folder.

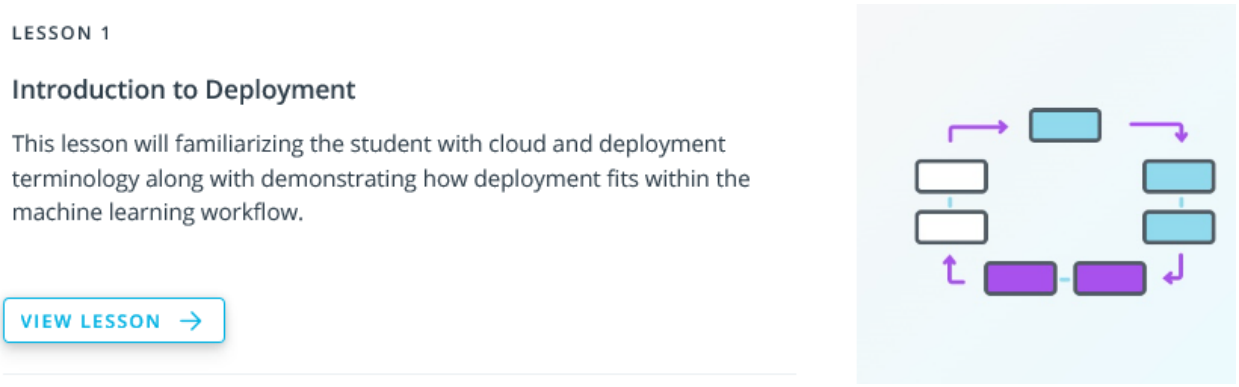
Besides the lesson on object-oriented programming, you might find **this package building summary guide from the Python website** helpful.
For a much more detailed explanation of distributing Python packages, check out the documentation on Distutils.

1. **Introduction**
2. **setup.py script**
3. **config file**
4. **source distributions**
5. **built distributions**
6. **uploading to PyPi**

Uploading to PyPi

When you are ready to upload your package, you can first upload to the **PyPi test repository**. Once everything is working as expected, you can upload to the public facing **PyPi repository**.
As a reminder, you'll need to create a username for both the test and public facing repositories. You'll also need to pip install the twine package with: `pip install twine`.
You can rewatch the lesson videos to see how to upload your package.
Continue on to the next sections to get some troubleshooting tips, and access the project workspace. When you're finished with your project, show off your work on a personal website, GitHub, and LinkedIn.

Machine Learning in Production



https://www.youtube.com/watch?v=jQ2IZzga8Nw&feature=emb_logo

What's Ahead?

In this lesson, you're going to get familiar with what's meant by *machine learning deployment*. Then in the upcoming lessons, you will put these ideas to practice by using **Amazon's SageMaker**. SageMaker is just *one* method for deploying machine learning models.

Specifically in this lesson, we will look at answering the following questions:

1. What's the *machine learning workflow*?
2. How does ***deployment*** fit into the *machine learning workflow*?
3. What is *cloud computing*?
4. Why would we use *cloud computing* for ***deploying*** machine learning models?
5. Why isn't ***deployment*** a part of many machine learning curriculums?
6. What does it mean for a model to be ***deployed***?

7. What are the *essential* characteristics associated with the code of *deployed models*?

8. What are *different* cloud computing platforms we might use to ***deploy*** our machine learning models?
   At the end of this lesson, you'll understand the broader idea of ***machine learning deployment***. Then Sean will be guiding you through using **SageMaker** to *deploy* your own machine learning models. This is a lot to cover, but by the end you will have a general idea of *all* the concepts related to *deploying machine learning models* into real world *production systems*.

Problem Intro:

https://www.youtube.com/watch?time_continue=1&v=-ZtVV7RvGYY&feature=emb_logo

Machine Learning Workflow

https://www.youtube.com/watch?time_continue=18&v=ku_96X6TZas&feature=emb_logo

Cloud Computing

In this section, we will focus on answering two questions:

- What is cloud computing?
- Why would a business decide to use cloud computing?
  This section will be a high-level look. However, you can dive more into the details by looking at the *optional pages* added at the end of this section.

What is cloud computing?

You can think of **cloud computing** as transforming an *IT product* into an *IT service*. Consider the following example:

> Have you ever had to backup and store important files on your computer? Maybe these files are family photos from your last vacation. You might store these photos on a *flash drive*. These days you have an *alternative option* to store these photos in the ***cloud*** using a *cloud storage provider*, like: **Google Drive**, **Apple's iCloud**, or **Microsoft's OneDrive**.

Cloud computing can simply be thought of as transforming an *Information Technology (IT) product* into a *service*. With our vacation photos example, we transformed storing photos on an *IT product*, the ***flash drive***; into storing them *using a service*, like ***Google Drive***.

Using a *cloud storage service* provides the ***benefits*** of making it *easier to access* and *share* your vacation photos, because you no longer need the flash drive. You'll only need a device with an internet connection to *access* your photos and to grant permission to *others to access* your photos.

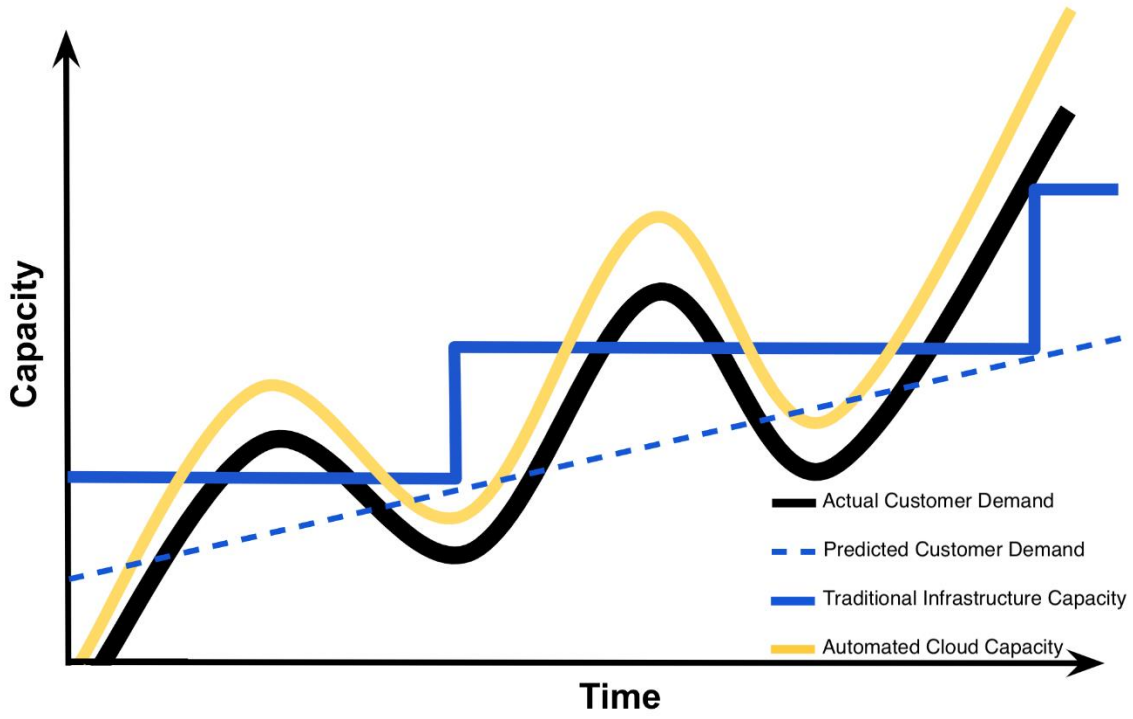
Generally, think of **cloud computing** as using an internet connected device to log into a *cloud computing service*, like ***Google Drive***, to access an *IT resource*, your vacation ***photos***. These *IT resources*, your vacation ***photos***, are stored in the *cloud provider's* ***data center***. Besides cloud storage, other cloud services include: cloud applications, databases, virtual machines, and other services like SageMaker.

Why would a business decide to use cloud computing?

Most of the factors related to choosing *cloud computing services*, instead of *developing on-premise IT resources* are related to ***time*** and ***cost***. The ***capacity utilization*** graph *below* shows how *cloud computing* compares to *traditional infrastructure* (on-premise IT resources) in meeting *customer demand*.

Capacity in the graph *below* can be thought of as the *IT resources* like: compute capacity, storage, and networking, that's needed to meet customer demand for a business' products and the ***costs*** associated with those *IT resources*. In our vacation photos example, *customer demand* is for storing and sharing customer photos. The *IT*

resources are the required software and hardware that enables photo storage and sharing in the *cloud* or *on-premise* (traditional infrastructure).
Looking at the graph, notice that *traditional infrastructure* doesn't scale when there are spikes in demand, and also leaves excess when preparing for *future* demand. This ability to easily meet unstable, fluctuating *customer demand* illustrates many of the benefits of *cloud computing*.



Summary of Benefits of Risks Associated with Cloud Computing

The ***capacity utilization*** graph above was initially used by cloud providers like Amazon to illustrate the ***benefits*** of cloud computing. Summarized below are the ***benefits*** of cloud computing that are often what *drives* businesses to include cloud services in their IT infrastructure [**1**]. These same ***benefits*** are echoed in those provided by cloud providers Amazon (**benefits**), Google (**benefits**), and Microsoft (**benefits**).

Benefits

1. Reduced Investments and Proportional Costs (providing cost reduction)

2. Increased Scalability (providing simplified capacity planning)

3. Increased Availability and Reliability (providing organizational agility)

Below we have also summarized he ***risks*** associated with cloud computing [**1**]. Cloud providers don't typically highlight the *risks* assumed when using their cloud services like they do with the *benefits*, but cloud providers like: Amazon (**security**), Google (**security**), and Microsoft (**security**) often provide details on security of their cloud services. It's up to the *cloud user* to understand the compliance and legal issues associated with housing data within a *cloud provider's* data center instead of on-premise. The service level agreements (SLA) provided for a cloud service often highlight security responsibilities of the cloud provider and those *assumed* by the cloud user.

Risks

1. (Potential) Increase in Security Vulnerabilities

2. Reduced Operational Governance Control (over cloud resources)

3. Limited Portability Between Cloud Providers

4. Multi-regional Compliance and Legal Issues

References

1. Erl, T., Mahmood, Z., & Puttini R,. (2013). *Cloud Computing: Concepts, Technology, & Architecture*. Upper Saddle River, NJ: Prentice Hall.

Chapter 3: Understanding Cloud Computing provides an outline of the business drivers, benefits and risks of cloud computing.

Additional Resources

For the purpose of deploying machine learning models, it's important to understand the *basics* of cloud computing. The essentials are provided above, but if you want to learn more about the overall role of cloud computing in developing software, check out the `[Optional] Cloud Computing Defined` and `[Optional] Cloud Computing Explained` sections at the end of this lesson along with the additional resources provided below.

- **National Institute of Standards and Technology** formal definition of **Cloud Computing**.
- Kavis, M. (2014). *Architecting the Cloud: Design Decisions for Cloud Computing Service Models.* Hoboken, NJ: Wiley. *Chapter 3* provides the worst practices of cloud computing which highlights both risks and benefits of cloud computing. *Chapter 9* provides the security responsibilities by service model.
- **Amazon Web Services** (AWS) discusses their definition of **Cloud Computing**.
- **Google Cloud Platform** (GCP) discusses their definition of **Cloud Computing**.
- **Microsoft Azure** (Azure) discusses their definition of **Cloud Computing**.

Machine Learning Applications

https://www.youtube.com/watch?time_continue=22&v=Q4rgQo6ofoc&feature=emb_logo

Notes:

All algorithms used within the machine learning workflow are *similar* for both the cloud and on-premise computing. The *only* real difference may be in the user interface and libraries that will be used to execute the machine learning workflow.

For ***personal use***, one's *likely* to use *cloud services*, if they don't have enough computing capacity.

With ***academic use***, quite often one will use the university's on-premise computing resources, given their availability. For smaller universities or research groups with few funding resources, *cloud services* might offer a viable alternative to university computing resources.

For ***workplace usage***, the amount of *cloud resources* used depends upon an organization's existing infrastructure and their vulnerability to the risks of cloud

computing. A workplace may have security concerns, operational governance concerns, and/or compliance and legal concerns regarding *cloud usage*. Additionally, a workplace may already have on-premise infrastructure that supports the workflow; therefore, making *cloud usage* an *unnecessary* expenditure. Keep in mind, many progressive companies may be incorporating *cloud computing* into their business due to the business drivers and benefits of cloud computing.

Deployment to Production

Recall that:

> ***Deployment to production*** can simply be thought of as a method that integrates a machine learning model into an existing production environment so that the model can be used to make *decisions* or *predictions* based upon *data input* into the model.

This means that moving from *modeling* to *deployment*, a *model* needs to be provided to those responsible for *deployment*. We are going to assume that the machine learning model we will be deploying was developed in Python.

Paths to Deployment

There are *three* primary methods used to transfer a model from the ***modeling*** *component* to the ***deployment*** *component* of the machine learning workflow. We will be discussing them in *order* of ***least*** to ***most*** commonly used. The ***third*** *method* that's *most* similar to what's used for *deployment* within ***Amazon's SageMaker***.

Paths to Deployment*:*

1. Python model is *recoded* into the programming language of the production environment.

2. Model is *coded* in *Predictive Model Markup Language* (PMML) or *Portable Format Analytics* (PFA).

3. Python model is *converted* into a format that can be used in the production environment.

Recoding Model into Programming Language of Production Environment

The *first* method which involves recoding the Python model into the language of the production environment, often Java or C++. This method is *rarely used anymore* because it takes time to recode, test, and validate the model that provides the *same* predictions as the *original*.

Model is Coded in PMML or PFA

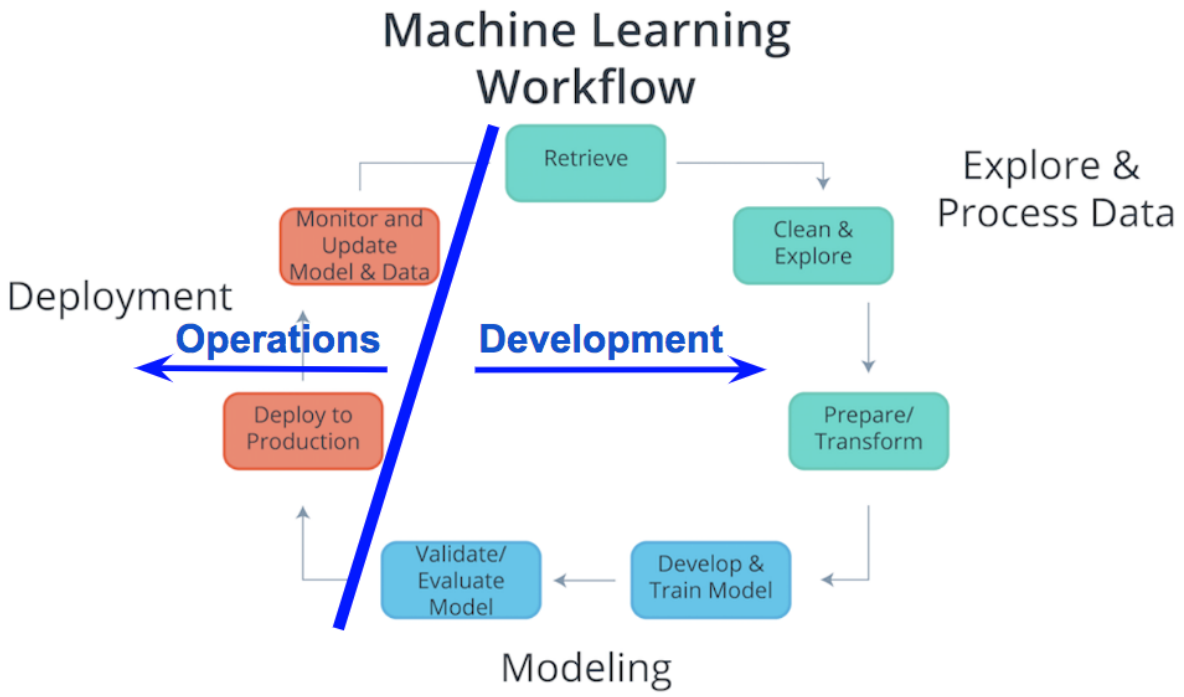
The *second* method is to code the model in Predictive Model Markup Language (PMML) or Portable Format for Analytics (PFA), which are two complementary standards that *simplify* moving predictive models to *deployment* into a *production environment*. The Data Mining Group developed both PMML and PFA to provide vendor-neutral executable model specifications for certain predictive models used by data mining and machine learning. Certain analytic software allows for the direct import of PMML including but *not limited* to IBM SPSS, R, SAS Base & Enterprise Miner, Apache Spark, Teradata Warehouse Miner, and TIBCO Spotfire.

Model is Converted into Format that's used in the Production Environment

The *third* method is to build a Python model and ***use*** *libraries* and *methods* that *convert* the model into ***code*** that can be used in the *production environment*. Specifically *most* popular machine learning software frameworks, like PyTorch, TensorFlow, SciKit-Learn, have methods that will *convert* Python models into *intermediate standard format*, like ONNX (**Open Neural Network Exchange** format). This *intermediate standard format* then can be ***converted*** into the software native to the *production environment*.

- This is the *easiest* and *fastest* way ***to move*** a Python model from *modeling* directly to *deployment*.
- Moving forward this is *typically* the way *models* are ***moved*** into the *production environment*.
- Technologies like *containers*, *endpoints*, and *APIs* (Application Programming Interfaces) also help ***ease*** the ***work*** required for *deploying* a model into the *production environment*.
  We will discuss these *technologies* in more detail in the ***next*** sections.

Machine Learning Workflow and DevOps



Machine Learning Workflow and Deployment

Considering the components of the *Machine Learning Workflow*, one can see how *Exploring and Processing Data* is tightly coupled with *Modeling*. The *modeling* **can't** occur without *first* having the *data* the model will be based upon *prepared* for the modeling process.

Comparatively *deployment* is **more** tightly coupled with the *production environment* than with *modeling* or *exploring and processing the data*. Therefore, *traditionally* there's was a separation between *Deployment* and the *other components* of the machine learning workflow. Specifically looking at the diagram ***above***, the **Process Data and Modeling** are considered ***Development***; whereas, **Deployment** is *typically* considered ***Operations***.

In the past typically, ***development*** was handled by **analysts**; whereas, ***operations*** was handled by **software developers** responsible for the *production environment*. With *recent* developments in *technology* (containers, endpoints, APIs) and the *most* common path of deployment; *this division* between ***development*** and ***operations*** *softens*. The

softening of this division enables **analysts** to handle certain aspects of ***deployment*** and enables faster updates to faltering models.

Deployment within Machine Learning Curriculum

Deployment is **not** commonly included in *machine learning curriculum*. This likely is associated with the *analyst's* typical focus on ***Exploring and Processing Data*** and ***Modeling***, and the *software developer's* focusing more on ***Deployment*** and the *production environment*. Advances in cloud services, like **SageMaker** and **ML Engine**, and deployment technologies, like Containers and REST APIs, allow for *analysts* to easily take on the responsibilities of ***deployment***.

Production Environment

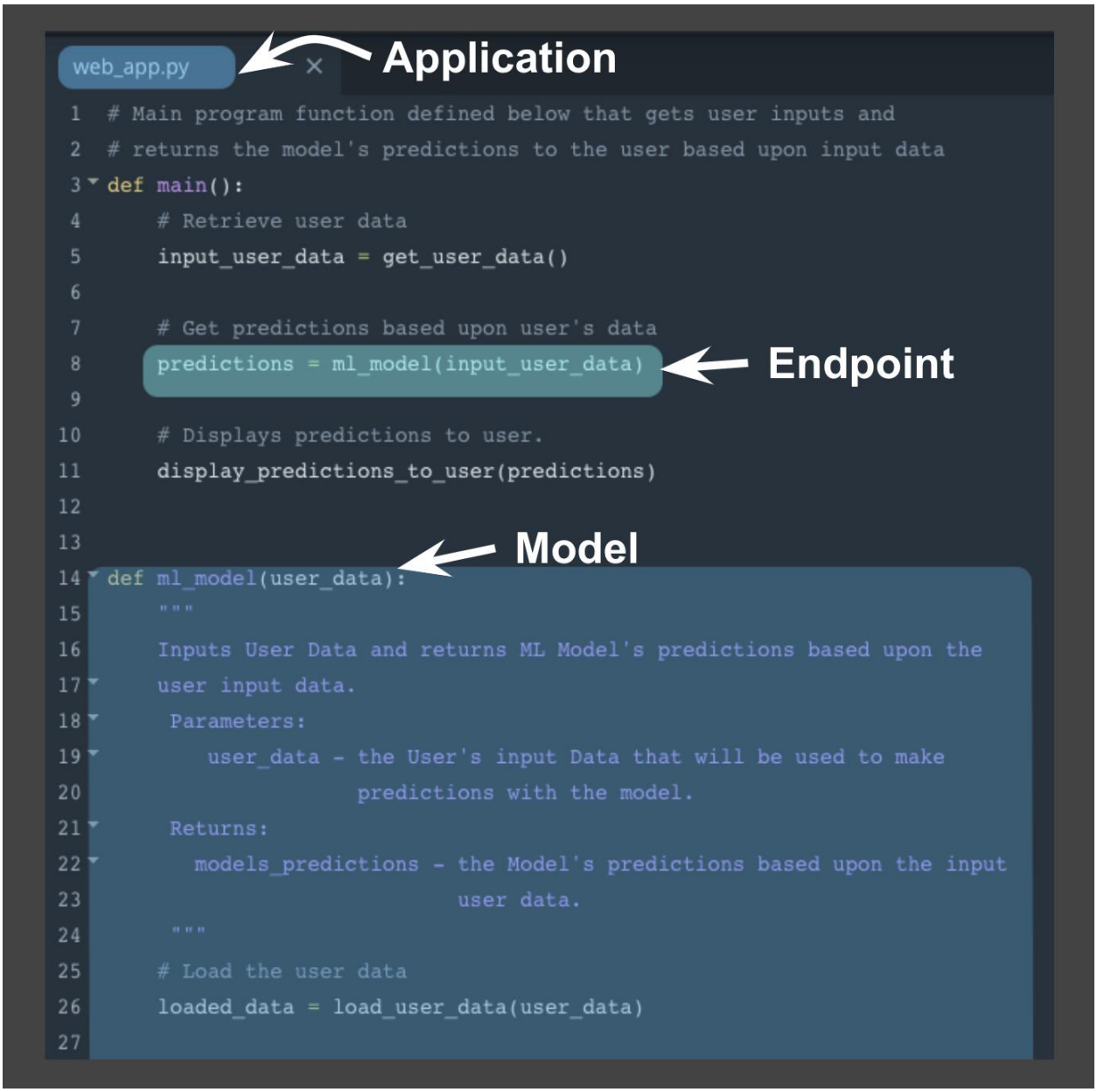
https://www.youtube.com/watch?time_continue=1&v=BH23Me3bbF4&feature=emb_logo

Production Environment and the Endpoint

When we discussed the *production environment*, the ***endpoint*** was defined as the **interface** to the model. This **interface** (***endpoint***) facilitates an ease of communication between the *model* and the *application*. Specifically, this **interface** (***endpoint***)

- Allows the *application* to send ***user data*** to the *model* and
- Receives ***predictions*** back from the *model* based upon that ***user data***.

Model, Application, and Endpoint

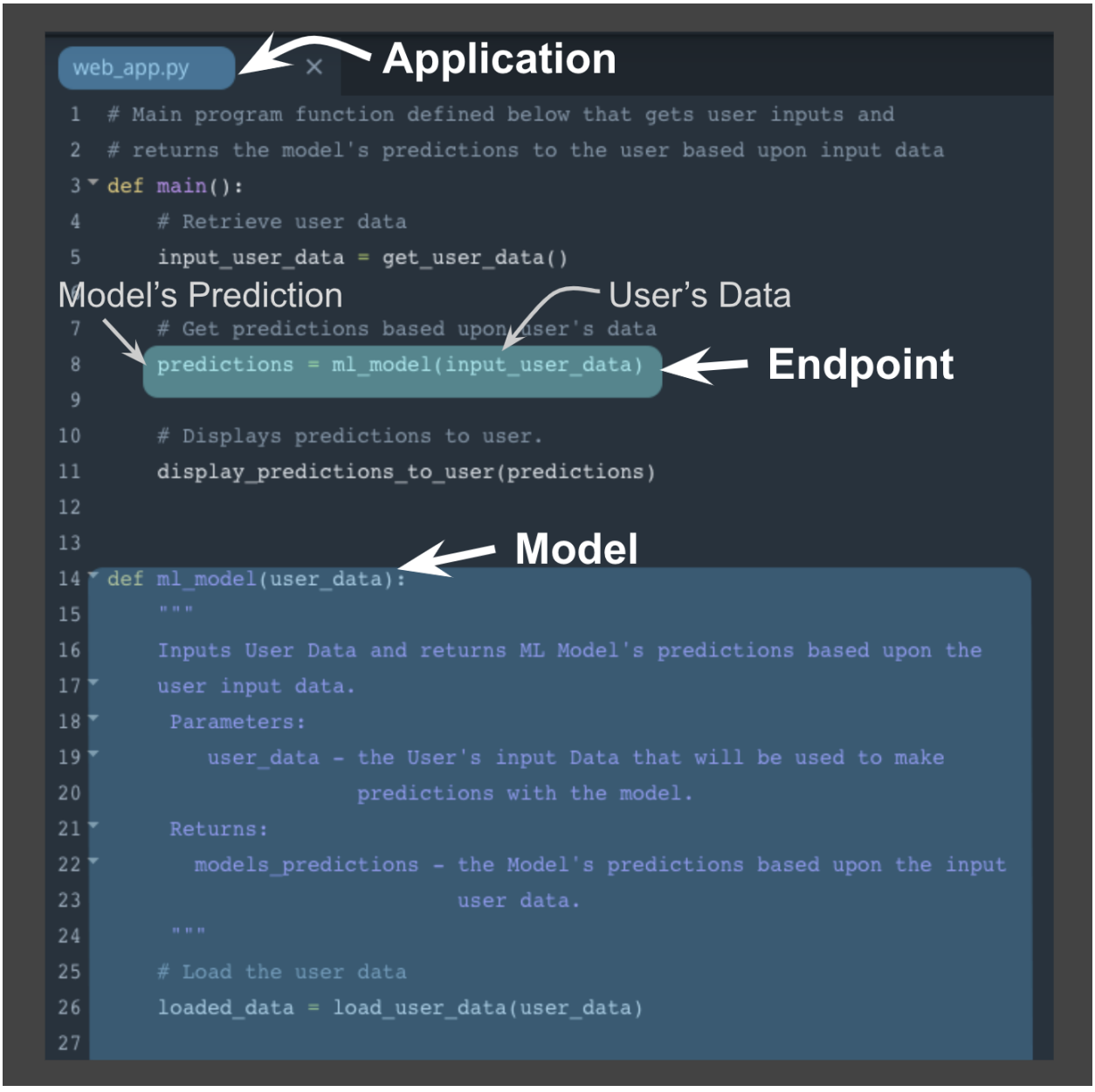


One way to think of the ***endpoint*** that acts as this *interface*, is to think of a *Python program* where:

- the **endpoint** itself is like a ***function call***
- the ***function*** itself would be the **model** and
- the ***Python program*** is the **application**.

The image **above** depicts the association between a ***Python program*** and the **endpoint**, **model**, and **application**.

- the **endpoint**: *line **8 function call*** to **ml_model**
- the **model**: beginning on *line **14 function definition*** for **ml_model**
- the **application**: ***Python program*** **web_app.py**



Using this example **above** notice the following:

- *Similar* to a ***function call*** the **endpoint** accepts *user data* as the ***input*** and ***returns*** the *model's prediction* based upon this ***input*** through the **endpoint**.

- In the example, the *user data* is the ***input argument*** and the *prediction* is the ***returned value*** from the ***function call***.
- The **application**, here the ***python program***, displays the *model's prediction* to the *application user*.

This example highlights how the **endpoint** itself is just the ***interface*** between the **model** and the **application**; where this ***interface*** enables users to get *predictions* from the ***deployed*** *model* based on their *user data*.
Next we'll focus on *how* the **endpoint** (***interface***) facilitates communication between **application** and **model**.

Endpoint and REST API

Communication between the **application** and the **model** is done through the **endpoint** (***interface***), where the **endpoint** is an **Application Programming Interface** (**API**).

- An *easy* way to think of an **API**, is as a *set of rules* that *enable* programs, here the **application** and the **model**, to *communicate* with each other.

- In this case, our **API** uses a **RE**presentational **S**tate **T**ransfer, **REST**, architecture that provides a framework for the *set of rules* and *constraints* that must be adhered to for *communication* between programs.

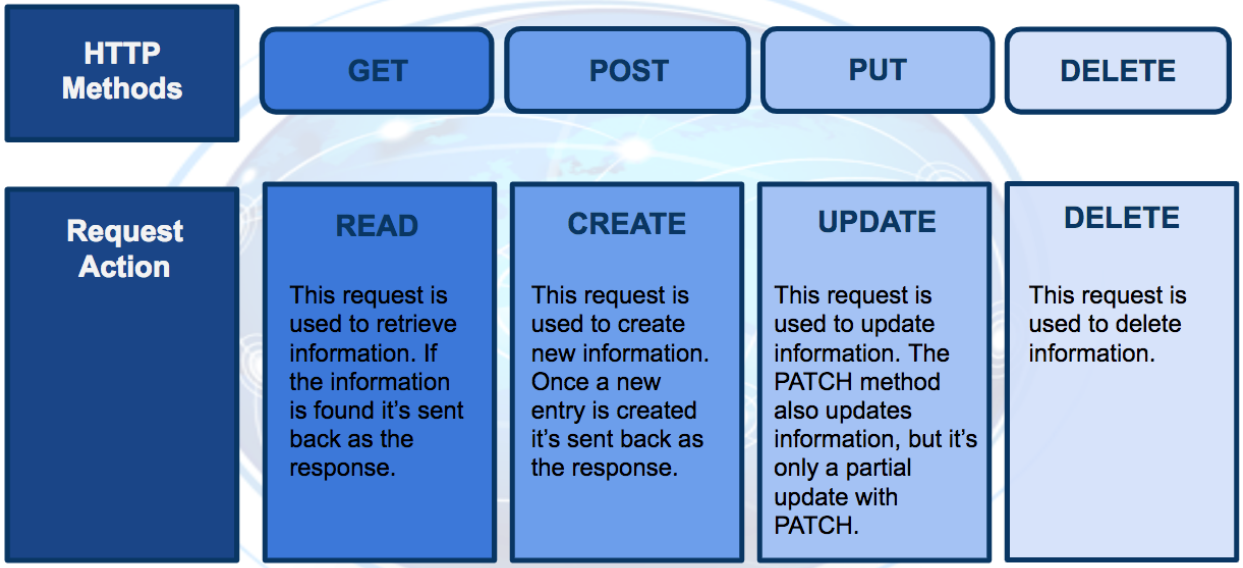
- This **REST API** is one that uses *HTTP requests* and *responses* to enable communication between the **application** and the **model** through the **endpoint** (***interface***).

- Noting that ***both*** the **HTTP request** and **HTTP response** are *communications* sent between the **application** and **model**.

The **HTTP request** that's sent from your **application** to your **model** is composed of *four* parts:

- **Endpoint**
  - This **endpoint** will be in the form of a URL, Uniform Resource Locator, which is commonly known as a web address.
- HTTP Method
  - Below you will find four of the **HTTP methods**, but for purposes of ***deployment*** our **application** will use the ***POST method*** *only*.
- HTTP Headers
  - The **headers** will contain additional information, like the format of the data within the message, that's passed to the *receiving* program.
- Message (Data or Body)

- The final part is the **message** (data or body); for ***deployment*** will contain the *user's data* which is input into the **model**.



The **HTTP response** sent from your model to your application is composed of *three* parts:

- HTTP Status Code
  - If the model successfully received and processed the *user's data* that was sent in the **message**, the status code should start with a ***2***, like *200*.
- HTTP Headers
  - The **headers** will contain additional information, like the format of the data within the **message**, that's passed to the receiving program.
- Message (Data or Body)
  - What's returned as the *data* within the **message** is the *prediction* that's provided by the **model**.

This *prediction* is then presented to the *application user* through the **application**. The **endpoint** is the ***interface*** that *enables communication* between the **application** and the **model** using a **REST API**.
As we learn more about **REST*ful* API**, realize that it's the **application's** responsibility:

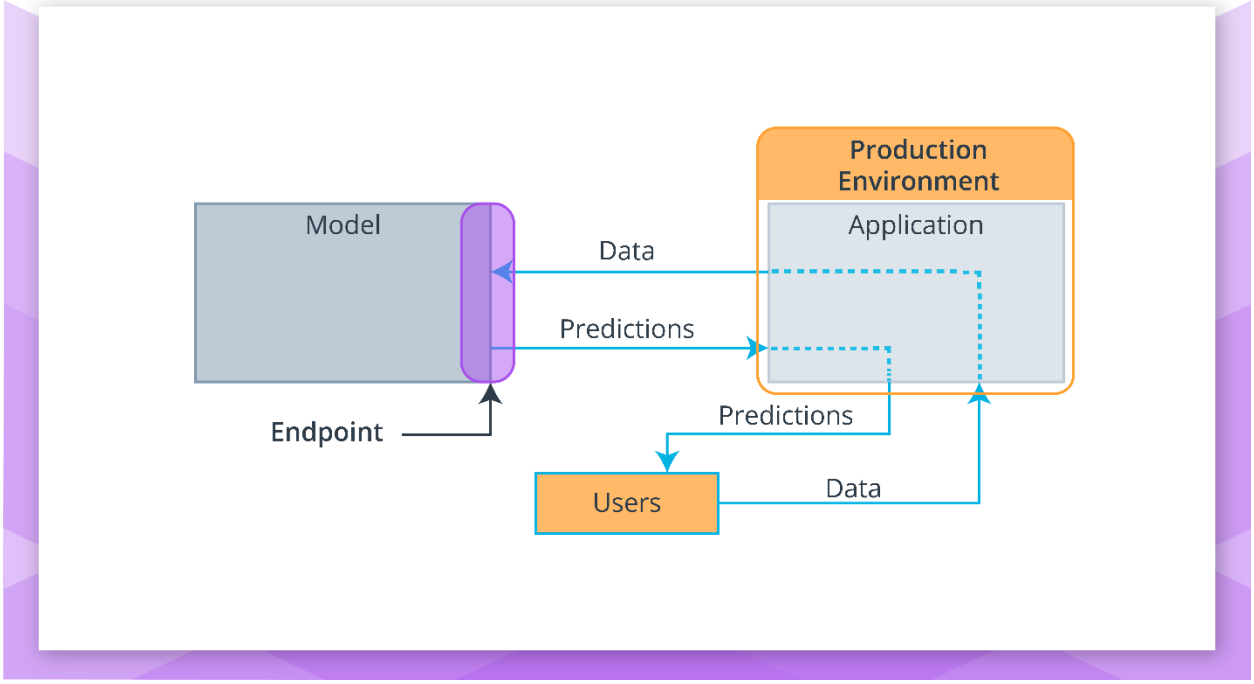
- To format the *user's data* in a way that can be easily put into the **HTTP request *message*** and ***used*** by the **model**.

- To translate the *predictions* from the **HTTP response *message*** in a way that's easy for the *application user's* to understand.

Notice the following regarding the *information* included in the ***HTTP messages*** sent between **application** and **model**:

- Often *user's data* will need to be in a *CSV* or *JSON* format with a specific *ordering* of the data that's dependent upon the **model** used.
- Often *predictions* will be returned in *CSV* or *JSON* format with a specific *ordering* of the returned *predictions* dependent upon the **model** used.

Containers



Model, Application, and Containers

When we discussed the *production environment*, it was composed of two primary programs, the **model** and the **application**, that *communicate* with each other through the **endpoint** (***interface***).

- The **model** is simply the *Python model* that's created, trained, and evaluated in the ***Modeling*** component of the *machine learning workflow*.

- The **application** is simply a *web* or *software application* that *enables* the application users to use the *model* to retrieve *predictions*.

Both the **model** and the **application** require a *computing environment* so that they can be run and available for use. One way to *create* and *maintain* these *computing environments* is through the use of ***containers***.

- Specifically, the **model** and the **application** can each be run in a ***container*** *computing environment*. The ***containers*** are created using a ***script*** that contains instructions on which software packages, libraries, and other computing attributes are needed in order to run a *software application*, in our case either the **model** or the **application**.

Containers Defined

- A ***container*** can be thought of as a *standardized collection/bundle of software* that is to be *used* for the specific purpose of *running an application*.

As stated **above *container*** *technology* is *used to create* the **model** and **application** *computational environments* associated with ***deployment*** in machine learning. A common **container** software is *Docker*. Due to its popularity sometimes *Docker* is used synonymously with **containers**.

Containers Explained

Often to first explain the concept of ***containers***, people tend to use the analogy of how Docker *containers* are similar to shipping containers.

- Shipping containers can contain a wide variety of products, from food to computers to cars.
- The structure of a shipping container provides the ability for it to hold *different types* of products while making it *easy* to track, load, unload, and transport products worldwide within a shipping container.

Similarly *Docker* **containers**:

- Can *contain* ***all*** types of *different* software.
- The structure of a *Docker* **container** enables the **container** to be *created*, *saved*, *used*, and *deleted* through a set of *common tools*.
- The *common tool set* works with ***any*** **container** regardless of the software the **container** contains.

Container Structure

The image **below** shows the basic structure of a **container**, you have:

- The underlying *computational infrastructure* which can be: a cloud provider's data center, an on-premise data center, or even someone's local computer.
- Next, you have an *operating system* running on this computational infrastructure, this could be the operating system on your local computer.
- Next, there's the *container engine*, this could be *Docker* software running on your local computer. The *container engine* software enables one to create, save, use, and delete containers; for our example, it could be *Docker* running on a local computer.
- The final two layers make up the composition of the *containers*.
    - The first layer of the container is the *libraries* and *binaries* required to launch, run, and maintain the *next* layer, the *application* layer.
- The image **below** shows *three* containers running *three* different applications.

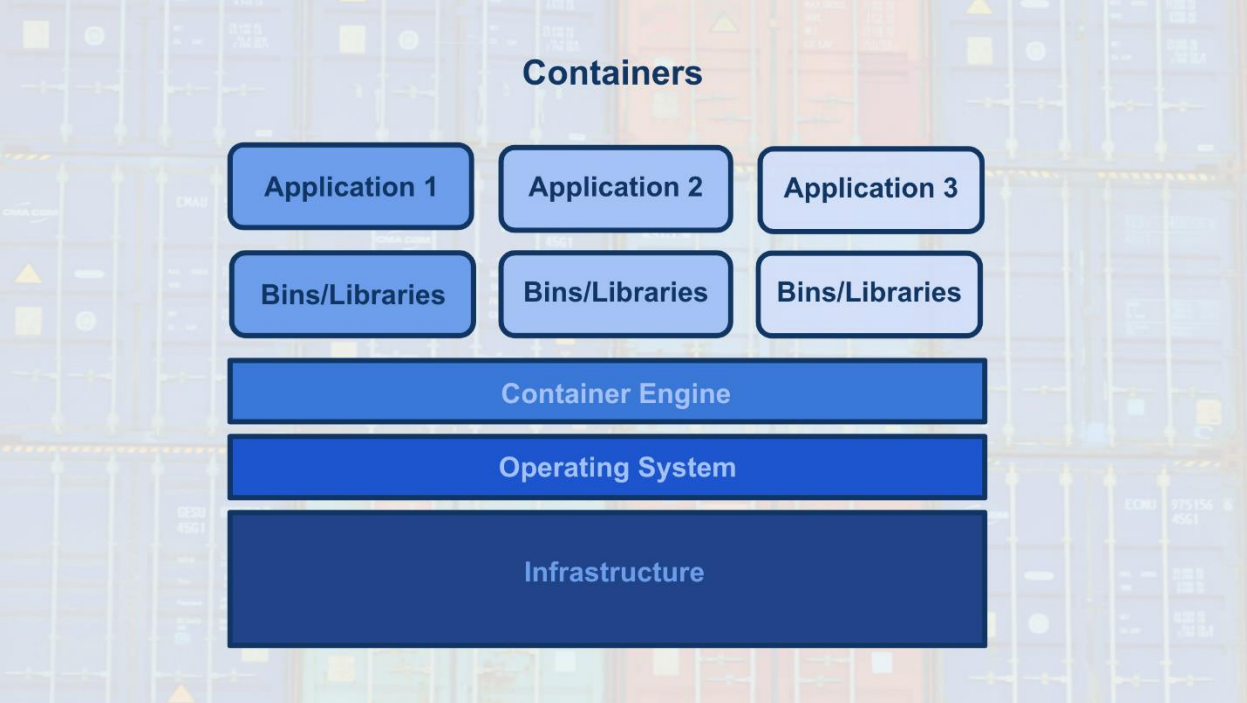
This *architecture* of **containers** provides the following *advantages*:

1. Isolates the application, which *increases* security.

2. Requires *only* software needed to run the application, which uses computational resources *more efficiently* and allows for faster application deployment.

3. Makes application creation, replication, deletion, and maintenance easier and the same across all applications that are deployed using containers.

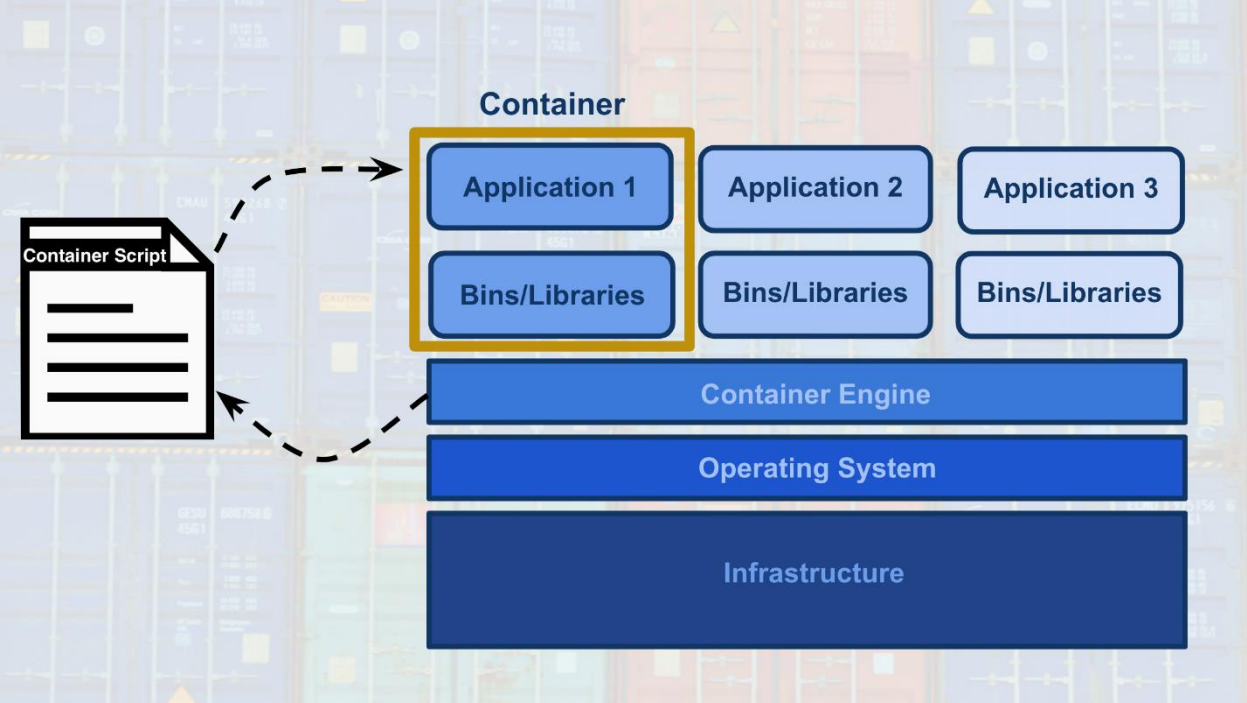
4. Provides a more simple and secure way to replicate, save, and share containers.



As indicated by the ***fourth*** *advantage* of using ***containers***, a ***container*** *script file* is used to create a ***container***.

- This *text script file* can easily be shared with others and provides a simple method to *replicate* a particular ***container***.
- This ***container*** *script* is simply the *instructions (algorithm)* that is used to create a ***container***; for *Docker* these ***container*** *scripts* are referred to as *dockerfiles*.

This is shown with the image **below**, where the *container engine* uses a ***container*** *script* to create a ***container*** for an application to run within. These ***container*** *script files* can be stored in repositories, which provide a simple means to share and replicate *containers*. For *Docker*, the **Docker Hub** is the official repository for storing and sharing *dockerfiles*. Here's an example of a **dockerfile** that creates a docker container with Python 3.6 and PyTorch installed.

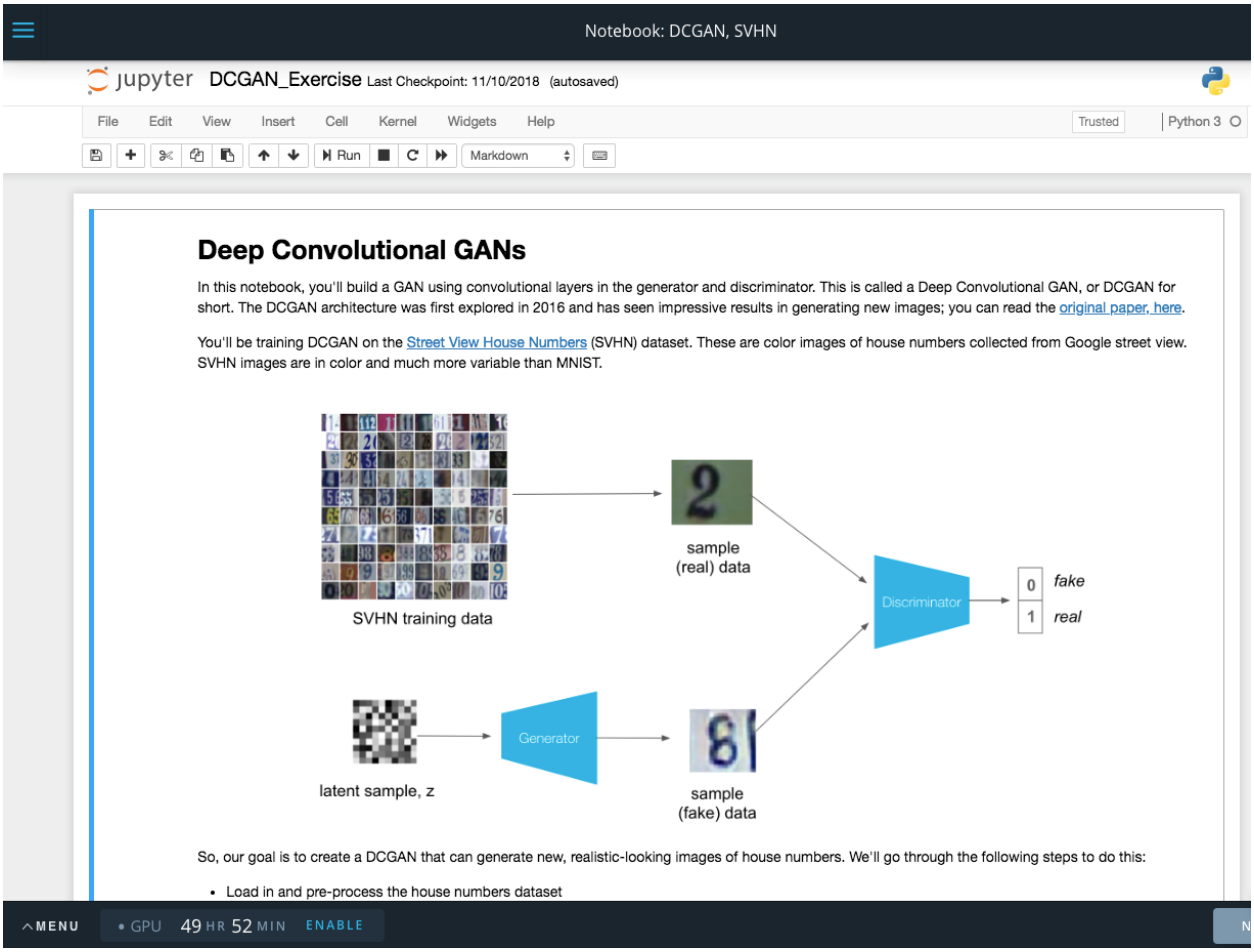


Expert Interview on Containers

We will be talking with Jesse Swidler, a senior software engineer at Udacity, to learn more about ***containers***. He will tell us more about what they *are*, what they *do*, and how they make his job *easier*. He'll even describe how we **use *containers*** when we work within a **Workspace**.

When you work on exercises and projects within a Nanodegree, *often* you will work within a **Workspace** environment within the classroom. Below you will find an image of a **Workspace** environment that contains a Jupyter notebook. The **Workspace** environment makes use of containers so that you can run PyTorch, Numpy, Pandas, without having to install those Python packages.

Example Workspace



https://www.youtube.com/watch?v=XimuK3WHOH4&feature=emb_logo

Characteristics of Modeling & Deployment

Send Feedback

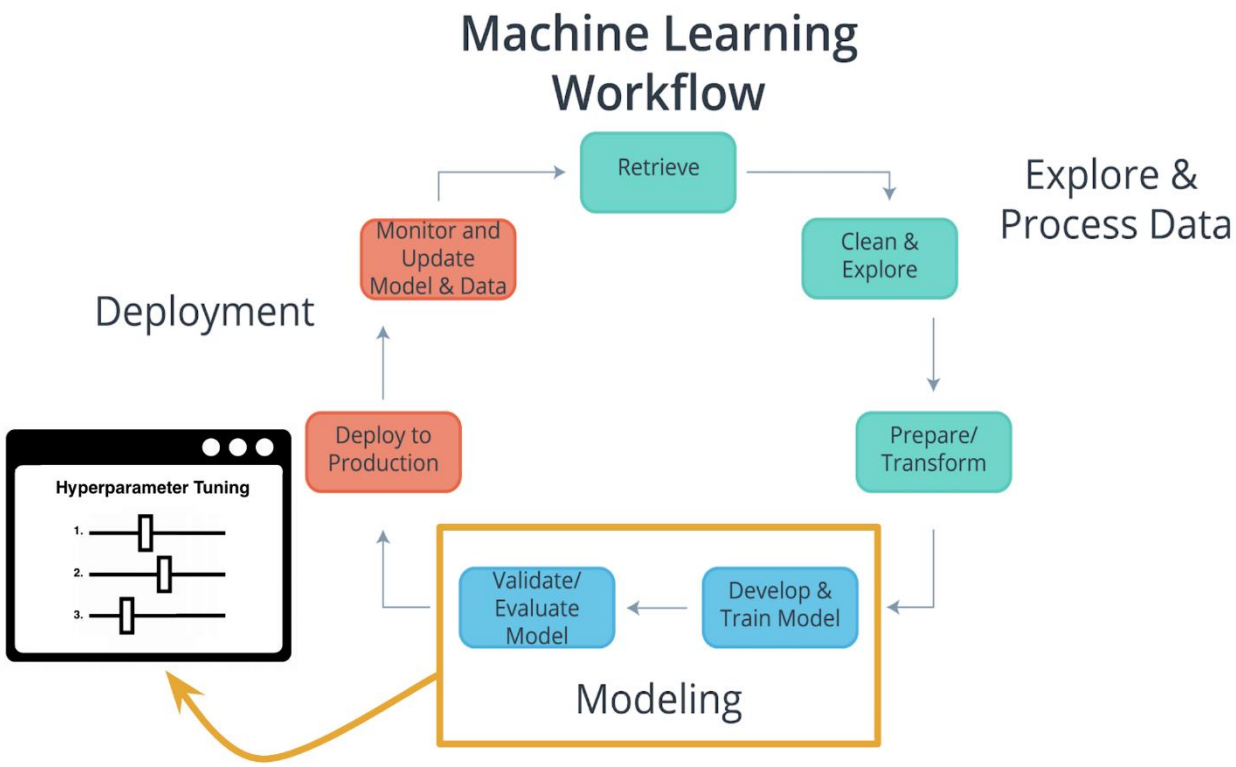
Characteristics of Deployment and Modeling

Recall that:

> ***Deployment*** to production can simply be thought of as a *method* that *integrates* a machine learning **model** into an *existing* **production environment** so that the **model** can be used to make *decisions* or *predictions* based upon *data* input into this **model**. Also remember that a ***production environment*** can be thought of as a *web*, *mobile*, or *other software* **application** that is *currently* being *used* by *many* people and must respond *quickly* to those users' requests.

Keeping these things in mind, there are a number of *characteristics* of **deployment** and **modeling** that I'm going to introduce here. These concepts are introduced *now* to provide

you with *familiarity* with these concepts for when you see them discussed in *future lessons*. Specifically, these concepts are provided as ***features*** that are made easier to use within cloud platforms services than if implemented with your own code.



Characteristics of Modeling

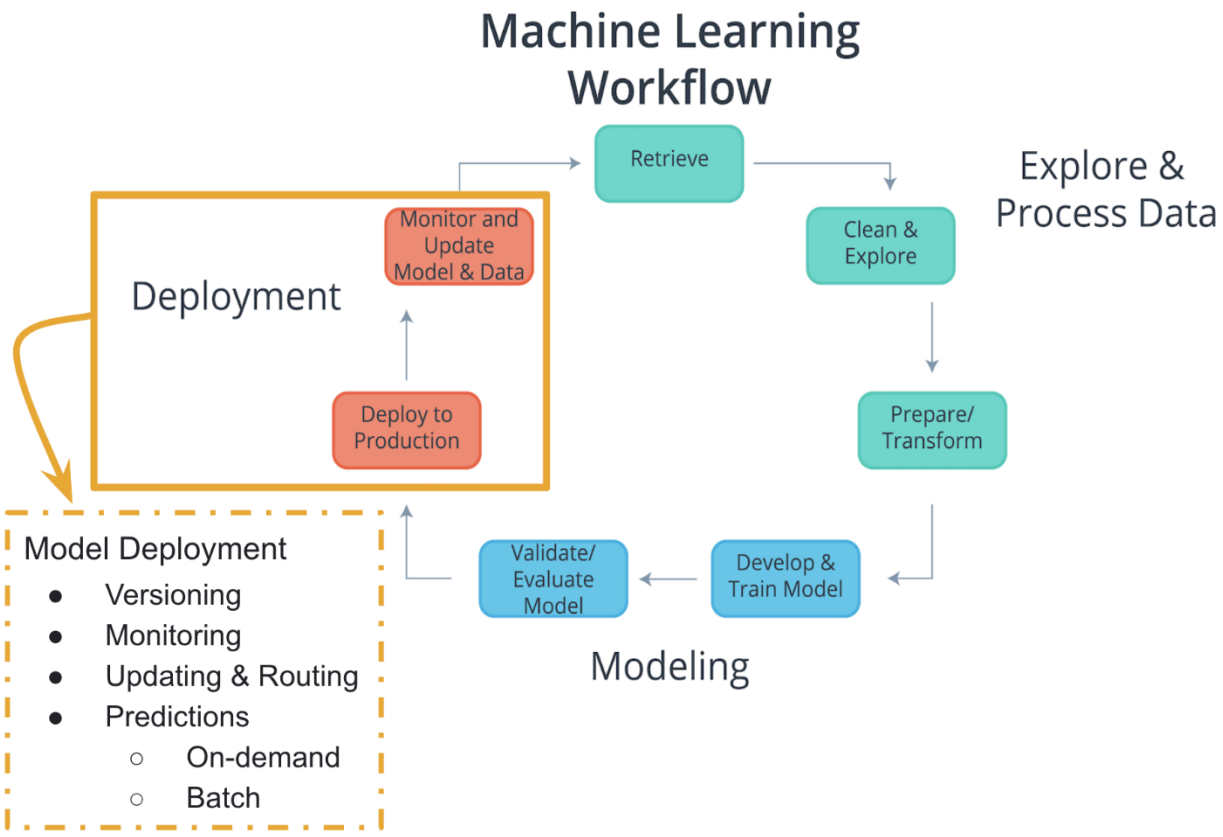
Hyperparameters

In machine learning, a **hyperparameter** is a parameter whose value *cannot* be estimated from the data.

- Specifically, a **hyperparameter** is *not directly* learned through the estimators; therefore, their value must be *set* by the model developer.
- This means that **hyperparameter** *tuning* for optimization is an ***important part*** of *model training*.
- Often cloud platform machine learning services provide methods that allow for **automatic hyperparameter** *tuning* for use with model training.

If the machine learning platform fails to offer an *automatic* **hyperparameter** option, one option is to use methods from **scikit-learn** Python library for **hyperparameter** *tuning*.

Scikit-learn is a free machine learning Python library that includes *methods* that help with **hyperparameter *tuning***.



Characteristics of Deployment

Model Versioning

One characteristic of deployment is the **version** of the model that is to be deployed.

- Besides saving the **model version** as a part of a *model's metadata* in a database, the *deployment platform* should allow one to indicate a deployed **model's version**.

This will make it easier to maintain, monitor, and update the *deployed model*.

Model Monitoring

Another characteristic of deployment is the ability to easily **monitor** your deployed models.

- Once a model is deployed you will want to make certain it continues to meet its performance metrics; otherwise, the application may need to be updated with a *better* performing model.

Model Updating and Routing

The ability to easily **update** your deployed model is another characteristic of deployment.

- If a deployed model is *failing* to meet its performance metrics, it's likely you will need to **update** this model.

If there's been a *fundamental change* in the *data* that's being input into the model for predictions; you'll want to **collect** this *input data* to be used to **update** the model.

- The *deployment platform* should support **routing** *differing* proportions of *user requests* to the deployed models; to allow *comparison* of performance between the deployed model *variants*.

Routing in this way allows for a test of a model *performance* as *compared* to other model *variants*.

Model Predictions

Another characteristic of deployment is the *type* of **predictions** provided by your deployed model. There are *two common* types of **predictions**:

- **On-demand predictions**
- **Batch predictions**

On-Demand Predictions

- **On-demand predictions** might also be *called*:
  - online,
  - real-time, or
  - synchronous predictions
- With these type of predictions, one *expects*:
  - a *low latency* of response to each prediction request,
  - but allows for possibility *high variability* in request volume.
- Predictions are returned in the response from the request. Often these requests and responses are done through an API using JSON or XML formatted strings.

- Each prediction request from the user can contain *one* or *many* requests for predictions. Noting that *many* is limited based upon the *size* of the data sent as the request. Common cloud platforms **on-demand prediction** request size limits can range from *1.5(ML Engine)* to *5 Megabytes (SageMaker)*.

On-demand predictions are commonly used to provide customers, users, or employees with real-time, online responses based upon a deployed model. Thinking back on our *magic eight ball web application* example, users of our web application *would be* making **on-demand prediction** requests.

Batch Predictions

- **Batch predictions** might also be *called*:
  - asynchronous, or
  - batch-based predictions.
- With these type of predictions, one *expects*:
  - *high volume* of requests with more *periodic submissions*
  - so *latency* won't be an issue.
- Each batch request will point to specifically *formatted data file* of requests and will return the predictions to a file. Cloud services ***require*** these files will be *stored* in the cloud provider's cloud.

- Cloud services typically have *limits* to how much data they can process with each batch request based upon *limits* they impose on the *size of file* you can store in their cloud storage service. For example, *Amazon's SageMaker* limits batch predictions requests to the size limit they enforce on an object in their S3 storage service.

Batch predictions are *commonly* used to help make *business decisions*. For example, imagine a business uses a complex model to predict customer satisfaction across a number of their products and they need these *estimates* for a *weekly* report. This would require processing customer data through a **batch prediction** request on a *weekly basis*.

Machine Learning Cloud Platforms

There are a number of machine learning cloud platforms, we provide *more* details about *a few* **below**. In the *next* few lessons, you will learn how to use **Amazon's SageMaker** to *deploy* machine learning models. Therefore, we *focused* on providing *more* information on **Amazon's SageMaker**. To allow for a comparison of features offered by **SageMaker**, we also provide detailed information about **Google's ML Engine** because it's *most* similar to **SageMaker**.

Amazon Web Services (AWS)

Amazon Web Services (AWS) SageMaker is Amazon's cloud service that allows you to *build*, *train*, and *deploy* machine learning models. Some advantages to using Amazon's SageMaker service are the following:

- **Flexibility in Machine Learning Software**: **SageMaker** has the flexibility to enable the use of ***any*** programming language or software framework for building, training, and deploying machine learning models in **AWS**. For the details see the three methods of modeling within **SageMaker below**.
  - **Built-in Algorithms** - There are at least fifteen built-in algorithms that are easily used within SageMaker. Specifically, built-in algorithms for discrete classification or quantitative analysis using **linear learner** or **XGBoost**, item recommendations using **factorization machine**, grouping based upon

attributes using **K-Means**, an algorithm for **image classification**, and many other algorithms.
    - **Custom Algorithms** - There are different programming languages and software frameworks that can be used to develop custom algorithms which include: **PyTorch**, **TensorFlow**, **Apache MXNet**, **Apache Spark**, and **Chainer**.
    - **Your Own Algorithms** - Regardless of the programming language or software framework, you can use your own algorithm when it **isn't** included within the *built-in* or *custom algorithms* **above**.
- **Ability to Explore and Process Data within SageMaker**: **SageMaker** enables the use of **Jupyter Notebooks** to explore and process data, along with creation, training, validation, testing, and deployment of machine learning models. This notebook interface makes data exploration and documentation easier.

- **Flexibility in Modeling and Deployment**: **SageMaker** provides a number of features and automated tools that make ***modeling*** and ***deployment*** easier. For the details on these features within **SageMaker** see **below**.
    - **Automatic Model Tuning**: **SageMaker** provides a feature that allows hyperparameter tuning to find the **best** version of the model for *built-in* and *custom algorithms*. For built-in algorithms **SageMaker** also provides evaluation metrics to evaluate the performance of your models.
    - **Monitoring Models**: **SageMaker** provides features that allow you to monitor your *deployed* models. Additionally with *model deployment*, one can choose *how much* traffic to route to *each* deployed model (model variant). More information on routing traffic to model variants can be found **here** and **here** .
    - **Type of Predictions**: **SageMaker** by *default* allows for **On-demand** type of predictions where *each* prediction *request* can contain *one* to *many* requests. **SageMaker** also allows for **Batch** predictions, and request *data* size limits are based upon S3 object size limits.

Google Cloud Platform (GCP)

Google Cloud Platform (GCP) ML Engine is Google's cloud service that allows you to *build*, *train*, and *deploy* machine learning models. Below we have highlighted some of the ***similarities*** and ***differences*** between these two cloud service platforms.

- **Prediction Costs**: The ***primary difference*** between the two is how they handle predictions. With ***SageMaker*** *predictions*, you must leave resources running to provide predictions. This enables *less* latency in providing predictions at the *cost* of paying for running *idle* services, if there are no (or few) prediction requests made while services are running. With ***ML Engine*** *predictions*, one has the option to *not* leave resources running which reduces cost associated with *infrequent* or *periodic* requests. Using this has *more* latency associated with predictions because the resources are in a offline state until they receive a prediction request. The *increased*

latency is associated to bringing resources back online, but one *only* pays for the time the resources are *in use*. To see more about **ML Engine pricing** and **SageMaker pricing**.

- **Ability to Explore and Process Data**: Another ***difference*** between **ML Engine** and **SageMaker** is the fact that *Jupyter Notebooks* are not available within **ML Engine**. To use *Jupyter Notebooks* within **Google's Cloud Platform** (GCP), one would use **Datalab**. **GCP** separates data exploration, processing, and transformation into other services. Specifically, **Google's Datalab** can be used for data exploration and data processing, **Dataprep** can be used to explore and transform raw data into clean data for analysis and processing, and **DataFlow** can be used to deploy batch and streaming data processing pipelines. Noting that **Amazon Web Services** (AWS), also have data processing and transformation pipeline services like **AWS Glue** and **AWS Data Pipeline**.

- **Machine Learning Software**: The final ***difference*** is that **Google's ML Engine** has *less* flexibility in available software frameworks for building, training, and deploying machine learning models in **GCP** as compared to **Amazon's SageMaker**. For the details regarding the two available software frameworks for modeling within **ML Engine** see **below**.
    - **Google's TensorFlow** is an open source machine learning framework that was originally developed by the Google Brain team. **TensorFlow** can be used for creating, training, and deploying machine learning and deep learning models. **Keras** is a higher level API written in Python that runs on top of **TensorFlow**, that's easier to use and allows for faster development. GCP provides both **TensorFlow examples** and a **Keras example**.
    - **Google's Scikit-learn** is an open source machine learning framework in Python that was originally developed as a Google Summer of Code project. **Scikit-learn** and an **XGBoost Python package** can be used together for creating, training, and deploying machine learning models. In the in **Google's example**, **XGBoost** is used for modeling and **Scikit-learn** is used for processing the data.

- **Flexibility in Modeling and Deployment**: **Google's ML Engine** provides a number of features and automated tools that make ***modeling*** and ***deployment*** easier, ***similar*** to the those provided by **Amazon's SageMaker**. For the details on these features within **ML Engine** see **below**.
    - **Automatic Model Tuning**: **Google's ML Engine** provides a feature that enables hyperparameter tuning to find the **best** version of the model.
    - **Monitoring Models**: **Google's ML Engine** provides features that allow you to monitor your models. Additionally **ML Engine** provides methods that enable **managing runtime versions** and **managing models and jobs**.

- **Type of Predictions**: **ML Engine** allows for **Online**(*or On-demand*) type of predictions where *each* prediction *request* can contain *one* to *many* requests. **ML Engine** also allows for **Batch** predictions. More information about **ML Engine's Online and Batch predictions**.

Microsoft Azure

Similar to **Amazon's SageMaker** and **Google's ML Engine**, Microsoft offers **Azure AI**. **Azure AI** offers an open and comprehensive platform that includes AI software frameworks like: **TensorFlow**, **PyTorch**, **scikit-learn**, **MxNet**, **Chainer**, **Caffe2**, and other software like their **Azure Machine Learning Studio**. For more details see **Azure AI** and **Azure Machine Learning Studio**.

Paperspace

Paperspace simply provides GPU-backed virtual machines with industry standard software tools and frameworks like: **TensorFlow**, **Keras**, **Caffe**, and **Torch** for machine learning, deep learning, and data science. **Paperspace** claims to provide more powerful and less expensive virtual machines than are offered by **AWS**, **GCP**, or **Azure**.

Cloud Foundry

Cloud Foundry is an open source cloud application platform that's backed by companies like: Cisco, Google, IBM, Microsoft, SAP, and more. **Cloud Foundry** provides a faster and easier way to build, test, deploy, and scale applications by providing a choice of clouds, developer frameworks, and applications services to it's users. **Cloud Foundry Certified Platforms** provide a way for an organization to have their cloud applications portable across platforms including **IBM** and **SAP** cloud platforms.

Lesson Summary:

https://www.youtube.com/watch?time_continue=1&v=fXl_MCYzcOU&feature=emb_logo

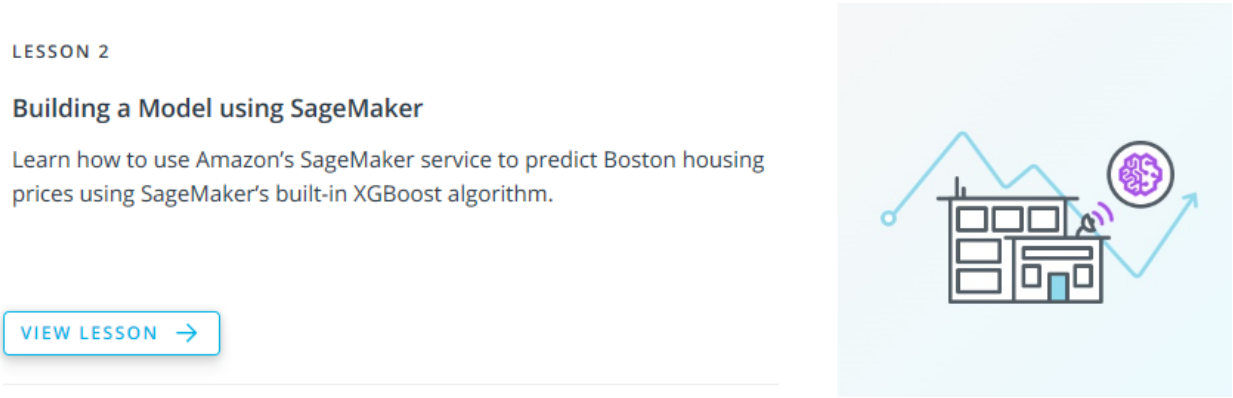
Summary

Let's summarize the ideas covered in this lesson to ensure you are leaving with the ***most*** important parts!

Specifically in this lesson, we looked at answering the following questions:

1. What is the ***machine learning workflow***?
2. How does ***deployment*** fit into the machine learning workflow?
3. What is ***cloud computing***?
4. Why would we use *cloud computing* for ***deploying machine learning*** models?
5. Why isn't ***deployment*** a part of many *machine learning curriculums*?
6. What does it mean for a model to be ***deployed***?
7. What are the ***crucial characteristics*** associated with ***deploying*** models?
8. What are *different* ***cloud computing platforms*** we might use to ***deploy*** our *machine learning models*?

Lesson 2: Bulding a model using SageMaker



Introduction to Amazon SageMaker

Hi, I'm Sean, and together we are going to learn about Amazon's SageMaker service.

To start with, we are going to need to set a few things up.

Note: Amazon is constantly iterating and improving the various services that they offer. Sometimes this involves changes to the way certain things look, however, the functionality should be the same. So in the screenshots and videos that follow, if *your* screen doesn't look exactly the same, don't worry!
https://www.youtube.com/watch?v=nJCc4_9-iAQ&feature=emb_logo

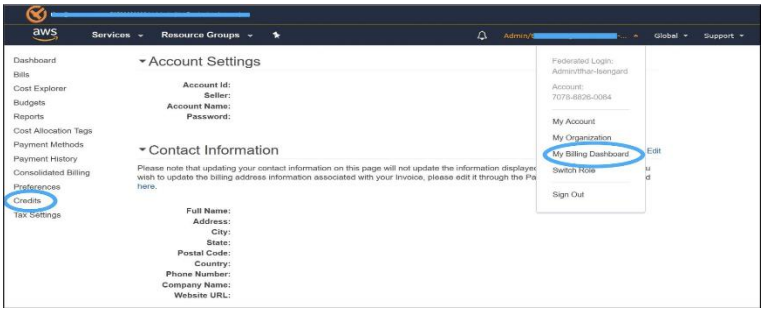
Create an AWS Account

1. Open a regular AWS account (if you don't already have one) following the instructions via the **Amazon Web Service Help Center**
    2. You will need a promo code from us so you can apply it to your account. To request a promo code, you can submit a support ticket **here**.
- Under the **"Reason for Contact"** field, choose **"Other"**, then choose **"External Tools"** in the dropdown.

- When the **"External Tools"** field appears, select **"AWS"**.
- Please note that a regular AWS account will receive a promo code from Udacity with a fixed amount of AWS credits.

3. To apply your promo code, follow below:

- Click **"Credits"** on the left side of the screen and enter the promo-code you received, then hit **"redeem"**.
- Refresh the page and you will be able to view your credits under: Below are all the credits you have redeemed with AWS. Credits will automatically be applied to your bill.



A. What is AWS Sagemaker?

AWS (or Amazon) SageMaker is a *fully managed* service that provides the ability to build, train, tune, deploy, and manage large-scale machine learning (ML) models quickly. Sagemaker provides tools to make each of the following steps simpler:

1. Explore and process data

- Retrieve
- Clean and explore
- Prepare and transform

2. Modeling

- Develop and train the model
- Validate and evaluate the model

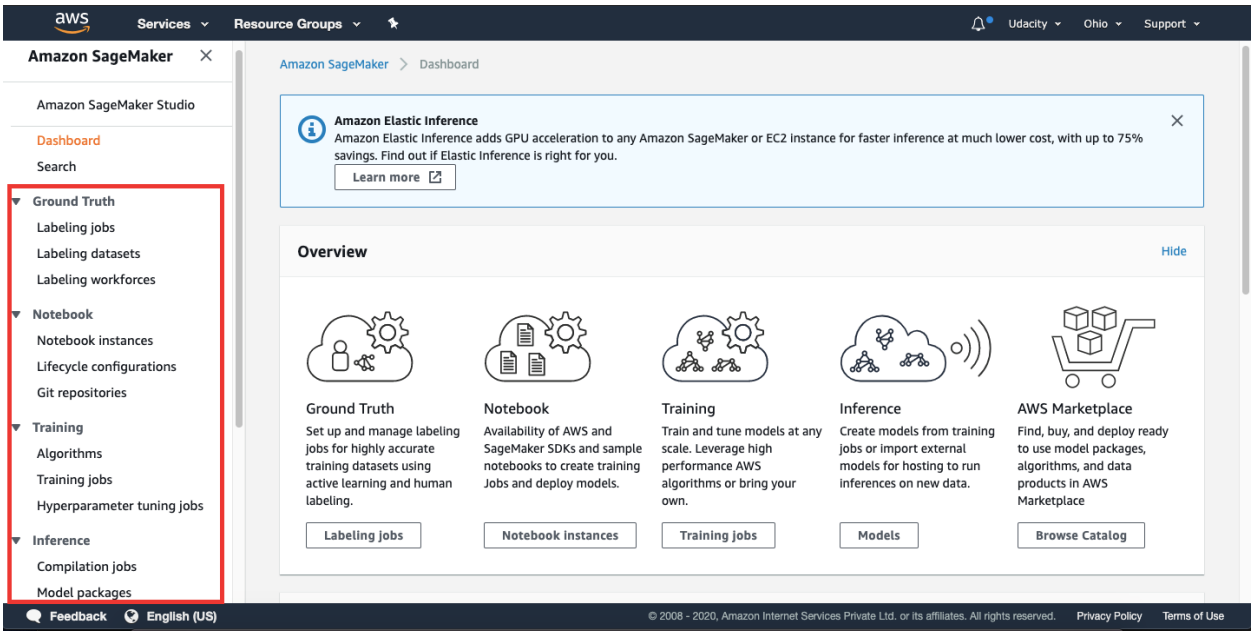
3. Deployment

- Deploy to production
- Monitor, and update model & data

The Amazon Sagemaker provides the following tools:

- Ground Truth - To label the jobs, datasets, and workforces
- Notebook - To create Jupyter notebook instances, configure the lifecycle of the notebooks, and attache Git repositories
- Training - To choose an ML algorithm, define the training jobs, and tune the hyperparameter
- Inference - To compile and configure the trained models, and endpoints for deployments

The snapshot of the *Sagemaker Dashboard* below shows the tools mentioned above.



IMPORTANT NOTICE: This is the current AWS UI as of April 6th, 2020. The AWS UI is subject to change on a regular basis. We advise students to refer to AWS documentation for the above process.

A.1. Why is SageMaker a "fully managed" service?

SageMaker helps to reduce the complexity of building, training and deploying your ML models by offering all these steps on a single platform. SageMaker supports building the ML models with *modularity*, which means you can reuse a model that you have already built earlier in other projects.

A.2. SageMaker Instances - Important to Read

SageMaker instances are the dedicated VMs that are optimized to fit different machine learning (ML) use cases. The supported instance types, names, and pricing in SageMaker are different than that of EC2. Refer the following links to have better insight:

- **Amazon SageMaker ML Instance Types** - ***See that an instance type is characterized by a combination of CPU, memory, GPU, GPU memory, and networking capacity.***
- **Amazon EC2 Instance Types** - To have you know the difference in naming and combinations of CPU, memory, storage, and networking capacity.

A.3. Supported Instance Types and Availability Zones

Amazon SageMaker offers a variety of instance types. Interestingly, ***the type of SageMaker instances that are supported varies with AWS Regions and Availability Zones***.

- First, you need to check the **List of the AWS Regions that support Amazon SageMaker**.
- Next, you can check the various available **Amazon SageMaker ML Instance Types**, again.

A.4. Instances Required for Deep Learning

The table below describes the three types of SageMaker instances that you would use in this course:

SageMaker Instance	vCPU	GPU	Mem (GiB)	GPU Mem (GiB)	Network Performance	Usage
`ml.t2.medium`	2	-	4	-	Low to Moderate	Run notebooks

SageMaker Instance	vCPU	GPU	Mem (GiB)	GPU Mem (GiB)	Network Performance	Usage
`ml.m4.xlarge`	4	-	16	-	High	• Train and batch transform XGBOOST models; • Deploy all models preceding the first project
`ml.p2.xlarge`	4	1xK80	61	12	High	Train and batch transform GPU accelerated Pytorch models for the first project

In this course, the `ml.m4.xlarge` is needed at an early stage, while `ml.p2.xlarge` is needed only when working on the for the first project: Deploying a Sentiment Analysis Model.

Note

Sagemaker quotas, also referred to as limits, are very tricky. Every AWS user does not get the default quotas for SageMaker instances, which is why the last column shows a range, e.g., 0 - 20. The **Default Quota** depends on the instance type, the task you want to run (see table above), and also the region in which the Sagemaker service is requested. Refer **this document** having a caveat that new accounts may not always get the default limits.

Recommended Read

AWS Sagemaker FAQs

A. AWS Service Utilization Quota (Limits)

You need to understand the way AWS imposes ***utilization quotas*** (limits) on almost all of its services. *Quotas*, also referred to as *limits*, are the maximum number of resources of a particular service that you can create in your AWS account.

- AWS provides default quotas, **for each AWS service.**
- Importantly, **each quota is region-specific.**
- There are three ways to **view your quotas**, as mentioned **here**:

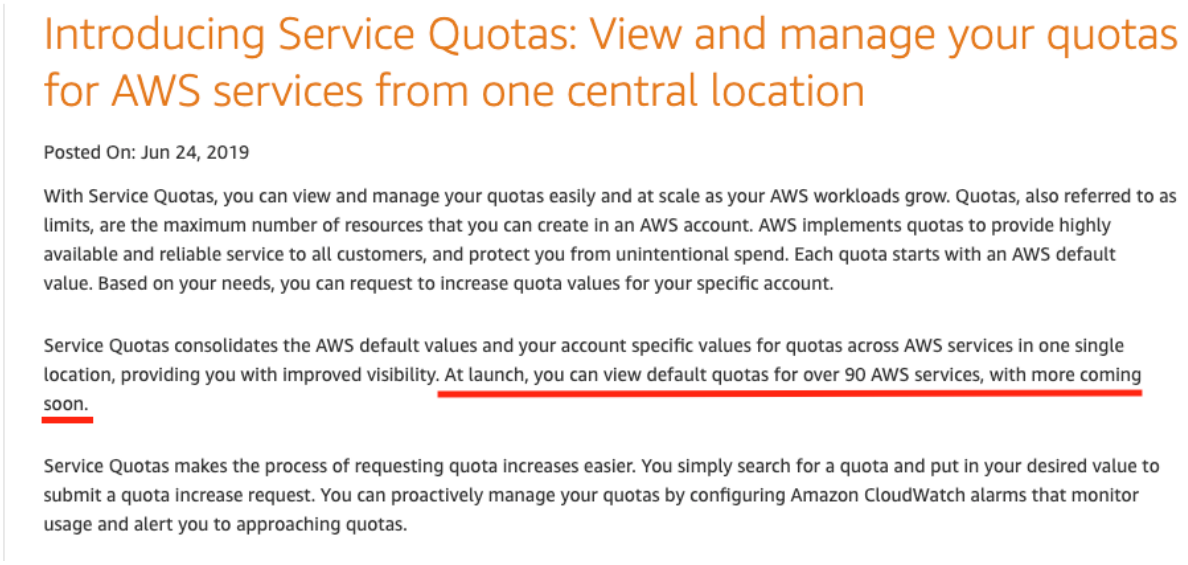
1. Service Endpoints and Quotas,
2. Service Quotas console, and
3. AWS CLI commands - `list-service-quotas` and `list-aws-default-service-quotas`

- In general, there are three ways to **increase the quotas**:

1. Using ***Amazon Service Quotas*** service - This service consolidates your account-specific values for quotas across all AWS services for improved manageability. Service Quotas is available at no additional charge. You can directly try logging into **Service Quotas console** here.
2. Using ***AWS Support Center*** - You can create a case for support from AWS.
3. AWS CLI commands - `request-service-quota-increase`

A.1. Amazon SageMaker Utilization Quota (Limits)

You can view the *Amazon SageMaker Service Limits* at **"Amazon SageMaker Endpoints and Quotas"** page. You can request to increase the AWS Sagemaker quota using the ***AWS Support Center*** only. Note that currently the *Amazon Service Quotas* does not support SageMaker service. However, SageMaker would be introduced soon into Service Quotas. AWS is moving to make users manage quotas for all AWS services from one central location.



Introducing Service Quotas: View and manage your quotas for AWS services from one central location

Posted On: Jun 24, 2019

With Service Quotas, you can view and manage your quotas easily and at scale as your AWS workloads grow. Quotas, also referred to as limits, are the maximum number of resources that you can create in an AWS account. AWS implements quotas to provide highly available and reliable service to all customers, and protect you from unintentional spend. Each quota starts with an AWS default value. Based on your needs, you can request to increase quota values for your specific account.

Service Quotas consolidates the AWS default values and your account specific values for quotas across AWS services in one single location, providing you with improved visibility. At launch, you can view default quotas for over 90 AWS services, with more coming soon.

Service Quotas makes the process of requesting quota increases easier. You simply search for a quota and put in your desired value to submit a quota increase request. You can proactively manage your quotas by configuring Amazon CloudWatch alarms that monitor usage and alert you to approaching quotas.

SageMaker would be introduced soon into Services Quota - Courtesy - *Amazon Service Quotas*

A.2. Increase Sagemaker Instance Quota (Limit) using AWS Support Center

Read the note and recommendation below before proceeding further.

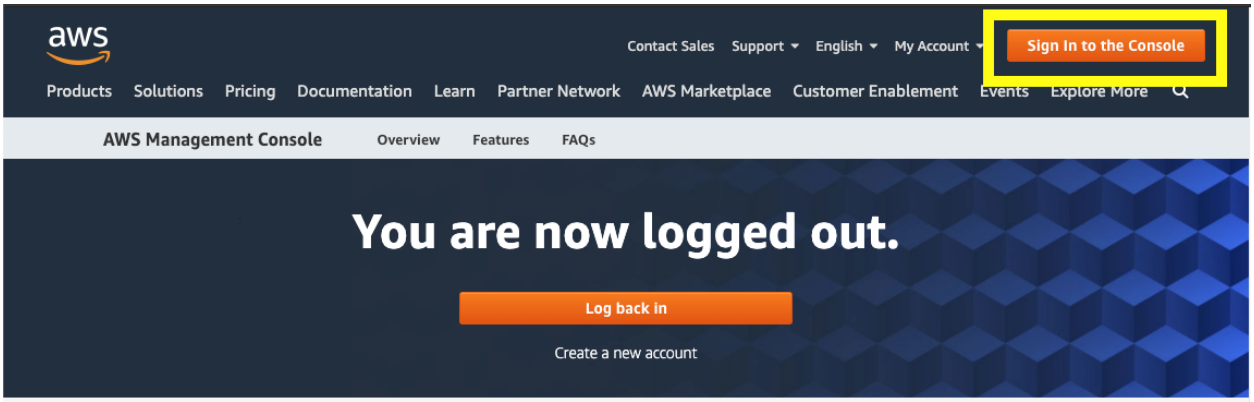
Note

Suppose a student has a quota of 20 instances of `ml.m4.xlarge` by default, they would not notice it unless they run the notebook that uses that instance. Now, if they go to the AWS Support Center, to request a service limit increase by 1, their instance limit will be degraded from 20 to 1.

Recommendation

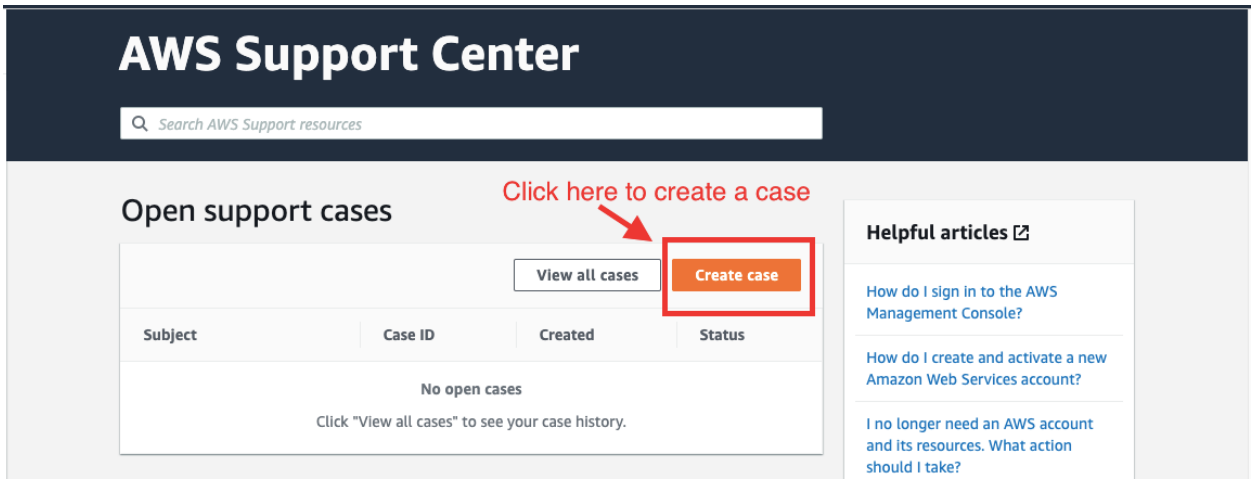
1. For `ml.m4.xlarge`- The default quota would be any number in the range [0 - 20]. Students can expect an error - '*ResourceLimitExceeded*', when executing the notebook in the concept ***Boston Housing Problem - Training The Model***, later in this lesson. In such a case only, the student must request a limit increase for `ml.m4.xlarge`.
2. For `ml.p2.xlarge` - The default quota would be either 0 or 1, therefore it is alright to go ahead and request an increase anytime.

1. Sign in to AWS console - ***https://aws.amazon.com/console/***



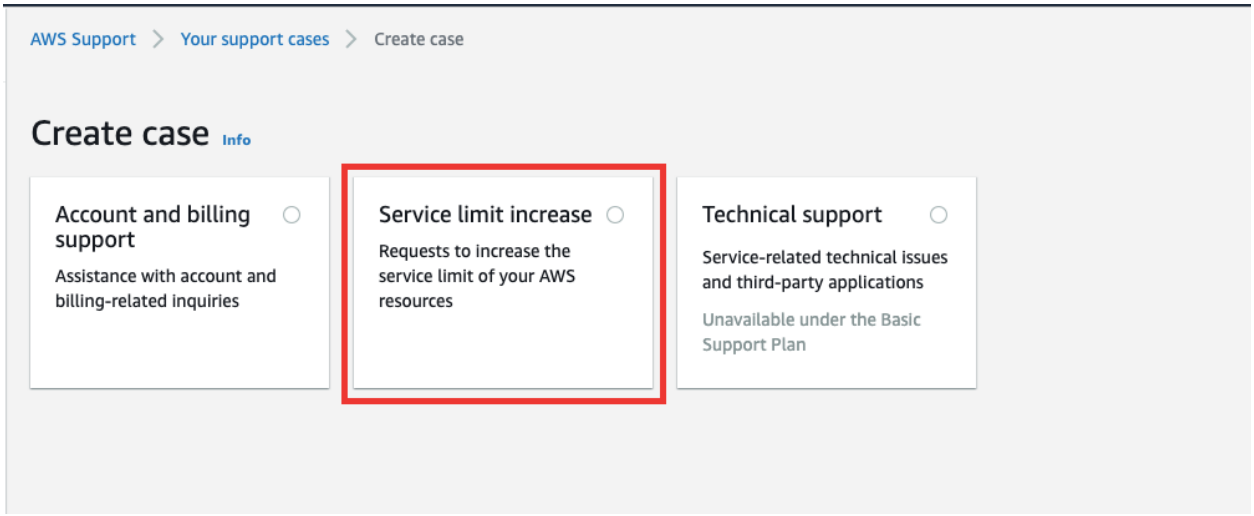
Sign in to AWS console

2. Go to the **AWS Support Center** and create a *case*.



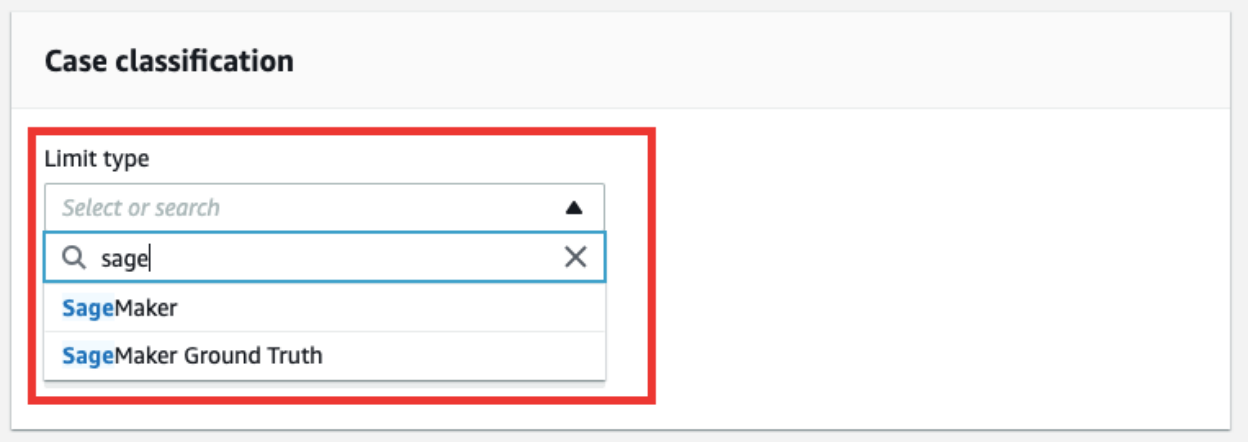
AWS Support Center

3. Click on *Service limit increase*



Create a case for support

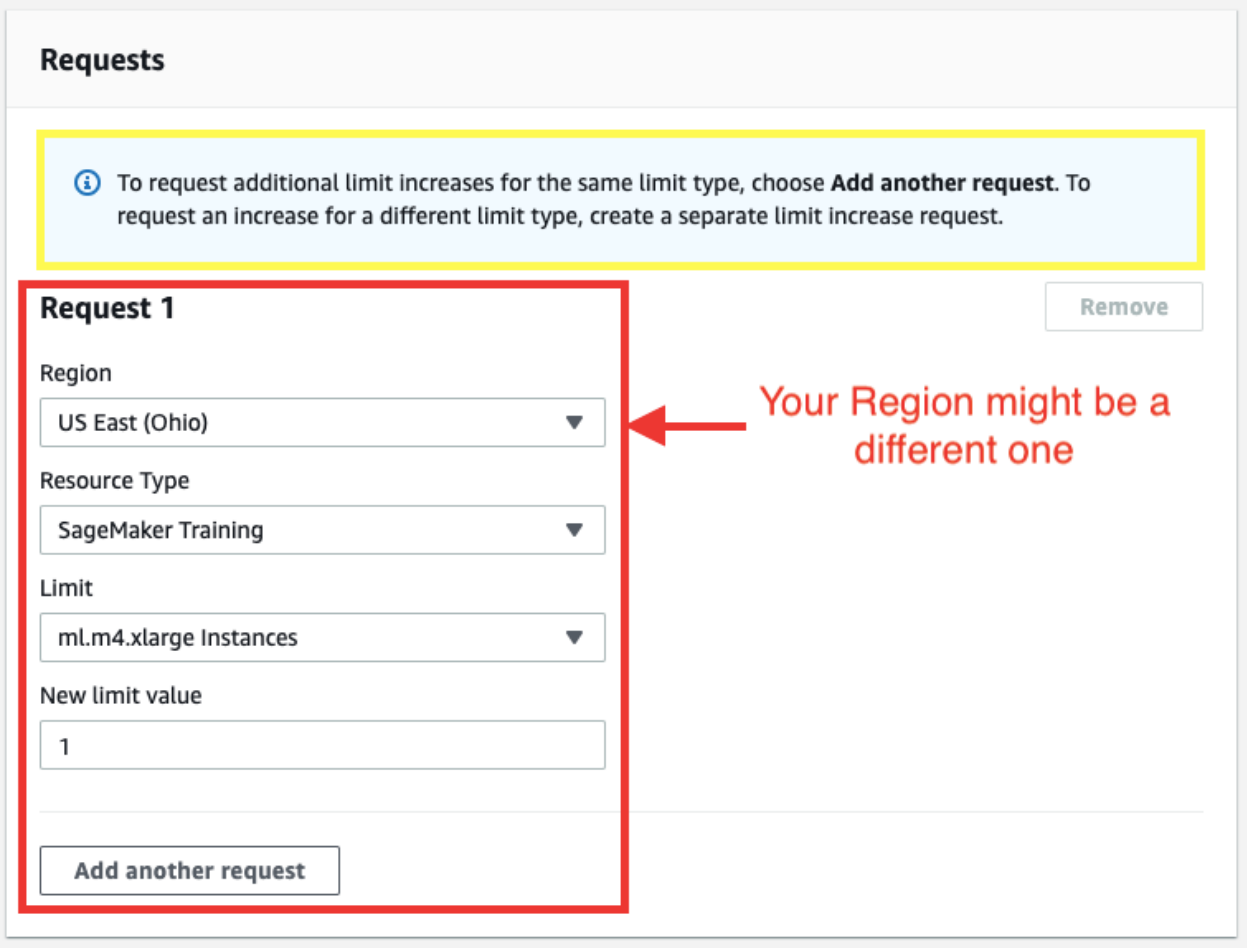
4. It will expand three sections - *Case classification*, *Case description*, and *Contact options* on the same page. In *Case classification* section, select **"Sagemaker"** as the *Limit type*.



Case classification section that takes the Limit type

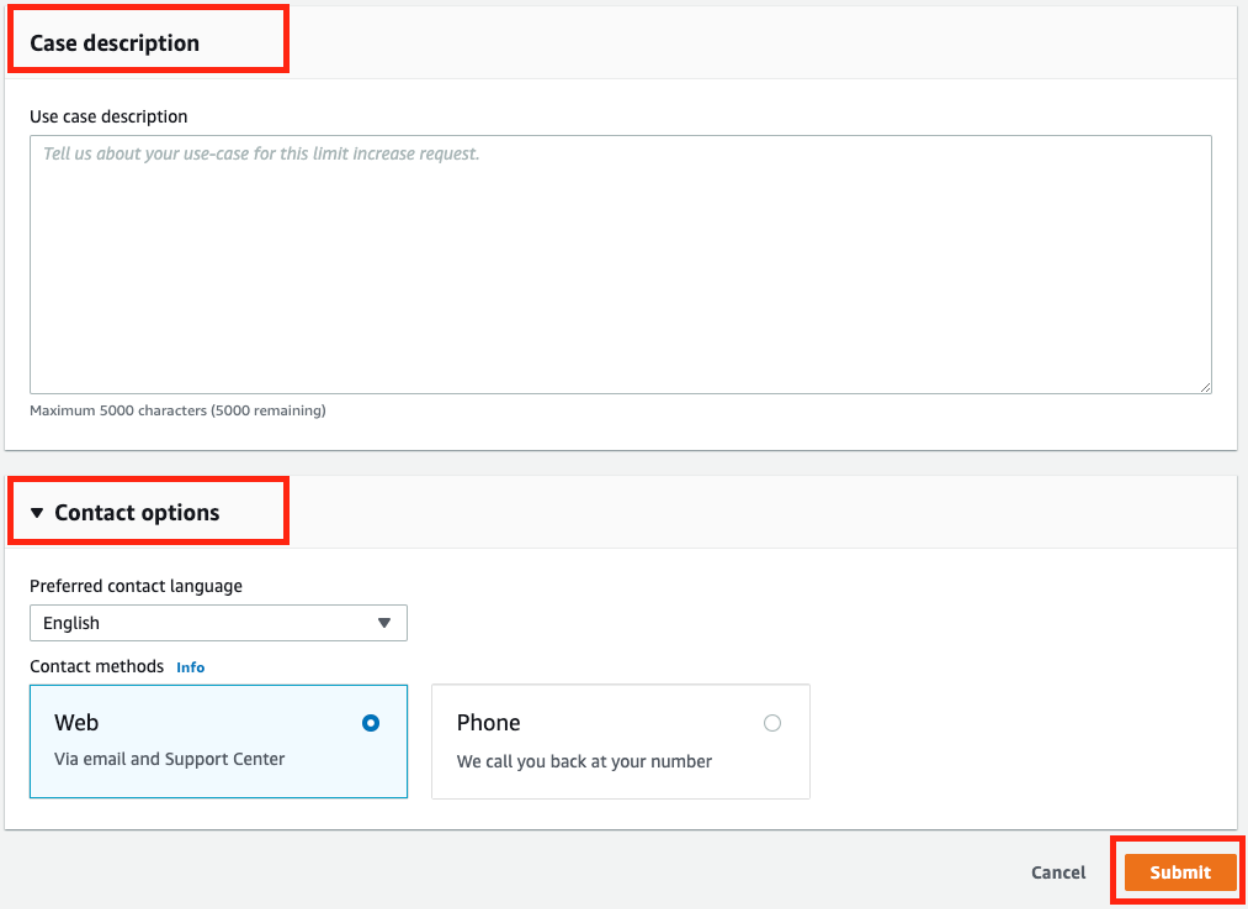
5. It will expand one more section - *Requests* on the same page. In *Request* section, and select the Region in which you are using the SageMaker service.

- Select Sagemaker Training as the Resource Type
- Select the instance type (ml.m4.xlarge or ml.p2.xlarge) under the Limit field
- Under new limit value, select 1



Request section that takes Region, Resource type, and LImit values

6. Provide a case description and the contact options before submitting the case to support.



IMPORTANT NOTICE: This is the current AWS UI as of April 6th, 2020. The AWS UI is subject to change on a regular basis. We advise students to refer to AWS documentation for the above process.

Setting up a Notebook Instance

The first thing we are going to need to do is set up a notebook instance!

This will be the primary way in which we interact with the SageMaker ecosystem. Of course, this is not the *only* way to interact with SageMaker's functionality, but it is the way that we will use in this module.
The video below guides you through setting up your first notebook instance. Also, if you prefer to read the instructions instead, these have been provided underneath the video.

Note: Once a notebook instance has been set up, by default, it will be **InService** which means that the notebook instance is running. This is important to know because the *cost*

of a notebook instance is based on the length of time that it has been running. This means that once you are finished using a notebook instance you should **Stop** it so that you are no longer incurring a cost. Don't worry though, you won't lose any data provided you don't delete the instance. Just start the instance back up when you have time and all of your saved data will still be there.
https://www.youtube.com/watch?v=TRUCNy5Eqjc&feature=emb_logo

Getting the Notebooks

Recently, SageMaker has added a line in the setup code to link directly to a Github repository and it's recommended that you use that setup!

Also on the `Actions` list, you should select `Open Jupyter` to get to the examples notebooks. from the dropdown menu, you'll still be able to see the `Stop` action.

Now that your notebook instance has been set up and is running, it's time to get the notebooks that we will be using during this module.

These notebooks are stored in a **repository on Github** and the easiest way to make them available inside of your notebook instance is to use **git** and clone the repository. The video below goes through this process. Also, if you would prefer to read the instructions, these have been provided and can be found underneath the video.
https://www.youtube.com/watch?v=jqL74whe9yo&feature=emb_logo

Cloning the Deployment Notebooks

In order to clone the deployment repository into your notebook instance, click on the **new** drop down menu and select **terminal**. By default, the working directory of the terminal instance is the home directory, however, the Jupyter notebook hub's root directory is under `SageMaker`. Enter the appropriate directory and clone the repository as follows:

```
cd SageMaker
git clone https://github.com/udacity/sagemaker-deployment.git
exit
```

After you have finished, close the terminal window.

Your notebook instance is now set up and ready to be used!

Has everything been set up correctly?

As you proceed through the lessons that follow you will get a chance to use Amazon's SageMaker service. However, in order to do so you will need to make sure that everything has been set up.

This is meant to be a checklist that you can use to make sure you've completed all of the preparation steps.

- [x] Create an AWS Account
- [x] Apply AWS Credits
- [x] Checked for GPU Access
- [x] Submit a Limit Increase Request (if necessary)
- [x] Create a SageMaker Notebook Instance
- [x] Clone the Deployment Repository (notebooks) into your Notebook Instance

Boston Housing Example

SageMaker Sessions & Execution Roles

SageMaker has some unique objects and terminology that will become more familiar over time. There are a few objects that you'll see come up, over and over again:

- **Session** - A session is a special *object* that allows you to do things like manage data in S3 and create and train any machine learning models; you can read more about the functions that can be called on a session, **at this documentation**. The `upload_data` function should be close to

the top of the list! You'll also see functions like `train`, `tune`, and `create_model` all of which we'll go over in more detail, later.

- **Role** - Sometimes called the *execution role*, this is the IAM role that you created when you created your notebook instance. The role basically defines how data that your notebook uses/creates will be stored. You can even try printing out the role with `print(role)` to see the details of this creation.

Uploading to an S3 Bucket

Another SageMaker detail that is new is the method of data storage. In these instances, we'll be using S3 buckets for data storage.

S3 is a virtual storage solution that is mostly meant for data to be written to few times and read from many times. This is, in some sense, the main workhorse for data storage and transfer when using Amazon services. These are similar to file folders that contain data *and* metadata about that data, such as the data size, date of upload, author, and so on.
S3 stands for Simple Storage Service (S3).
After you upload data to a session, you should see that an S3 bucket is created, as indicated by an output like the following:

```
INFO: sagemaker: Created S3 bucket: <message specific to your locale, ex. sagemaker-u
s-west-1-#>
```

If you'd like to learn more about how we're creating a csv file, you can check out **the pandas documentation**. Above, we are just concatenating x and y data sets as columns of data (`axis=1`) and converting that pandas dataframe into a csv file using `.to_csv`.

Boston Housing Data

For our very first time using SageMaker we will be looking at the problem of estimating the median cost of a house in the Boston area using the **Boston Housing Dataset**.
We will be using this dataset often throughout this module as it provides a great example on which to try out all of SageMaker's features.

In addition, we will be using a random tree model. In particular, we will be using the **XGBoost** algorithm. The details of XGBoost are beyond the scope of this module as we are interested in learning about SageMaker. If you would like to learn more about XGBoost I would recommend starting with the documentation which you can find at **https://xgboost.readthedocs.io/en/latest/**

The notebook we will be working though in this video and in the following two videos can be found in the `Tutorial` directory and is called `Boston Housing - XGBoost (Batch Transform) - High Level.ipynb`. Now that you know why **Boston Housing** and **XGBoost** are in the name, let's talk a bit about the rest of it.

First, **Batch Transform** is the method we will be using to test our model once we have trained it. This is something that we will discuss a little more later on.

Second, **High Level** describes the API we will be using to get SageMaker to perform various machine learning tasks. In particular, it refers to the Python SDK whose documentation can be found here: **https://sagemaker.readthedocs.io/en/latest/**. This high level approach simplifies a lot of the details when working with SageMaker and can be very useful.

https://www.youtube.com/watch?v=78y5cTR-JxM&feature=emb_logo

XGBoost in Competition

There's a **list of winning XGBoost-based solutions** to a variety of competitions, at the linked XGBoost repository.

Estimators

You can read **the documentation on estimators** for more information about this object. Essentially, the Estimator is an object that specifies some details about how a model will be trained. It gives you the ability to create and deploy a model.

Training Jobs

A training job is used to train a specific estimator.

When you request a training job to be executed you need to provide a few items:

1. A location on S3 where your training (and possibly validation) data is stored,
2. A location on S3 where the resulting model will be stored (this data is called the model artifacts),
3. A location of a docker container (certainly this is the case if using a built in algorithm) to be used for training
4. A description of the compute instance that should be used.

Once you provide all of this information, SageMaker will executed the necessary instance (CPU or GPU), load up the necessary docker container and execute it, passing in the location of the training data. Then when the container has finished training the model, the *model artifacts* are packaged up and stored on S3.

You can see a high-level (which we've just walked through) example of training a KMeans estimator, **in this documentation**. This high-level example defines a KMeans estimator, and uses `.fit()` to train that model. Later, we'll show you a low-level model, in which you have to specify many more details about the training job.

Supporting Materials

XGBoost paper

https://www.youtube.com/watch?v=rqYlkCTLmIY&feature=emb_logo

Boston Housing Example - Testing

https://www.youtube.com/watch?v=CZRKuS_qYtg&feature=emb_logo

Transformer

You can read more about the transform and wait functions, in **the transformer documentation**. In this case, the transformer is used to create a transform job and **evaluate a trained model**. The `transform` function takes in the location of some test data, and some information about how that test data is formatted.

Mini-Project: Building Your First Model

Now that you've seen an example of how to use SageMaker, it's *your* turn to try it out! If you look at the deployment **Gitub repository**, inside of the `Mini-Projects` folder is a notebook called `IMDB Sentiment Analysis - XGBoost (Batch Transform).ipynb`. Inside of the notebook are some tasks for you to complete.

As you progress through the notebook you will construct an XGBoost model that tries to determine the sentiment, positive or negative, of a movie review using the **IMDB Dataset**. Moving forward, most of the mini-projects that you will complete throughout this module will use the IMDB dataset.

Note: For the most part, creating this XGBoost model is pretty similar to the Boston Housing example that we just looked at so you can look there if you get stuck. In addition, a solution has been provided and in the next video we will go over my solution to this notebook.

https://www.youtube.com/watch?v=ouLvRqMMbbY&feature=emb_logo

solution: https://www.youtube.com/watch?v=utUxiW-tZrY&feature=emb_logo

Boston Housing In-Depth

Now that we've had a chance to look at how SageMaker is used, let's take a deeper look at what is going on behind the scenes.

In the previous notebooks we looked at, we use the Python SDK to interact with SageMaker, calling this the high-level approach. Now we will look at the low level approach where we describe different tasks we want SageMaker to perform. The documentation for the low level approach can be found in the **Amazon SageMaker Developer Guide**

The notebook that we will be looking at in this video and in the remainder of this lesson is contained in the `Tutorial` folder and is the `Boston Housing - XGBoost (Batch Transform) - Low Level.ipynb` notebook.

You will notice as we go through the details that describing the different tasks we want SageMaker to do can be quite involved. However there is a reason to understand it!

The high level approach makes developing new models very straightforward, requiring very little code. The reason this can be done is that certain decisions have been made for you. The low level approach allows you to be far more particular in how you want the various tasks executed, which is good for when you want to do something a little more complicated.
https://www.youtube.com/watch?v=TA-Ms7djeL0&feature=emb_logo

Boston Housing In-Depth - Creating a Training Job

https://www.youtube.com/watch?time_continue=9&v=1CIbWNUSZXo&feature=emb_logo

Boston Housing In-Depth - Building a Model

https://www.youtube.com/watch?v=JJyVsmcV2M4&feature=emb_logo

Boston Housing In-Depth - Creating a Batch Transform Job

https://www.youtube.com/watch?time_continue=2&v=JwPJMYRl3nw&feature=emb_logo

What have we learned so far?

In this lesson we went over the basics of how models can be constructed and trained using Amazon SageMaker. In addition, we saw some of how SageMaker works and how it interacts with other services.

In particular, we learned how Amazon S3 is used as a central storage service when using SageMaker. In order to train a model, data must first be available on S3, and once the model has been trained, the model artifacts are also stored on S3.

We also saw how to use SageMaker to train models and fit them to data, saving the results (called model artifacts).

Lastly, we looked at how we could use SageMaker's Batch Transform functionality to test our models.

SageMaker Models

What are the main components of a SageMaker model?

Your reflection

The main components are, s3 memory component

Things to think about

In SageMaker, a model is a collection of information that describes how to perform inference. For the most part, this comprises two very important pieces.

The first is the container that holds the model inference functionality. For different types of models this code may be different but for simpler models and models provided by Amazon this is typically the same container that was used to train the model.

The second is the model artifacts. These are the pieces of data that were created during the training process. For example, if we were fitting a linear model then the coefficients that were fit would be saved as model artifacts.

Fitting Models

What happens when a model is fit using SageMaker?

Your reflection

it will transform the test data and calculate accuray

Things to think about

When a model is fit using SageMaker, the process is as follows.

First, a compute instance (basically a server somewhere) is started up with the properties that we specified.

Next, when the compute instance is ready, the code, in the form of a container, that is used to fit the model is loaded and executed. When this code is executed, it is provided access to the training (and possibly validation) data stored on S3.

Once the compute instance has finished fitting the model, the resulting model artifacts are stored on S3 and the compute instance is shut down.

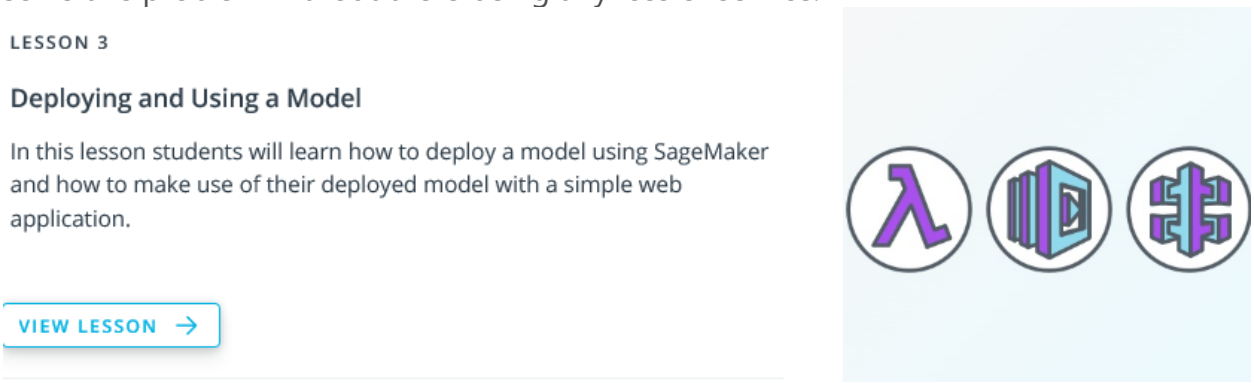
What's next?

In the next few lessons we are going to look at some of the more advanced functionality of SageMaker.

To begin with, we will look at deploying a model using SageMaker. This means making a model available for other entities to use. Along the way we will create a simple web app that interacts with a deployed model.

In addition, we will look at hyperparameter tuning. Which is a way to train a bunch of different models, all with different hyperparameters, and then select the one that performs the best.

Lastly, we will take a look at updating a deployed model. Sometimes a model may not work as well as it once did due to changes in the underlying data. In **this resource**, you can read more about how a model's predictions and accuracy may degrade as a result of something called *concept drift*, which is a change in the underlying data distribution over time. When this happens we might want to update a deployed model, however, our model may be in use so we don't want to shut it down. SageMaker allows us to solve this problem without there being any loss of service.



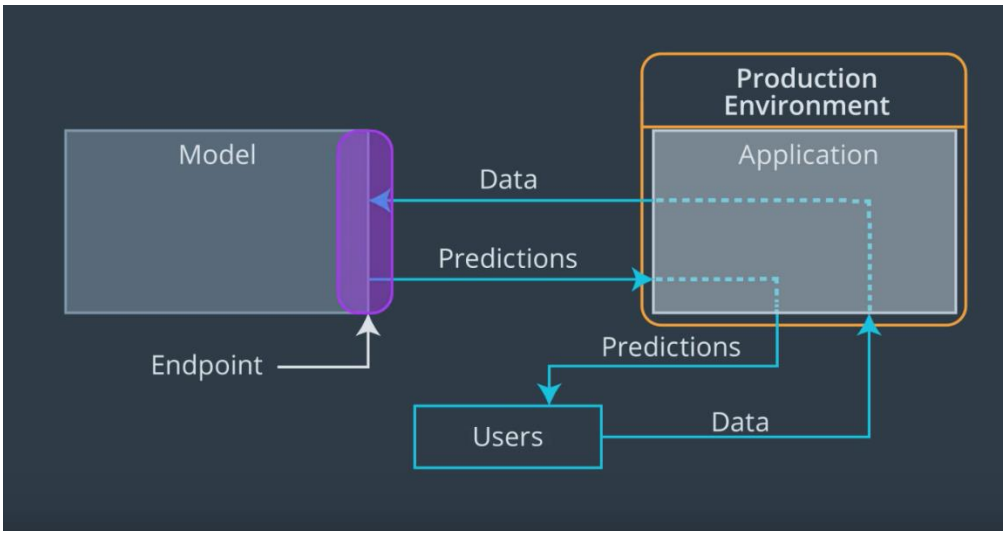
Lesson 3: Deploying a model in SageMaker

Deploying a Model in SageMaker

Deploying a Model in SageMaker

In this lesson, we're going to take a look at how we can use a model that has been created in SageMaker. We will do this by first *deploying* our model. For us, this means using SageMaker's functionality to **create an endpoint that will be used as a way to send data to our model**.

Recall, from the first lesson in this section, that an endpoint is basically a way to allow a model and an application to communicate. An application, such as a web app, will be responsible for accepting user input data, and through an endpoint we can send that data to our model, which will produce predictions that can be sent back to our application!



For our purposes **an endpoint is just a URL**. Instead of returning a web page, like a typical url, this endpoint URL returns the *results* of performing inference. In addition, we

are able to send data to this URL so that our model knows what to perform inference on. We won't go too far into the details of how this is all set up since SageMaker does most of the heavy lifting for us.
An important aspect that we will encounter is that SageMaker endpoints are secured. In this case, that means that only other AWS services with permission to access SageMaker endpoints can do so.

To start with, we won't need to worry about this too much since we will be working inside of a SageMaker notebook and so we will be able to access our deployed endpoints easily.

Later on we will talk about how to set things up so that a simple web app, which doesn't need to be given special permission, can access our SageMaker endpoint.
https://www.youtube.com/watch?v=g_GYZpcVcFE&feature=emb_logo

Boston Housing Example

Now that you've had some time to try and build models using SageMaker, we are going to learn how to deploy those models so that our models can be interacted with using an endpoint.

Inside of the `Tutorials` folder is the `Boston Housing - XGBoost (Deploy) - High Level.ipynb` notebook which we will be looking at in the video below.
Using the high level approach makes deploying a trained model pretty straightforward. All we need to do is call the `deploy` method and SageMaker takes care of the rest. Similarly, sending data to the deployed endpoint and capturing the resulting inference is easy too as SageMaker wraps everything up into the `predict` method, provided we make sure that our data is serialized correctly. In our case, serializing means converting the data structure we wish to send to our endpoint into a string, something that can be transferred using HTTP.
In the next video we'll take a more in-depth look at how our model is being deployed.

WARNING- SHUT DOWN YOUR DEPLOYED ENDPOINT

Sorry for the yelling, but this is pretty important. The cost of a deployed endpoint is based on the length of time that it is running. This means that if you aren't using an endpoint you **really** need to shut it down.

https://www.youtube.com/watch?v=0PBsV-SzSlo&feature=emb_logo

Boston Housing In-Depth

Now we will look at deploying a model using the low level approach. This method requires us to describe the various properties that our endpoint should have and what inference code and model should be used.

To follow along, open up the `Boston Housing - XGBoost (Deploy) - Low Level.ipynb` notebook in the `Tutorials` folder.
Using the low level approach to deploy our model requires us to create an endpoint, which will be used to send data to our model and to get inference results.

In order to create an endpoint in SageMaker, we first need to describe an endpoint configuration. This describes to SageMaker the various properties we want our endpoint to have. Once we've created the endpoint configuration we can ask SageMaker to create an endpoint with the properties we want.

The actual endpoint that is created by SageMaker is a combination of a compute instance (some remote server) running a docker container with the inference code on it and a URL that data can be sent to and returned from. This URL is used as an interface to the compute instance, which receives data, performs inference using our model and returns the result.

https://www.youtube.com/watch?v=1IzWAzypJ9k&feature=emb_logo

Deploying and Using a Sentiment Analysis Model

You've learned how to create and train models in SageMaker *and* how you can deploy them. In this example we are going to look at how we can make use of a deployed model in a simple web app.

In order for our simple web app to interact with the deployed model we are going to have to solve a couple problems.

The first obstacle is something that has been mentioned earlier.

The endpoint that is created when we deploy a model using SageMaker is secured, meaning that only entities that are authenticated with AWS can send or receive data from the deployed model. This is a problem since authenticating for the purposes of a simple web app is a bit more work than we'd like.
So we will need to find a way to work around this.

The second obstacle is that our deployed model expects us to send it a review after it has been processed. That is, it assumes we have already tokenized the review and then created a bag of words encoding. However, we want our user to be able to type any review into our web app.
We will also see how we can overcome this.

To solve these issues we are going to need to use some additional Amazon services. In particular, we are going to look at Amazon Lambda and API Gateway.

In the mean time, I would encourage you to take a look at the `IMDB Sentiment Analysis - XGBoost - Web App.ipynb` notebook in the `Tutorials` folder. In the coming videos we will go through this notebook in detail, however, each of the steps involved is pretty well documented in the notebook itself.

https://www.youtube.com/watch?v=r7XVQEojRKk&feature=emb_logo

Text Processing

I mentioned that one of our tasks will be to convert any user input text into data that our deployed model can see as input. You've seen a few examples of text pre-processing and the steps usually go something like this:

1. Get rid of any special characters like punctuation
2. Convert all text to lowercase and split into individual words

3. Create a vocabulary that assigns each unique word a numerical value or converts words into a vector of numbers
   This last step is often called **word tokenization** or vectorization.
   And in the next example, you'll see exactly how I do these processing steps; I'll also be vectorizing words using a method called **bag of words**. If you'd like to learn more about bag of words, please check out the video below, recorded by another of our instructors, Arpan!

Bag of Words

You can read more about the bag of words model, and its applications, **on this page**.
It's a useful way to represent words based on their frequency of occurrence in a text.

https://www.youtube.com/watch?v=A7M1z8yLl0w&feature=emb_logo

Building and Deploying the Model

Remember to shutdown your endpoint, if you are not going to use it for a while; we'll remind you again at the end of this video.
To begin with, we are going to extend the mini-project that you worked on in the last lesson by deploying it. There are a couple of changes made to the way that data is processed in this version and the reason for this is to simplify some of what follows.

For the most part, however, we simply add on an extra deployment step to the sentiment analysis mini-project and then test that our deployed endpoint is working correctly.

Once this is done we know that we have a sentiment analysis model that has been trained, is performing well and is working, a great place to start!

Don't forget to SHUT DOWN your endpoint!

https://www.youtube.com/watch?v=JCiQhhXbeuc&feature=emb_logo

How to use a Deployed Model

As mentioned earlier, there are two obstacles we are going to need to overcome. The first is the security issue and the second is data processing. The way that we are going to approach solving these issues is by making use of Amazon Lambda and API Gateway.

What this means is that when someone uses our web app, the following will occur.

1. To begin with, a user will type out a review and enter it into our web app.
2. Then, our web app will send that review to an endpoint that we created using API Gateway. This endpoint will be constructed so that anyone (including our web app) can use it.
3. API Gateway will forward the data on to the Lambda function
4. Once the Lambda function receives the user's review, it will process that review by tokenizing it and then creating a bag of words encoding of the result. After that, it will send the processed review off to our deployed model.
5. Once the deployed model performs inference on the processed review, the resulting sentiment will be returned back to the Lambda function.
6. Our Lambda function will then return the sentiment result back to our web app using the endpoint that was constructed using API Gateway.

Don't forget!

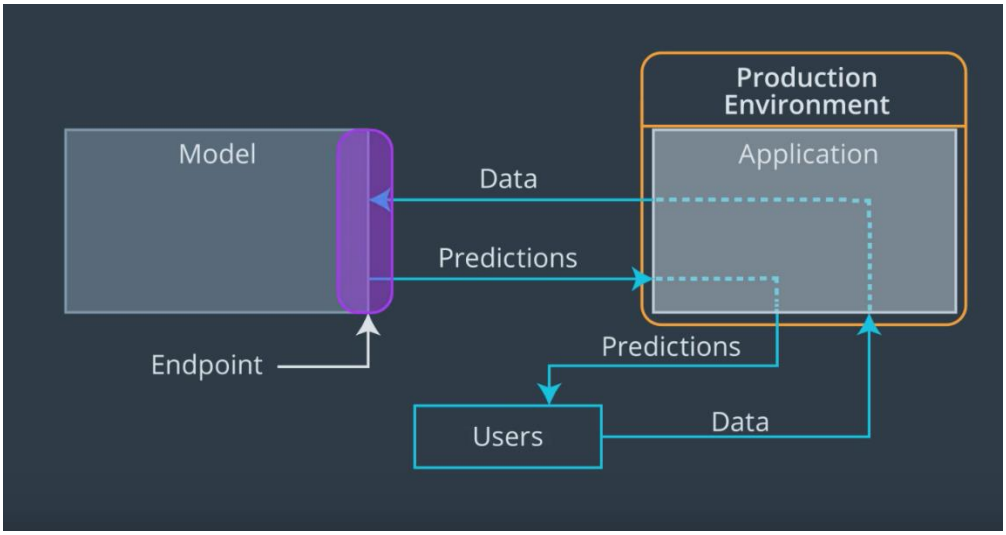
Currently our endpoint is running. The reason for this is that in the next few videos we are going to interact with our deployed endpoint. If you are following along, don't forget that your endpoint is running. If you need to take a break, don't forget to shut down your endpoint!

https://www.youtube.com/watch?v=WTwj-7XcTro&feature=emb_logo

Creating and Using Endpoints

You've just learned a lot about how to use SageMaker to deploy a model and perform inference on some data. Now is a good time to review some of the key steps that we've covered. You have experience processing data and creating estimators/models, so I'll focus on what you've learned about endpoints.

An endpoint, in this case, is a URL that allows an application and a model to speak to one another.



Endpoint steps

- You can start an endpoint by calling `.deploy()` on an estimator and passing in some information about the instance.

```
xgb_predictor = xgb.deploy(initial_instance_count = 1, instance_type = 'ml.m4.xlarge'
)
```

- Then, you need to tell your endpoint, what type of data it expects to see as input (like .csv).

```
from sagemaker.predictor import csv_serializer

xgb_predictor.content_type = 'text/csv'
```

```
xgb_predictor.serializer = csv_serializer
```

- Then, perform inference; you can pass some data as the "Body" of a message, to an endpoint and get a response back!

```
response = runtime.invoke_endpoint(EndpointName = xgb_predictor.endpoint,    # The nam
e of the endpoint we created
                                   ContentType = 'text/csv',
# The data format that is expected
                                   Body = ','.join([str(val) for val in test_bow]
).encode('utf-8'))
```

- The inference data is stored in the "Body" of the response, and can be retrieved:

```
response = response['Body'].read().decode('utf-8')
print(response)
```

- Finally, do not forget to **shut down your endpoint** when you are done using it.

```
xgb_predictor.delete_endpoint()
```

Building a Lambda Function

In general, a Lambda function is an example of a 'Function as a Service'. It lets you perform actions in response to certain events, called triggers. Essentially, you get to describe some events that you care about, and when those events occur, your code is executed.

For example, you could set up a trigger so that whenever data is uploaded to a particular S3 bucket, a Lambda function is executed to process that data and insert it into a database somewhere.

One of the big advantages to Lambda functions is that since the amount of code that can be contained in a Lambda function is relatively small, you are only charged for the *number* of executions.
In our case, the Lambda function we are creating is meant to process user input and interact with our deployed model. Also, the trigger that we will be using is the endpoint that we will create using API Gateway.

Create a Lambda Function

The steps to create a lambda function are outlined in the notebook and here, for convenience.

Setting up a Lambda function The first thing we are going to do is set up a Lambda function. This Lambda function will be executed whenever our public API has data sent to it. When it is executed it will receive the data, perform any sort of processing that is required, send the data (the review) to the SageMaker endpoint we've created and then return the result.

Part A: Create an IAM Role for the Lambda function

Since we want the Lambda function to call a SageMaker endpoint, we need to make sure that it has permission to do so. To do this, we will construct a role that we can later give the Lambda function.

Part B: Create a Lambda function

Now it is time to actually create the Lambda function. Remember from earlier that in order to process the user provided input and send it to our endpoint we need to gather two pieces of information:

Once you have added the endpoint name to the Lambda function, click on Save. Your Lambda function is now up and running!

```
# We need to use the low-level library to interact with SageMaker since the SageMaker
API
# is not available natively through Lambda.
import boto3

# And we need the regular expression library to do some of the data processing
import re

REPLACE_NO_SPACE = re.compile("(\.)|(\;)|(\:)|(\!)|(\')|(\?)|(\,)|(\")|(\()|(\))|(\[)
|(\])")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")
```

```
def review_to_words(review):
    words = REPLACE_NO_SPACE.sub("", review.lower())
    words = REPLACE_WITH_SPACE.sub(" ", words)
    return words

def bow_encoding(words, vocabulary):
    bow = [0] * len(vocabulary) # Start by setting the count for each word in the voc
abulary to zero.
    for word in words.split():  # For each word in the string
        if word in vocabulary:  # If the word is one that occurs in the vocabulary, i
ncrease its count.
            bow[vocabulary[word]] += 1
    return bow


def lambda_handler(event, context):

    vocab = "*** ACTUAL VOCABULARY GOES HERE ***"

    words = review_to_words(event['body'])
    bow = bow_encoding(words, vocab)

    # The SageMaker runtime is what allows us to invoke the endpoint that we've creat
ed.
    runtime = boto3.Session().client('sagemaker-runtime')

    # Now we use the SageMaker runtime to invoke our endpoint, sending the review we
were given
    response = runtime.invoke_endpoint(EndpointName = '***ENDPOINT NAME HERE***',# Th
e name of the endpoint we created
                                       ContentType = 'text/csv',                 # Th
e data format that is expected
                                       Body = ','.join([str(val) for val in bow]).enc
ode('utf-8')) # The actual review

    # The response is an HTTP response whose body contains the result of our inferenc
e
    result = response['Body'].read().decode('utf-8')

    # Round the result so that our web app only gets '1' or '0' as a response.
    result = round(float(result))

    return {
        'statusCode' : 200,
        'headers' : { 'Content-Type' : 'text/plain', 'Access-Control-Allow-Origin' :
'*' },
```

```
        'body' : str(result)
    }
```

Building an API

At this point we've created and deployed a model, and we've constructed a Lambda function that can take care of processing user data, sending it off to our deployed model and returning the result. What we need to do now is set up some way to send our user data to the Lambda function.

The way that we will do this is using a service called API Gateway. Essentially, API Gateway allows us to create an HTTP endpoint (a web address). In addition, we can set up what we want to happen when someone tries to send data to our constructed endpoint.

In our application, we want to set it up so that when data is sent to our endpoint, we trigger the Lambda function that we created earlier, making sure to send the data to our Lambda function for processing. Then, once the Lambda function has retrieved the inference results from our model, we return the results back to the original caller.
https://www.youtube.com/watch?v=AzBQ-aDQSG4&feature=emb_logo

Using the Final Web Application

Now we get to reap the rewards of all our hard work, we get to deploy our web app!

The back end of our app has been set up so at this point all we need to do is finish up the user facing portion, the website itself. To do this we just need to tell our website where it should send data to.

Don't forget!

In order for our web app to work, we need to have our model deployed. This means that we are incurring a cost. So, once you have finished playing with your newly created web app, make sure to shut it down!

You may also want to clean up the endpoint that you constructed and the Lambda function. This isn't too important, however, since each of these services only incur a cost when used.

Some notes on Lambda and Gateway usage

For Lambda functions you are only charged *per execution*, which for this class will be very few and still within the free tier. Deleting a lambda function is just a good cleanup step; you won't be charged if you just leave it there (without executing it). Similarly, for APIs created using API Gateway you are only charged per request, and the number of requests we require in this course should still fall under the free tier.
https://www.youtube.com/watch?v=VgG41Q_a15I&feature=emb_logo

What have we learned so far?

In this lesson we learned how to deploy a model that has been created using SageMaker. We took a look at how to construct endpoints and how to use those endpoints to send data to a deployed model.

In addition, we looked at what we needed to do if we wanted anyone to have access to our deployed model. To make this work we first implemented a Lambda function that took care of data processing and interacting with the model. Then we created an interface through which we could send data to our Lambda function using API Gateway.

How does the data flow?

Describe how the data flows through our sentiment analysis web app.

Your reflection

The training data is first stored locally, and then it is uploaded to s3 for training. When we submit a review through our Webapp, the data is sent in a body via API through the lambda function to the model.

Things to think about

To begin with, the user enters a review on our website.

Next, our website sends that data off to an endpoint, created using API Gateway.

Our endpoint acts as an interface to our Lambda function so our user data gets sent to the Lambda function.

Our Lambda function processes the user data and sends it off to the deployed model's endpoint.

The deployed model perform inference on the processed data and returns the inference results to the Lambda function.

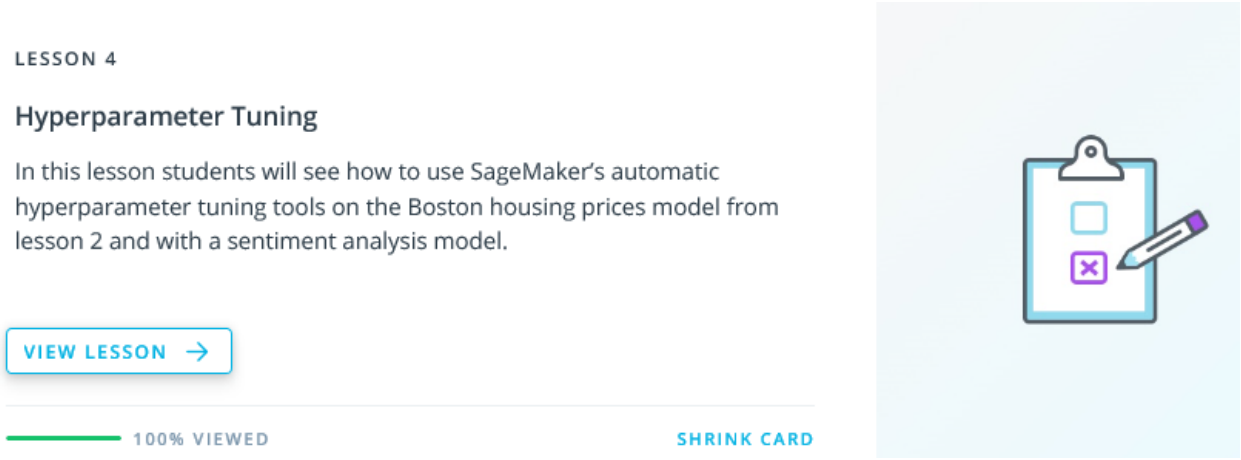
The Lambda function returns the results to the original caller using the endpoint constructed using API Gateway.

Lastly, the website receives the inference results and displays those results to the user.

What's next?

In the next lesson we are going to look at hyperparameter tuning. This is a method by which we can test a variety of different hyperparameters and chose the ones that work best for our data set.

While doing this we will also take a look at CloudWatch, which is a service that allows us to look at the logs generated by the various SageMaker tasks we perform.



Lesson 4: Hyperparameter tuning

Hyperparameter Tuning

In this lesson, we are going to take a look at how we can improve our models using one of SageMaker's features. In particular, we are going to explore how we can use SageMaker to perform hyperparameter tuning.

In many machine learning models there are some parameters that need to be specified by the model creator and which can't be determined directly from the data itself. Generally the approach to finding the best parameters is to train a bunch of models with different parameters and then choose the model that works best.

SageMaker provides an automated way of doing this. In fact, SageMaker also does this in an intelligent way using Bayesian optimization. What we will do is specify ranges for our hyperparameters. Then, SageMaker will explore different choices within those ranges, increasing the performance of our model over time.

In addition to learning how to use hyperparameter tuning, we will look at Amazon's CloudWatch service. For our purposes, CloudWatch provides a user interface through

which we can examine various logs generated during training. This can be especially useful when diagnosing errors.

Next

https://www.youtube.com/watch?v=ohVX3RUTghg&feature=emb_logo

Introduction to Hyperparameter Tuning

Let's take a look at how we can use SageMaker to improve our Boston housing model. To begin with, we will remind ourselves how we train a model using SageMaker. Essentially, tuning a model means training a bunch of models, each with different hyperparameters, and then choosing the best performing model. Of course, we still need to describe two different aspects of hyperparameter tuning:

1) What is a *bunch* of models? In other words, how many different models should we train?
2) Which model is the *best* model? In other words, what sort of metric should we use in order to distinguish how well one model performs relative to another.

https://www.youtube.com/watch?v=nah8kxqp55U&feature=emb_logo

Boston Housing Example

In the video below we will look at how we can set up a hyperparameter tuning job using the high level approach in SageMaker. To following along, open up the `Boston Housing - XGBoost (Hyperparameter Tuning) - High Level.ipynb` notebook in the `Tutorials` directory.

Generally speaking, the way to think about hyperparameter tuning inside of SageMaker is that we start with a **base** collection of hyperparameters which describe a default model. We then give some additional set of hyperparameters ranges. These ranges tell SageMaker which hyperparameters can be varied, with the goal being to improve the default model.

We then describe how to compare models, which in our instance is just by way of specifying a metric. Then we describe how many total models we want SageMaker to train.

Note: In addition to creating a tuned model in this notebook, we also saw how the `attach` method can be used to create an `Estimator` object which is attached to an already completed training job. This method is useful in other situations as well.
You will notice that throughout this module we train the same model multiple times. In most of the Boston Housing notebooks, for example, we train an XGBoost model with the same hyperparameters. The reason for this is so that each notebook is self contained and can be run even if you haven't run the other notebooks.

In *your* case however, you've probably already created an XGBoost model on the Boston Housing training set with the standard hyperparameters. If you wanted to, you could use the `attach` method to avoid having to re-train the model.
https://www.youtube.com/watch?v=lsYRtKivrGc&feature=emb_logo

Mini-Project: Tuning the Sentiment Analysis Model

Now that you've seen an example of how to use SageMaker to tune a model, it's *your* turn to try it out!
Inside of the `Mini-Projects` folder is a notebook called `IMDB Sentiment Analysis - XGBoost (Hyperparameter Tuning).ipynb`. Inside of the notebook are some tasks for you to complete.
Note: To make things a little more interesting, there is a small error in the notebook. Try not to anticipate where the error is. Instead, just continue through the notebook as if nothing is wrong and when an error occurs, try to use CloudWatch to diagnose and fix it.
https://www.youtube.com/watch?v=7XORMSX7vAY&feature=emb_logo

sol:
https://www.youtube.com/watch?v=Q2Vthdca49I&feature=emb_logo

Mini-Project: Solution - Fixing the Error and Testing

Machine Learning Engineer Nanodegree Material

https://www.youtube.com/watch?time_continue=17&v=i-EjGkZ8z30&feature=emb_logo

Boston Housing In-Depth - Creating a Tuning Job

https://www.youtube.com/watch?v=vlsZ-jC5C8Y&feature=emb_logo

Now we will look at creating a hyperparameter tuning job using the low level approach. Just like in the other low level approaches, this method requires us to describe the various properties that our hyperparameter tuning job should have.

To follow along, open up the `Boston Housing - XGBoost (Hyperparameter Tuning) - Low Level.ipynb` notebook in the `Tutorials` folder.
Creating a hyperparameter tuning job using the low level approach requires us to describe two different things.

The first, is a training job that will be used as the **base** job for the hyperparameter tuning task. This training job description is almost exactly the same as the standard training job description except that instead of specifying **HyperParameters** we specify **StaticHyperParameters**. That is, the hyperparameters that we do **not** want to change in the various iterations.
The second thing we need to describe is the tuning job itself. This description includes the different ranges of hyperparameters that we **do** want SageMaker to vary, in addition to the total number of jobs we want to have created and how to determine which model is best.

Boston Housing In-Depth - Monitoring the Tuning Job

https://www.youtube.com/watch?time_continue=33&v=WXjIkSHYEyM&feature=emb_logo

Boston Housing In-Depth - Building and Testing the Model

https://www.youtube.com/watch?v=ap7d7DZL0Ic&feature=emb_logo

What have we learned so far?

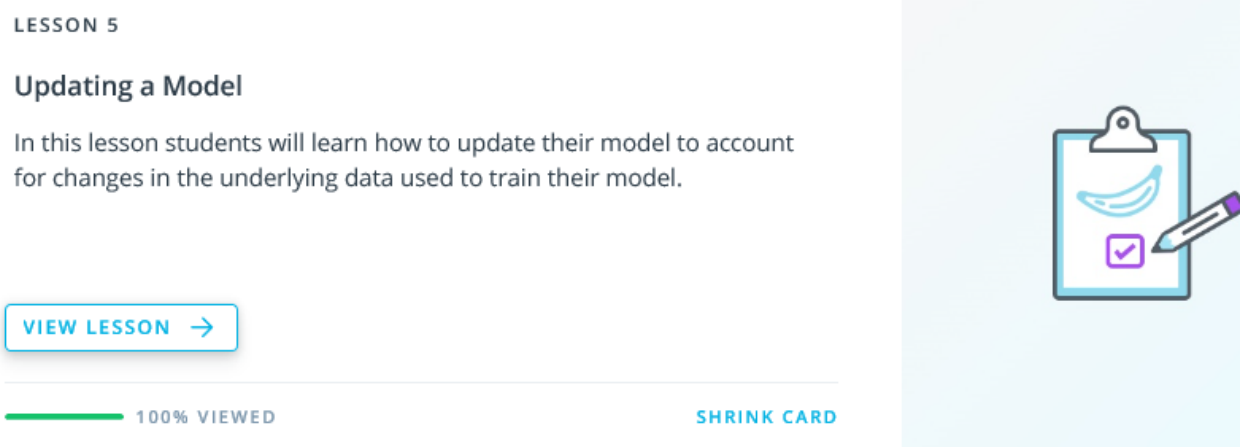
In this lesson we took a look at how we can use SageMaker to tune a model. This can be helpful once we've found a good base model and we want to try and iterate a bit to refine our model and get a little more out of it.

We also looked at using CloudWatch to monitor our training jobs so that we can better diagnose errors when they arise. This can be especially helpful when training more complicated models such as those in which you can incorporate your own code.

What's next?

In the next lesson we will take a look at updating a deployed model. Once you've developed a model and deployed it the story is rarely over. What happens if some of the built in assumptions about your model change over time?

We will look at how you can create a new model which more accurately reflects the current state of your problem and then update an existing endpoint so that it uses your new model rather than the original one. In addition, using SageMaker, we can do this without needing to shut down the endpoint. This means that whatever application is using your deployed model won't notice any sort of disruption in service.



Lesson 5: Updating a Model

In this lesson we are going to take a look at updating an existing endpoint so that it conforms to a different endpoint configuration. There are many reasons for wanting to do this, the two that we will look at are, performing an A/B test and updating a model which is no longer performing as well.

To start, we will look at performing an A/B test between two different models. Then, once we've decided on a model to use, updating the existing endpoint so that it only sends data to a single model.

For the second example, it may be the case that once we've built a model and begun using it, the assumptions on which our model is built begin to change.

For instance, in the sentiment analysis examples that we've looked at our models are based on a vocabulary consisting of the 5000 most frequently appearing words in the training set. But what happens if, over time, the usage of words changes? Then our model may not be as accurate.

When this happens we may need to modify our model, often this means re-training it. When we do, we'd like to update the existing endpoint without having to shut it down. Fortunately, SageMaker allows us to do this in a straightforward way.

https://www.youtube.com/watch?v=7wI168JzBiU&feature=emb_logo

Building a Sentiment Analysis Model

To begin with we will create an XGBoost model similar to the ones that we have constructed in the past in order to predict the median housing cost in Boston.

The difference this time is that we are using a hybrid approach, including both the high level and low level functionality. In this case we use the high level approach to train a model (to produce model artifacts) and then we use the low level approach to construct

the model itself and to construct the endpoint configuration. The reason for this is so that we can have more control over how our endpoint behaves.
https://www.youtube.com/watch?v=dwRkA0ig3uU&feature=emb_logo

Building a Sentiment Analysis Model

Depending on the application you have in mind for a particular machine learning model, accuracy may not always be the metric you wish to optimize. There may be some other constraints on getting the model to work in production. For example, your model may not be very easy to interpret or maybe performing inference for a particular model may be too costly.

In any case you may want to try alternative models. In the example we are working on here we construct a linear learner model as an alternative to the previously created XGBoost model.

Note: It is important to notice that the result returned by the linear learner model is json, compared to the csv data returned by the XGBoost model. You can't always assume that different models will return data in the same way although typically the return type is specified in the documentation.
https://www.youtube.com/watch?v=7TdiVF6qS1k&feature=emb_logo

Combining the Models

Using the low level approach to creating endpoint configurations allows us to create endpoints that are more sophisticated. For example, endpoints which receive data and route that data to one of many different models. In the example here we are only using two different models but there may be situations in which you would want more.

In addition, SageMaker provides functionality to update an existing endpoint so that it conforms to a different endpoint configuration. Further, SageMaker does this in a way that does **not** require the existing endpoint to be shut down.
This is very beneficial as you may be working in an environment where there are other services that depend on your deployed endpoint.
https://www.youtube.com/watch?v=OYYJerDHu0o&feature=emb_logo

Mini-Project: Updating a Sentiment Analysis Model

In this mini-project we will take a look at situation in which we have a trained model which is working well, but then something changes with the underlying distribution on which our model is based. First we need to take a look at what might be the problem. Then we want to create a new, updated model and replace our old model without taking down the corresponding endpoint.

This mini-project notebook is called `IMDB Sentiment Analysis - XGBoost (Updating a Model).ipynb` and can be found inside of the `Mini-Projects` folder.
https://www.youtube.com/watch?v=v7dYwxuKXzI&feature=emb_logo

Loading and Testing the New Data

https://www.youtube.com/watch?time_continue=27&v=75RxW3R6674&feature=emb_logo

Exploring the New Data

https://www.youtube.com/watch?time_continue=2&v=sEBK1dmiUfE&feature=emb_logo

Building a New Model

https://www.youtube.com/watch?time_continue=42&v=RUVxrKcWAsU&feature=emb_logo

SageMaker Retrospective

In this module we looked at various features offered by Amazon's SageMaker service. These features include the following.

- **Notebook Instances** provide a convenient place to process and explore data in addition to making it very easy to interact with the rest of SageMaker's features.
- **Training Jobs** allow us to create *model artifacts* by fitting various machine learning models to data.
- **Hyperparameter Tuning** allow us to create multiple training jobs each with different hyperparameters in order to find the hyperparameters that work best for a given problem.
- **Models** are essentially a combination of *model artifacts* formed during a training job and an associated docker container (code) that is used to perform inference.
- **Endpoint Configurations** act as blueprints for endpoints. They describe what sort of resources should be used when an endpoint is constructed along with which models should be used and, if multiple models are to be used, how the incoming data should be split up among the various models.
- **Endpoints** are the actual HTTP URLs that are created by SageMaker and which have properties specified by their associated endpoint configurations. **Have you shut down your endpoints?**
- **Batch Transform** is the method by which you can perform inference on a whole bunch of data at once. In contrast, setting up an endpoint allows you to perform inference on small amounts of data by sending it do the endpoint bit by bit.
  In addition to the features provided by SageMaker we used three other Amazon services.

  In particular, we used **S3** as a central repository in which to store our data. This included test / training / validation data as well as model artifacts that we created during training. We also looked at how we could combine a deployed SageMaker endpoint with **Lambda** and **API Gateway** to create our own simple web app.
  https://www.youtube.com/watch?v=Vdacqn_w-e4&feature=emb_logo

Cleaning Up Your AWS Account

Once you have finished making use of Amazon's services you should make sure to clean up your account. One of the main reasons for this is so that you don't get an unexpected bill!

Machine Learning Engineer Nanodegree Material

https://www.youtube.com/watch?v=8z24cb3EfMc&feature=emb_logo

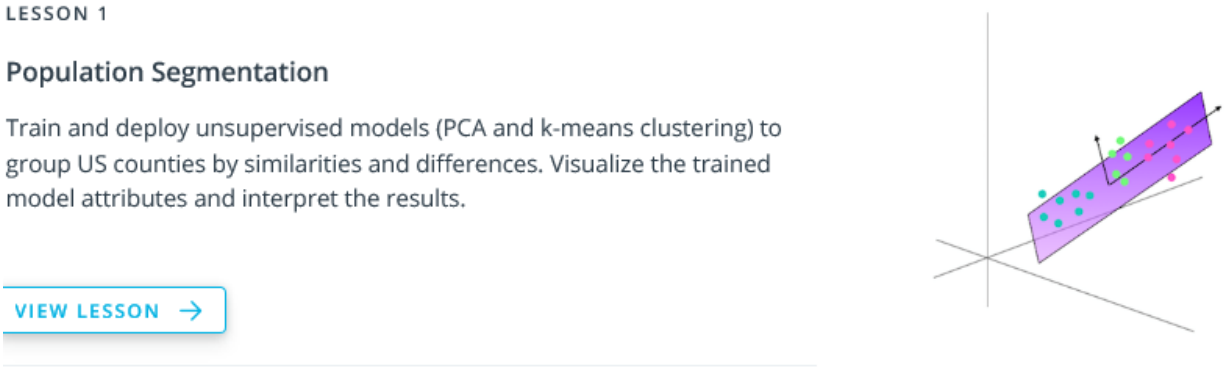
Tips and Tricks

SageMaker Documentation

- **Developer Documentation** can be found here: https://docs.aws.amazon.com/sagemaker/latest/dg/
- **Python SDK Documentation** (also known as the high level approach) can be found here: https://sagemaker.readthedocs.io/en/latest/
- **Python SDK Code** can be found on github here: https://github.com/aws/sagemaker-python-sdk
  https://www.youtube.com/watch?v=ilnX9rUlV_w&feature=emb_logo

MACHINE LEARNING CASE STUDIES

Lesson 1: Population Segmentation



Expert Interview: AWS SageMaker

In these exclusive interview segments, learn about SageMaker and how it is applied to real-world use cases. One of the values that SageMaker and Udacity share is that they

want to make machine learning **accessible**. We do this through education, and they do it through making tools and scalable infrastructure available to learners and engineers. Later in this course, you'll learn about how SageMaker has developed over time and hear some predictions about the future of ML-powered technology.

Please view the segments that seem interesting to you!

How do you define SageMaker?

https://www.youtube.com/watch?time_continue=3&v=JWRtWcd92E4&feature=emb_logo

What applications does SageMaker make possible?

https://www.youtube.com/watch?time_continue=1&v=iXN30g70PJ0&feature=emb_logo

Why should students gain skills in SageMaker and cloud services?

https://www.youtube.com/watch?v=Hp6qTdiqU3g&feature=emb_logo

Course Outline

Throughout this course, we'll be focusing on deployment tools and the machine learning workflow; answering a few big questions along the way:

- How do you decide on the correct machine learning algorithm for a given task?
- How can we utilize cloud ML services in SageMaker to work with interesting datasets or improve our algorithms?

To approach these questions, we'll go over a number of real-world **case studies**, and go from task and problem formulation to deploying models in SageMaker. We'll also utilize a number of SageMaker's built-in algorithms.

Case Studies

Case studies are in-depth examinations of specific, real-world tasks. In our case, we'll focus on three different problems and look at how they can be solved using various machine learning strategies. The case studies are as follows:

Case Study 1 - Population Segmentation using SageMaker

You'll look at a portion of US census data and, using a combination of unsupervised learning methods, extract meaningful components from that data and group regions by similar census-recorded characteristics. This case study will be a deep dive into Principal Components Analysis (PCA) and K-Means clustering methods, and the end result will be groupings that are used to inform things like localized marketing campaigns and voter campaign strategies.

Case Study 2 - Detecting Credit Card Fraud

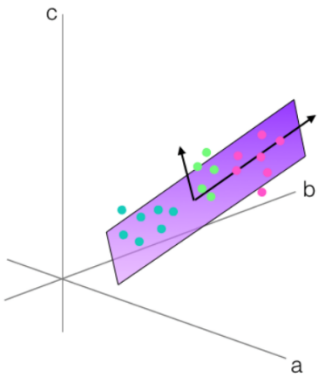
This case will demonstrate how to use supervised learning techniques, specifically SageMaker's `LinearLearner`, for fraud detection. The payment transaction dataset we'll work with is unbalanced, with many more examples of valid transactions vs. fraudulent, and so you will investigate methods for compensating for this imbalance and tuning your model to improve its performance according to a specific product goal.

Custom Models - Non-Linear Classification

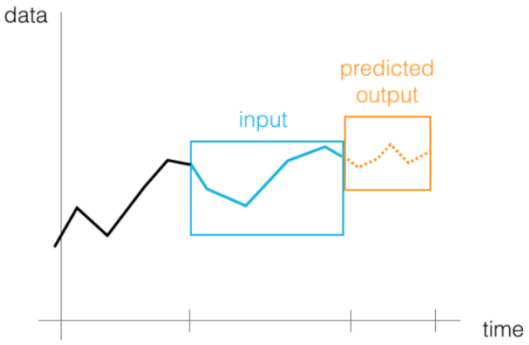
Adding on to what you have learned in the credit card fraud case study, you will learn how to manage cases where classes of data are not separable by a linear line. You'll train and deploy a custom, PyTorch neural network for classifying data.

Case Study 3 - Time-Series Forecasting

This case demonstrates how to train SageMaker's DeepAR model for forecasting predictions over time. Time-series forecasting is an active area of research because a good forecasting algorithm often takes in a number of different features and accounts for seasonal or repetitive patterns. In this study, you will learn a bit about creating features out of time series data and formatting it for training.



Clustering & Dimensionality Reduction



Time Series Forecasting

Project: Plagiarism Detection

You'll apply the skills that you've learned to a final project; building and deploying a plagiarism classification model. This project will test your ability to do [text] data processing and feature extraction, your ability to train and evaluate models according to an accuracy specification, and your ability to deploy a trained model to an endpoint.

By the end of this course, you should have all the skills you need to build, train and deploy models to solve tasks of your own design!

Unsupervised v Supervised Learning

https://www.youtube.com/watch?v=9M6T9Bx3oNA&feature=emb_logo

Model Design

https://www.youtube.com/watch?v=zxNoSTZ3s90&feature=emb_logo

Population Segmentation

https://www.youtube.com/watch?v=3pXFLrnk7q0&feature=emb_logo

K-Means Clustering

To perform population segmentation, one of our strategies will be to use k-means clustering to group data into similar clusters. To review, the k-means clustering algorithm can be broken down into a few steps; the following steps assume that you have n-dimensional data, which is to say, data with a discrete number of features associated with it. In the case of housing price data, these features include traits like house size, location, etc. **features** are just measurable components of a data point. K-means works as follows:

You select `k`, a predetermined number of clusters that you want to form. Then `k` points (centroids for k clusters) are selected at random locations in feature space. For each point in your training dataset:

1. You find the centroid that the point is closest to
2. And assign that point to that cluster
3. Then, for each cluster centroid, you move that point such that it is in the center of *all* the points that are were assigned to that cluster in step 2.
4. Repeat steps 2 and 3 until you've either reached convergence and points no longer change cluster membership _or_ until some specified number of iterations have been reached.

This algorithm can be applied to any kind of unlabelled data. You can watch a video explanation of the k-means algorithm, as applied to color image segmentation, below. In this case, the k-means algorithm looks at R, G, and B values as features, and uses those features to cluster individual pixels in an image!

Color Image Segmentation

Data Dimensionality

One thing to note is that it's often easiest to form clusters when you have low-dimensional data. For example, it can be difficult, and often noisy, to get good clusters from data that has over 100 features. In high-dimensional cases, there is often a

dimensionality reduction step that takes place *before* data is analyzed by a clustering algorithm. We'll discuss PCA as a dimensionality reduction technique in the practical code example, later.

https://www.youtube.com/watch?v=Cf_LSDCEBzk&feature=emb_logo

AWS Console & SageMaker Notebooks

The majority of work that you do in this course will be completed using Amazon SageMaker. For each case study, you will be expected to work through a SageMaker notebook instance, which has access to all the libraries we need as well as SageMaker-specific data management and deployment tools!

If you are familiar with the AWS console and SageMaker, you may proceed to the next video to get set up. If not, it is suggested that you read through these instructions carefully to setup an AWS account.

1. Create an AWS Account

First, visit Amazon AWS and click on the **Create an AWS Account** button.

- If you have an AWS account already, sign in.
- If you do not have an AWS account, sign up.

When you sign up, you will need to provide a credit card. But don't worry, you won't be charged for anything yet. Furthermore, when you sign up, you will also need to choose a support plan.

*You can choose the free **Basic Support Plan**.*

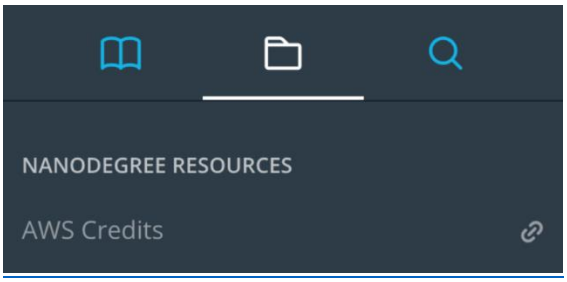
Once you finish signing up, wait a few minutes to receive your AWS account confirmation email. Then return to Amazon AWS and sign in.

The Console

In the future, you can access the AWS console by visiting console.aws.amazon.com. The AWS console is a central hub from which you can access all of the various Amazon Web Services, including SageMaker.

2. Apply AWS Credits

All students are provided with AWS credits to train their neural networks. To get your AWS credits, go to the 'Resources' tab in the left of the classroom; there will be an 'AWS Credits' link to click there.



AWS credits in the "Resources" tab

Click on the 'Go to AWS' button to request your credits. This should bring up a form which you will need to complete.

In your AWS account, ***your AWS Account ID can be found under 'My Account',*** *which is itself found in the dropdown under the name of your account, between the bell and the 'Global' dropdown.*

After you've gone through all the steps, you'll receive an email with your code. The email will be sent to the email address you entered on the AWS credits application. It may take up to 48 hours to receive this email, though it is much quicker in most cases

Under "AWS Promotional Credit " in the email, you'll find your code. Use this code on the link provided, which is your account credits page.

3. Set up a Notebook Instance

Then, you'll need to set up a notebook instance that is connected to the main Github repository for this course. The notebook instance will be the primary way in which we interact with the SageMaker ecosystem.

Watch the next video to see how to set up this notebook instance.

4. Clean up (Eventually)

One thing that I want to note, is that you'll want to intermittently clean up your SageMaker resources. SageMaker notebooks, models, and endpoints may cost some AWS credits to run and store, and as we go through these notebooks, I will do my best to remind you to **delete any resources that you are no longer usin**

The Github Repository

You can find a link to all of the exercise and project code for this course in the repository: https://github.com/udacity/ML_SageMaker_Studies. Copy and paste this repository into the Github clone option when you create your notebook instance!
Note: Once a notebook instance has been set up, by default, it will be **InService** which means that the notebook instance is running. This is important to know because the *cost*

of a notebook instance is based on the length of time that it has been running. This means that once you are finished using a notebook instance you should **Stop** it so that you are no longer incurring a cost. Don't worry though, you won't lose any data provided you don't delete the instance. Just start the instance back up when you have time and all of your saved data will still be there.

https://www.youtube.com/watch?v=w2GBAnhUlOw&feature=emb_logo

Notebook: Population Segmentation, Exercise

Now, you're ready to approach the task of population segmentation! As you follow along with this lesson, you are encouraged to open the referenced SageMaker notebooks. We will present a solution to you, but please try to work on a solution of your own, when prompted. Much of the value in this experience will come from experimenting with the code, **in your own way**.
To open this notebook:

- *Navigate to your SageMaker notebook instance, in the SageMaker console, which has been linked to the main Github exercise repository*
- *Activate the notebook instance (if it is in a "Stopped" state), and open it via Jupyter*
- *Click on the exercise notebook in the* `Population_Segmentation` *directory.*

  You may also directly view the exercise and solution notebooks via the repository at the following links:

- Exercise notebook
- Solution notebook

  The solution notebook is meant to be consulted if you are stuck or want to check your work.

Notebook Outline

We'll go over the following steps to complete the notebook.

- Load in and explore population data
- Perform dimensionality reduction with a deployed PCA model
- Cluster components with K-Means
- Visualize the results

Later: Delete Resources

At the end of this exercise, and intermittently, you will be reminded to delete your endpoints and resources so that you do not incur any extra processing or storage fees!

Exercise: Data Loading & Processing

https://www.youtube.com/watch?time_continue=8&v=YlG9T17KcbU&feature=emb_logo

Solution: Data Pre-Processing

https://www.youtube.com/watch?time_continue=1&v=2jUouM70A1I&feature=emb_logo

Solution: Normalization

https://www.youtube.com/watch?time_continue=23&v=UDWwdG4e1a0&feature=emb_logo

PCA

Principal Component Analysis (PCA) attempts to reduce the number of features within a dataset while retaining the "principal components", which are defined as *weighted* combinations of existing features that:

1. Are uncorrelated with one another, so you can treat them as independent features, and
2. Account for the largest possible variability in the data!

So, depending on how many components we want to produce, the first one will be responsible for the largest variability on our data and the second component for the second-most variability, and so on. Which is exactly what we want to have for clustering purposes!

PCA is commonly used when you have data with many *many* features.
You can learn more about the details of the PCA algorithm in the video, below.

https://www.youtube.com/watch?v=uyl44T12yU8&feature=emb_logo

Now, in our case, we have data that has 34-dimensions and we'll want to use PCA to find combinations of features that produce the most variability in the population dataset. The idea is that components that cause a larger variance will help us to better differentiate between data points and (therefore) better separate data into clusters.

So, next, I'll go over how to use **SageMaker's built-in PCA model** to analyze our data.

PCA Estimator & Training

https://www.youtube.com/watch?time_continue=34&v=HGEqgi2MKcU&feature=emb_logo

Solution: Variance

https://www.youtube.com/watch?time_continue=6&v=C-BRBjxlUuE&feature=emb_logo

Component Makeup

https://www.youtube.com/watch?time_continue=1&v=fiSr_Xjm3qI&feature=emb_logo

Solution: Creating Transformed Data

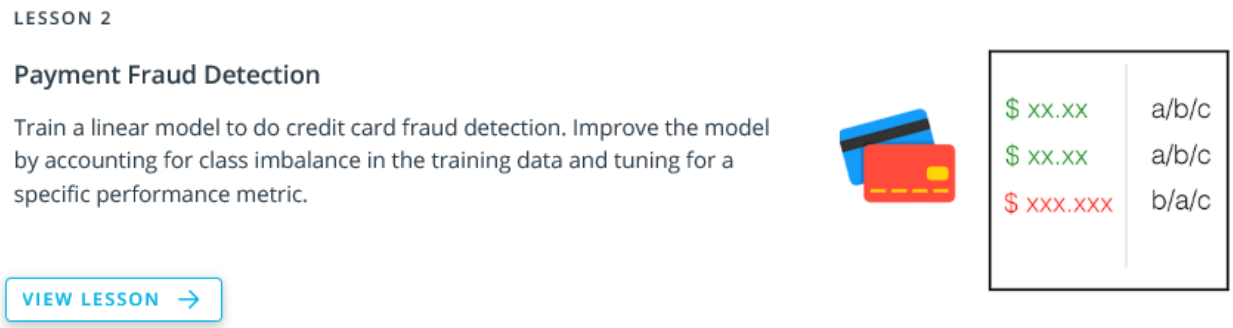
https://www.youtube.com/watch?time_continue=12&v=4l2UHyyVV7Y&feature=emb_logo

Solution: K-means Predictor

https://www.youtube.com/watch?time_continue=13&v=0xx2p2vnCg0&feature=emb_logo

Solution: Model Attributes

https://www.youtube.com/watch?time_continue=23&v=VS-hVhsCBPw&feature=emb_logo



Lesson 2: Payment Fraud detection

Fraud Detection

https://www.youtube.com/watch?time_continue=8&v=zDnyR5Tci5M&feature=emb_logo

Notebook: Fraud Detection, Exercise

Next, you'll approach the task of payment fraud detection! This is a real-world problem, with fraud accounting for billions of dollars worth of loss, worldwide. As you follow along with this lesson, you should work in the referenced SageMaker notebooks. We will present a solution to you, but please try to work on a solution of your own, when prompted. Much of the value in this experience will come from experimenting with the code, **in your own way**.

To open this notebook:

- Navigate to your SageMaker notebook instance, in the **SageMaker console**, which has been linked to the main **Github exercise repository**
- Activate the notebook instance (if it is in a "Stopped" state), and open it via Jupyter
- Click on the exercise notebook in the `Payment_Fraud_Detection` directory.

You may also directly view the exercise and solution notebooks via the repository at the following links:

- **Exercise notebook**
- **Solution notebook**

The solution notebook is meant to be consulted if you are stuck or want to check your work.

Notebook Outline

We'll go over the following steps to complete the notebook.

- Load in and explore payment transaction data
- Train a LinearLearner to classify the data
- Improve a basic model by accounting for class imbalance in the dataset and different metrics for model "success"

Later: Delete Resources

At the end of this exercise, and intermittently, you will be reminded to delete your endpoints and resources so that you do not incur any extra processing or storage fees!

Solution: Data Distribution & Splitting

https://www.youtube.com/watch?time_continue=11&v=Cjn82LqTB00&feature=emb_logo

LinearLearner & Class Imbalance

https://www.youtube.com/watch?time_continue=6&v=pjs5pP9OOMc&feature=emb_logo

Instantiate a `LinearLearner`

Now that you've uploaded your training data, it's time to define and train a model! In the main exercise notebook, you'll define and train the SageMaker, built-in algorithm, `LinearLearner`.

EXERCISE: Create a LinearLearner Estimator

You've had some practice instantiating built-in models in SageMaker. All estimators require some constructor arguments to be passed in.

See if you can complete this task, instantiating a LinearLearner estimator, using only the **LinearLearner documentation** as a resource.
You'll find that this estimator takes in a lot of arguments, but not all are *required*. My suggestion is to start with a simple model, and utilize default values where applicable. Later, we will discuss some specific hyperparameters and their use cases.

Instance Types

It is suggested that you use instances that are available in the free tier of usage: 'ml.c4.xlarge' for training and 'ml.t2.medium' for deployment.

Here is what the exercise code looks like in the main notebook:

```
# import LinearLearner
from sagemaker import LinearLearner

# instantiate LinearLearner
```

Try to complete this code on your own, and I'll go over one possible solution, next!

Sol:

https://www.youtube.com/watch?time_continue=2&v=WaqDbA_5dNE&feature=emb_logo

Train your Estimator

After defining a model, you can format your training data and call `.fit()` to train the `LinearLearner`.
In the notebook, these exercises look as follows:

EXERCISE: Convert data into a RecordSet format

Prepare the data for a built-in model by converting the train features and labels into numpy array's of float values. Then you can use the `record_set` function to format the data as a RecordSet and prepare it for training!

```
# create RecordSet of training data
formatted_train_data = None
```

EXERCISE: Train the Estimator

After instantiating your estimator, train it with a call to .fit(), passing in the formatted training data.

```
%%time
# train the estimator on formatted training data
```

Complete this code, and you may check out a solution, next!

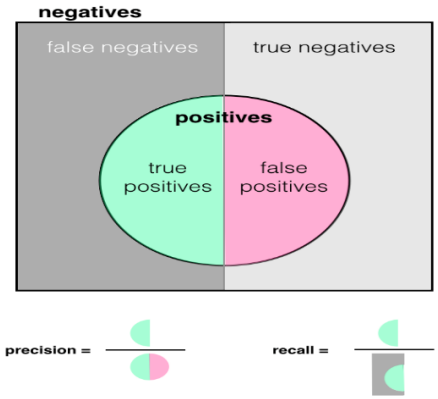
SOL: https://www.youtube.com/watch?time_continue=3&v=-whnaHFkPxU&feature=emb_logo

Precision & Recall

Precision and recall are just different metrics for measuring the "success" or performance of a trained model.

- **precision** is defined as the number of true positives (truly fraudulent transaction data, in this case) over *all* positives, and will be the higher when the amount of false positives is low.
- **recall** is defined as the number of true positives over true positives *plus* false negatives and will be higher when the number of false negatives is low.
  Both take into account true positives and will be higher for high, positive accuracy, too.

I find it helpful to look at the below image to wrap my head around these measurements:



In many cases, it may be worthwhile to optimize for a higher recall or precision, which gives you a more granular look at false positives and negatives.

Deploy an Endpoint and Evaluate Predictions

Finally, you are ready to deploy your trained `LinearLearner` and see how it performs according to various metrics. As you evaluate this model, I want you to think about:

- Which metrics *best* define success for this model?
- Is it important that we catch all cases of fraud?
- Is it important to prioritize a smooth user experience and never flag valid transactions?

The answers to these questions may vary based on use case!

In the main exercise notebook, you'll see the following instructions for deploying an endpoint and using it to make predictions:

EXERCISE: Deploy the trained model

Deploy your model to create a predictor. We'll use this to make predictions on our test data and evaluate the model.

```
%%time
# deploy and create a predictor
linear_predictor = None
```

Evaluating Your Model

Once your model is deployed, you can see how it performs when applied to the test data. Let's first test our model on just one test point, to see the resulting list.

```
# test one prediction
test_x_np = test_features.astype('float32')
```

```
result = linear_predictor.predict(test_x_np[0])

print(result)
```

You should proceed with investigating and evaluating the model test results. And next, I will discuss the results I got after deploying.

Shutting Down an Endpoint

As always, after deploying a model and making/saving predictions, you are free to delete your model endpoint and clean up that resource.

Next

SOL: https://www.youtube.com/watch?v=ZknaWInjSa4&feature=emb_logo

Model Improvements

https://www.youtube.com/watch?time_continue=5&v=JjZMuUnxKw4&feature=emb_logo

Improvement, Model Tuning

https://www.youtube.com/watch?time_continue=17&v=bb7zG0TdtRM&feature=emb_logo

Model Improvement: Accounting for Class Imbalance

We have a model that is *tuned* to get a higher recall, which aims to reduce the number of **false negatives**. Earlier, we discussed how class imbalance may actually bias our model towards predicting that all transactions are valid, resulting in higher **false negatives and true negatives**. It stands to reason that this model could be further improved if we account for this imbalance!

To account for class imbalance during training of a binary classifier, `LinearLearner` offers the hyperparameter, `positive_example_weight_mult`, which is the weight

assigned to positive (fraudulent data) examples when training a binary classifier. The weight of negative examples (valid data) is fixed at 1.
From the **hyperparameter documentation** on positive_example_weight_mult, it reads: "If you want the algorithm to choose a weight so that errors in classifying negative vs. positive examples have equal impact on training loss, specify `balanced`."
In the main exercise notebook, your exercises from defining to deploying an improved model looks as follows:

EXERCISE: Create a LinearLearner with a `positive_example_weight_mult` parameter

In addition to tuning a model for higher recall, you should add a parameter that helps account for class imbalance.

```
# instantiate a LinearLearner

# include params for tuning for higher recall
# *and* account for class imbalance in training data
linear_balanced = None
```

EXERCISE: Train the balanced estimator

Fit the new, balanced estimator on the formatted training data.

```
%%time
# train the estimator on formatted training data
```

EXERCISE: Deploy and evaluate the balanced estimator

Deploy the balanced predictor and evaluate it. Do the results match with your expectations?

```
%%time
# deploy and create a predictor
balanced_predictor = None
```

An important question here, when evaluating your model, is: **Do the results match with your expectations?** Much like in a scientific experiment it is good practice to start with a hypothesis that drives your idea for improving a model; if the trained model reacts in a different way than you expect (i.e. the model metrics are worse), it is worth revisiting your assumptions and approach.
Try to complete all these tasks, and if you get stuck, you can reference the solution video, next!

Shutting Down the Endpoint

Remember to delete your deployed, model endpoint after you finish with evaluation.

Sol: https://www.youtube.com/watch?time_continue=2&v=ncoPZdiVLJg&feature=emb_logo

Exercise: Define a Model w/ Specifications

Model Design

Now that you've seen how to tune and balance a `LinearLearner`, it is your turn to put together all that you've learned and build a new model, based on a real, business problem. This exercise is meant to be more open-ended, so that you get practice with the steps involved in designing a model and deploying it. In this exercise you'll:

- Create a LinearLearner model, according to specifications
- Train and deploy the model

- Evaluate the results
- Delete the endpoint (after evaluation)

Here is what you'll see in the main exercise notebook:

EXERCISE: Train and deploy a LinearLearner with appropriate hyperparameters, according to the given scenario

Scenario:

A bank has asked you to build a model that optimizes for a good user experience; users should only ever have up to about 15% of their valid transactions flagged as fraudulent. This requires that you make a design decision: Given the above scenario, **what metric (and value)** should you aim for during training?

You may assume that performance on a training set will be within about 5-10% of the performance on a test set. For example, if you get 80% on a training set, you can assume that you'll get between about 70-90% accuracy on a test set.

Your final model should account for class imbalance and be appropriately tuned.

```
%%time
# instantiate and train a LinearLearner

# include params for tuning for higher precision
# *and* account for class imbalance in training data

%%time
# deploy and evaluate a predictor

## IMPORTANT
# delete the predictor endpoint after evaluation
```

In this case, I will not be walking through a detailed solution (and there are multiple ways to approach this task and come up with a solution), but you can see one example solution in the solution notebook and on the next page.

Final Cleanup!

After completing these tasks, double check that you have deleted **all** your endpoints, and associated files. I'd also suggest manually deleting your S3 bucket, models, and endpoint configurations directly from your AWS console. You can find thorough cleanup instructions, **in the documentation**.

One Possible Solution

To optimize for *few* false positives (misclassified, valid transactions), I defined a model that accounts for class imbalance and optimizes for a **high precision**.
Let's review the scenario:

A bank has asked you to build a model that optimizes for a good user experience; users should only ever have up to about 15% of their valid transactions flagged as fraudulent.
My thoughts: If we're allowed about 15/100 incorrectly classified valid transactions (false positives), then I can calculate an approximate value for the precision that I want as: 85/(85+15) = 85%. I'll aim for about 5% higher during training to ensure that I get closer to 80-85% precision on the test data.

```
%%time
# One possible solution
# instantiate and train a LinearLearner

# include params for tuning for higher precision
# *and* account for class imbalance in training data
linear_precision = LinearLearner(role=role,
                                 train_instance_count=1,
                                 train_instance_type='ml.c4.xlarge',
                                 predictor_type='binary_classifier',
                                 output_path=output_path,
                                 sagemaker_session=sagemaker_session,
                                 epochs=15,
                                 binary_classifier_model_selection_criteria='recall_at
_target_precision',
                                 target_precision=0.9,
                                 positive_example_weight_mult='balanced')


# train the estimator on formatted training data
```

```
linear_precision.fit(formatted_train_data)
```

This model trains for a fixed precision of 90%, and, under that constraint, tries to get as high a recall value as possible. After training, I deployed the model to create a predictor:

```
%%time
# deploy and evaluate a predictor
precision_predictor = linear_precision.deploy(initial_instance_count=1, instance_type
='ml.t2.medium')
```

INFO:sagemaker:Creating model with name: linear-learner-2019-03-11-04-07-10-993
INFO:sagemaker:Creating endpoint with name linear-learner-2019-03-11-03-36-56-524
Then evaluated the model by seeing how it performed on test data:

```
print('Metrics for tuned (precision), LinearLearner.\n')

# get metrics for balanced predictor
metrics = evaluate(precision_predictor,
                   test_features.astype('float32'),
                   test_labels,
                   verbose=True)
```

These were the results I got:

Metrics for tuned (precision), LinearLearner.

```
prediction (col)    0.0  1.0
actual (row)
0.0               85276   26
1.0                  31  110

Recall:     0.780
Precision:  0.809
Accuracy:   0.999
```

As you can see, we still misclassified 26 of the valid results and so I may have to go back and up my aimed-for precision; the recall and accuracy are not too bad, considering the precision tradeoff.

Finally, I made sure to **delete the endpoint** after I was doe with evaluation.

```
## IMPORTANT
# delete the predictor endpoint
delete_endpoint(precision_predictor)
```

Deleted linear-learner-2019-03-11-03-36-56-524

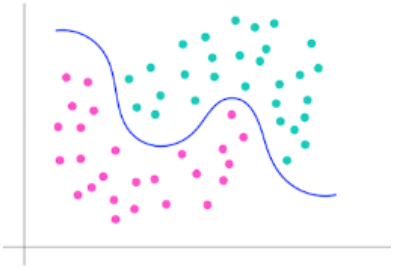
SUMMARY: https://www.youtube.com/watch?v=VsjDz3agnhQ&feature=emb_logo

LESSON 4

Deploying Custom Models

Design and train a custom PyTorch classifier by writing a training script. This is an especially useful skill for tasks that cannot be easily solved by built-in algorithms.

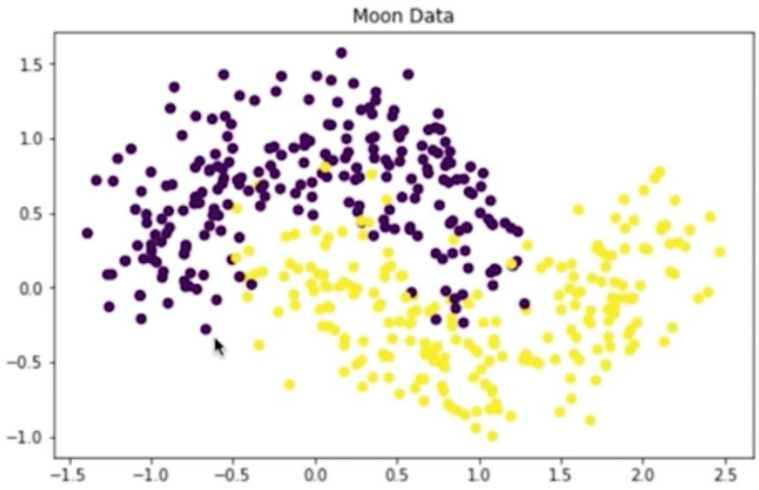
VIEW LESSON →



Lesson 4: Deploying Custom Models

Notebook: Custom Models & Moon Data, Exercise

Next, you'll approach the task of building and training a custom PyTorch classifier to classify data! Specifically, you'll be tasked with classifying "moon data," which is 2-dimensional data whose classes are distributed to look a bit like moons in 2D space.



Slightly noisy, purple (top) and yellow (bottom) "moon" data.

Building and training a custom model is presented as an alternative to something like a LinearLearner, which is great in many cases but may fail for data that is not easily separable. As you follow along with this lesson, you should work in the referenced SageMaker notebooks. We will present a solution to you, but please try to work on a solution of your own, when prompted. Much of the value in this experience will come from experimenting with the code, **in your own way**.

To open this notebook:

- Navigate to your SageMaker notebook instance, in the **SageMaker console**, which has been linked to the main **Github exercise repository**
- Activate the notebook instance (if it is in a "Stopped" state), and open it via Jupyter
- Click on the exercise notebook in the `Moon_Data` directory.

  You may also directly view the exercise and solution notebooks via the repository at the following links:

- **Exercise notebook**
- **Solution notebook**

  In this particular case, you will also find a directory `source` and `source_solution` for further reference.

Machine Learning Engineer Nanodegree Material

The solutions are meant to be consulted if you are stuck or want to check your work.

Notebook Outline

We'll go over the following steps to complete the notebook.

- Upload data to S3
- Define a PyTorch neural network for binary classification
- Write a custom **training script**
- Train and evaluate the custom model

Later: Delete Resources

At the end of this exercise, and intermittently, you will be reminded to delete your endpoints and resources so that you do not incur any extra processing or storage fees!

Moon Data & Custom Models

https://www.youtube.com/watch?v=vb5ojq8Jw7k&feature=emb_logo

Upload Data to S3

https://www.youtube.com/watch?time_continue=17&v=Mz08Bac6h2Y&feature=emb_logo

Exercise: Custom PyTorch Classifier

SOL: https://www.youtube.com/watch?time_continue=1&v=FINTJpz1Yx0&feature=emb_logo

Solution: Complete Training Script

https://www.youtube.com/watch?time_continue=5&v=xmrB3sqbeTU&feature=emb_logo

Defining a Custom Model

To define a custom model, you need to have the **model** itself and the following two scripts:

- A **training** script that defines how the model will accept input data, and train. This script should also save the trained model parameters.
- A **predict** script that defines how a trained model produces an output and in what format.

PyTorch

In PyTorch, you have the option of defining a neural network of your own design. These models do not come with any built-in predict scripts and so you have to write one yourself.

SKLearn

The `scikit-learn` library, on the other hand, has many pre-defined models that come with train and predict functions attached!
You can define custom SKLearn models in a very similar way that you do PyTorch models only you typically **only** have to define the training script. You can use the default predict function.

PyTorch Estimator

https://www.youtube.com/watch?time_continue=37&v=pJOkQfMtxpc&feature=emb_logo

Create and Deploy a Trained Model

Before you can deploy a custom PyTorch model, you have to take one more step: creating a `PyTorchModel`. In earlier exercises, you could see that a call to `.deploy()` created **both** a *model* and an *endpoint*, but for PyTorch models, these steps have to be separate. PyTorch models do not automatically come with `.predict()` functions attached (as many Amazon and Scikit-learn models do, for example) and you may have noticed that you've been given a `predict.py` file in the `source` directory. This file is responsible for loading a trained model and applying it to passed in, numpy data.

Now, when we created a PyTorch **estimator**, we specified where the training script, `train.py` was located. And we'll have to do something very similar here, but for a PyTorch **model** and the `predict.py` file.

EXERCISE: Instantiate a PyTorchModel

You can create a `PyTorchModel` from your trained, estimator attributes. This model is responsible for knowing how to execute a specific `predict.py` script. And this model is what you'll deploy to create an endpoint.

Model Parameters

To instantiate a `PyTorchModel`, (**documentation, here**) you pass in the same arguments as your PyTorch estimator, with a few additions/modifications:

- **model_data**: The trained model.tar.gz file created by your estimator, which can be accessed as estimator.model_data.
- **entry_point**: This time, this is the path to the Python script SageMaker runs for prediction rather than training, predict.py.

In the exercise notebook, you've been given the following code to fill in:

```
%%time
# importing PyTorchModel
from sagemaker.pytorch import PyTorchModel

# Create a model from the trained estimator data
# And point to the prediction script
model = None
```

EXERCISE: Deploy the trained model

Deploy your model to create a predictor. We'll use this to make predictions on our test data and evaluate the model.

```
%%time
```

```
# deploy and create a predictor
predictor = None
```

Evaluate your Model

After deploying your model, you have been given some code to evaluate its performance according to a variety of metrics!

Remember to **delete your predictor endpoint** *after* you've finished evaluating the model.

Try to complete these steps on your own, and I'll go over one solution, next!

SOL:

https://www.youtube.com/watch?time_continue=1&v=qZTyQqo9FWM&feature=emb_logo

Clean up Resources

It is good practice to always clean up and delete any resources that you are no longer using. That is, after you complete an exercise, and you are done with predictions and data analysis, you should get rid of any:

- Data source in S3 that you are no longer using
- Endpoint configuration files that you no longer need
- Endpoints that you will no longer use
- CloudWatch logs that are no longer useful

Deleting Endpoints

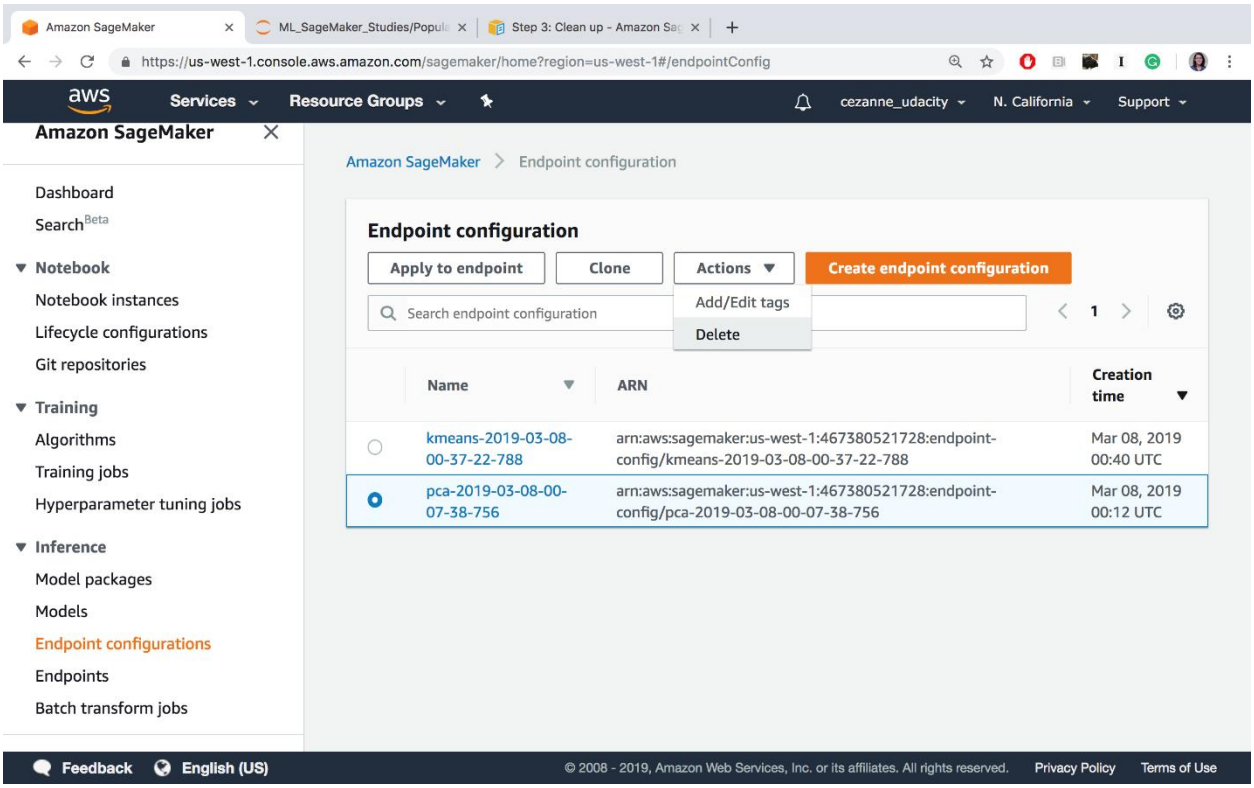
In the notebook, we have usually included code to delete your endpoints after creating some predictions, for example:

```
# delete predictor endpoint
session.delete_endpoint(predictor.endpoint)
```

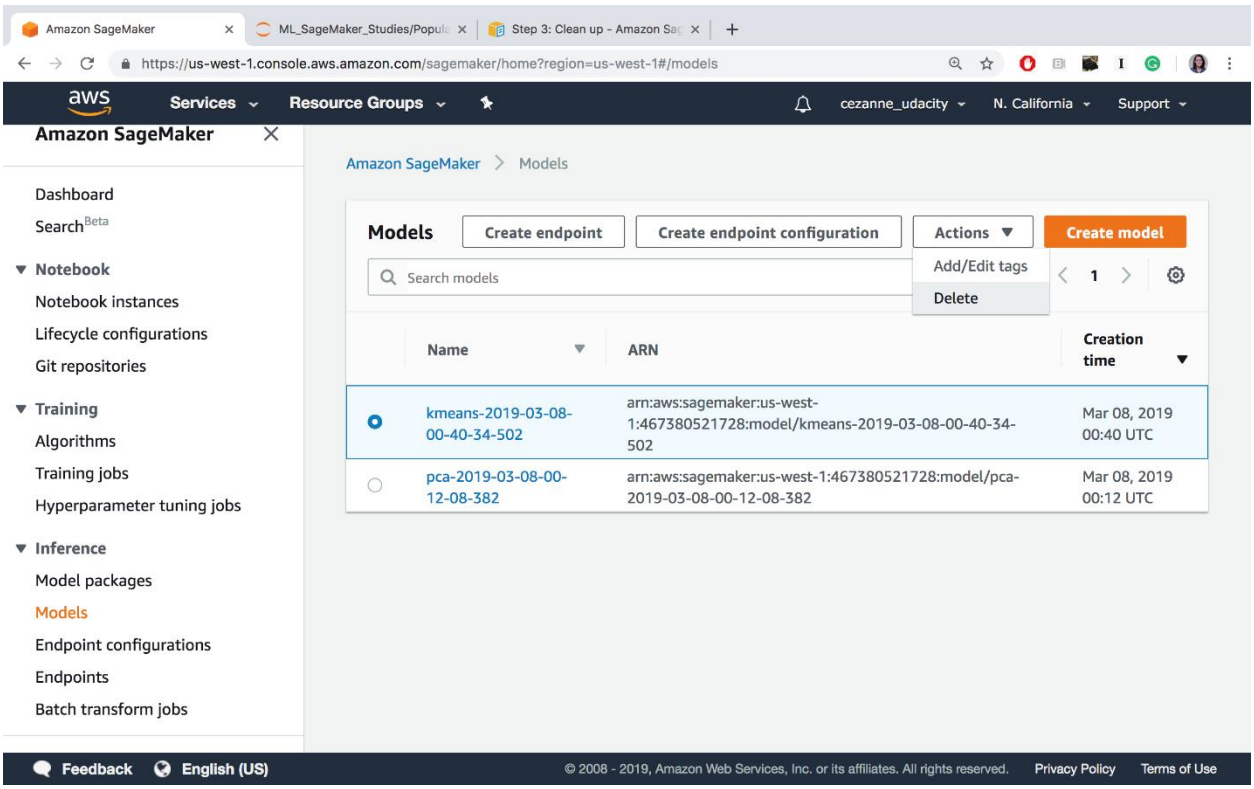
Thorough Clean up

You can find a link for instructions on cleaning up all your resources, **in this documentation** and I will go over some of these details, next.

- Open the Amazon SageMaker console at **https://console.aws.amazon.com/sagemaker/** and delete the following resources:
- The endpoint configuration.
- The model.

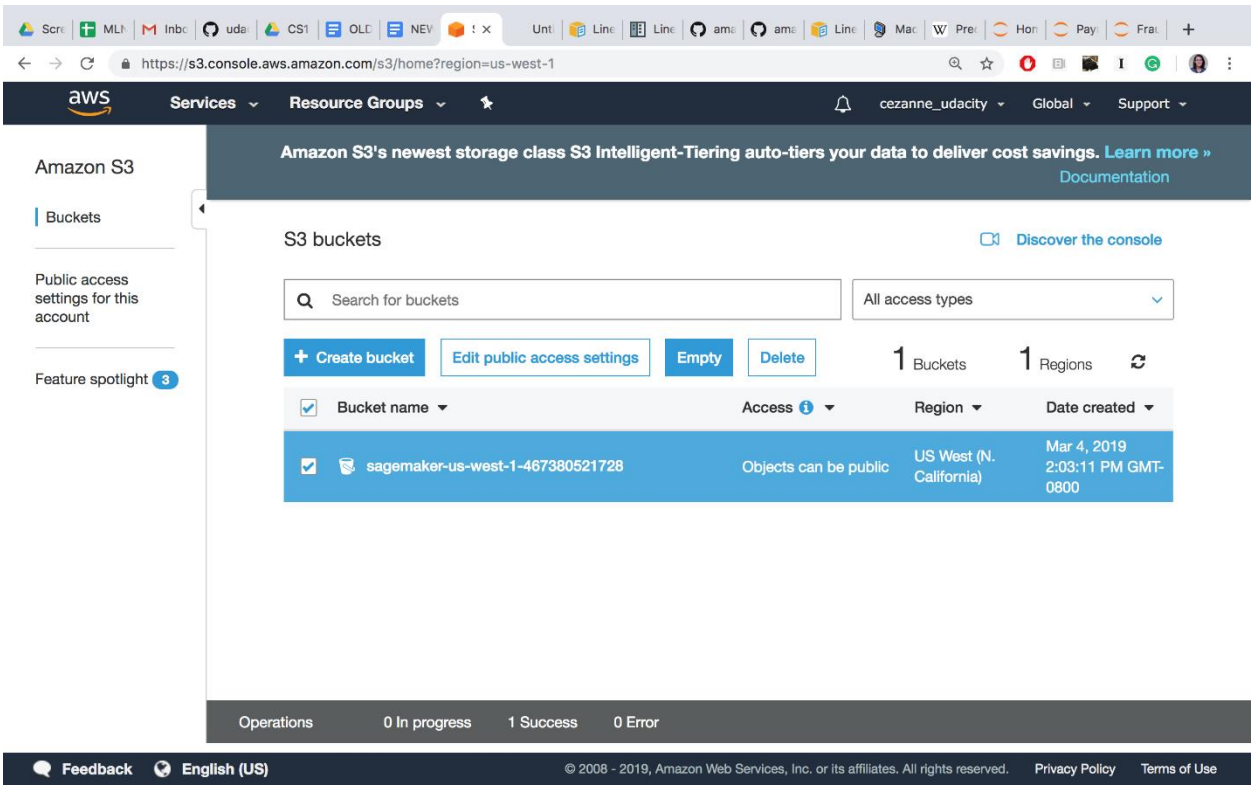


Delete endpoint config files.



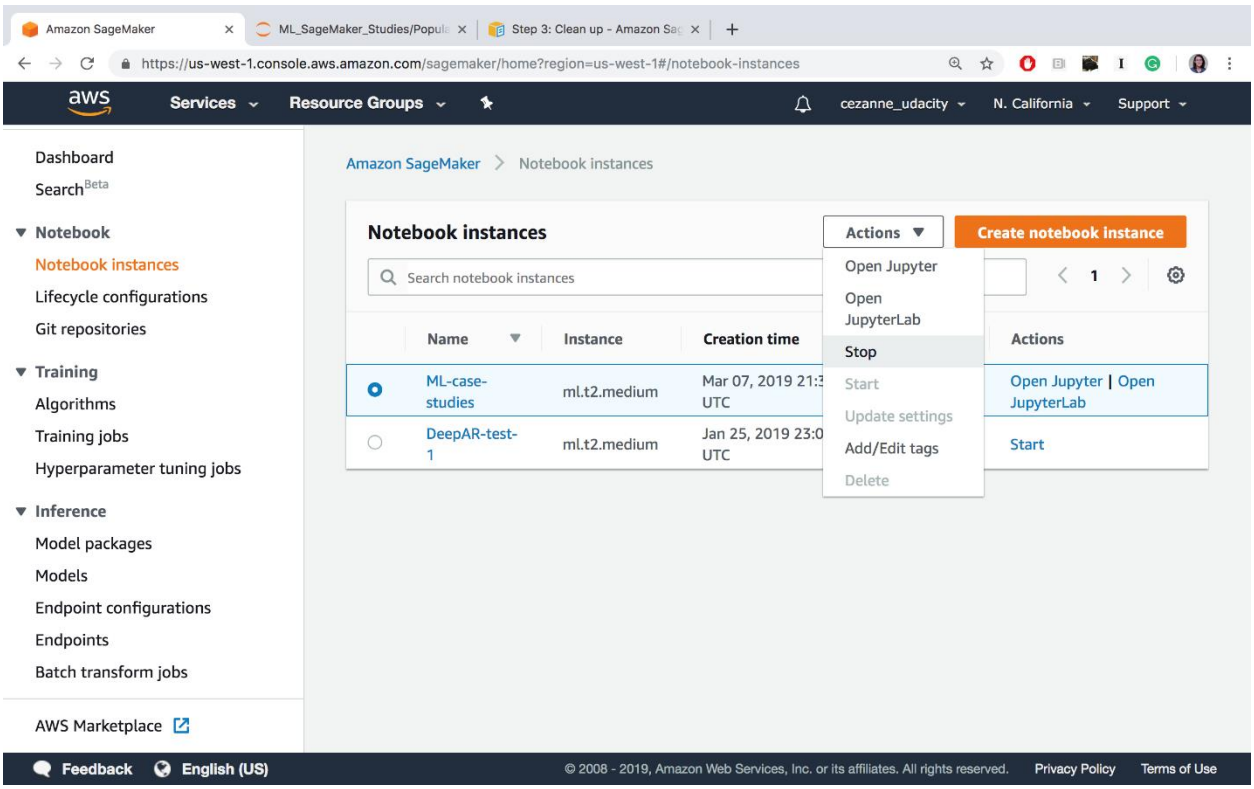
Deleting models

- Open the Amazon S3 console at **https://console.aws.amazon.com/s3/** and delete or empty the bucket that you created for storing model artifacts and the training dataset.



Delete or empty your S3 bucket (empty is recommended until the end of the course, when you should delete this bucket entirely)

- Open the Amazon CloudWatch console at **https://console.aws.amazon.com/cloudwatch/** and delete all of the log groups that have names starting with /aws/sagemaker/.
  At the end of this course, you may also choose to delete the entire notebook instance and IAM Role, but you may keep these as is, for now. In between lessons, if you are taking a break, you may want to **Stop** your notebook and pause it from continuously running.



Stopping the ML-case-studies notebook

Cleaning up resources at the end of an exercise or lesson is a great practice to get into!

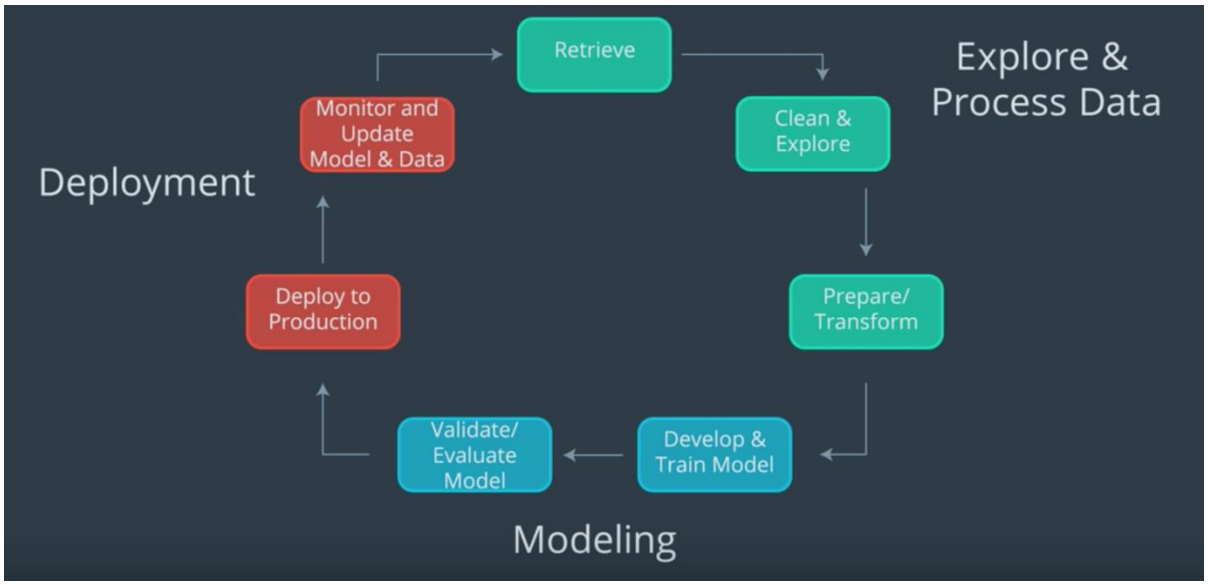
IMPORTANT

To avoid incurring additional charges, it is suggested that you *DELETE* any unused notebooks and data resources on S3 and CloudWatch.

Next

Summary of Skills

You've really learned a lot at this point in the course! You should be familiar with each part of the machine learning workflow from data loading and processing to model training and deployment.



Machine learning workflow.

By studying specific case studies, you should have a better idea of what kinds of machine learning models may be applied to different scenarios.

- Essentially, you want to choose an **unsupervised** or **supervised** model based on the given data
- Often you'll want to process and transform your given data to extract the most relevant and comparable **features** for a given task
- Then you'll want to refine your choice of model and tune it based on constraints given by a business problem or the data itself
- You also looked at different ways to visualize what your trained models have learned and different ways to measure success through a variety of **model evaluation metrics**

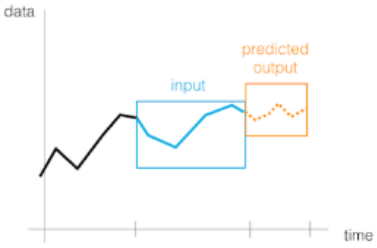
In this last example, you saw how to define and train a custom model of your own design, by specifying the model architecture and a **training script**. Now, you are ready to move on to the final project!

LESSON 5

Time-Series Forecasting

Learn how to format time series data into context (input) and prediction (output) data, and train the built-in algorithm, DeepAR; this uses an RNN to find recurring patterns in time series data.

VIEW LESSON →



Lesson 5: Time-Series Forecasting

https://www.youtube.com/watch?v=U8k2Fl2zgJ8&feature=emb_logo

Time Series Notebook

https://www.youtube.com/watch?time_continue=2&v=OZJu6or8Fl0&feature=emb_logo

Notebook: Time-Series Forecasting, Exercise

Next, you'll approach the task of time-series forecasting. You'll be taking a look at household energy consumption data, originally taken from **Kaggle**. As you follow along with this lesson, you should work in the referenced SageMaker notebooks. We will present a solution to you, but please try to work on a solution of your own, when prompted. Much of the value in this experience will come from experimenting with the code, **in your own way**.

To open this notebook:

- Navigate to your SageMaker notebook instance, in the **SageMaker console**, which has been linked to the main **Github exercise repository**
- Activate the notebook instance (if it is in a "Stopped" state), and open it via Jupyter
- Click on the exercise notebook in the `Time_Series_Forecasting` directory.

  You may also directly view the exercise and solution notebooks via the repository at the following links:

- **Exercise notebook**
- **Solution notebook**

  The solution notebook is meant to be consulted if you are stuck or want to check your work.

Notebook Outline

We'll go over the following steps to complete the notebook.

- Load in and explore household energy consumption data
- Clean the data and transform it to prepare for training a model
- Format the data into JSON Lines
- Train a DeepAR model on defined context and prediction data points
- Evaluate the model by comparing known and predicted consumption values

Later: Delete Resources

At the end of this exercise, and intermittently, you will be reminded to delete your endpoints and resources so that you do not incur any extra processing or storage fees!

Processing Energy Data

https://www.youtube.com/watch?time_continue=224&v=zxnoYK4sYgk&feature=emb_logo

Splitting in Time

We'll evaluate our model on a test set of data. For machine learning tasks like classification, we typically create train/test data by randomly splitting examples into different sets. For forecasting it's important to do this train/test split in **time** rather than a random split of all data points.

Training Time Series

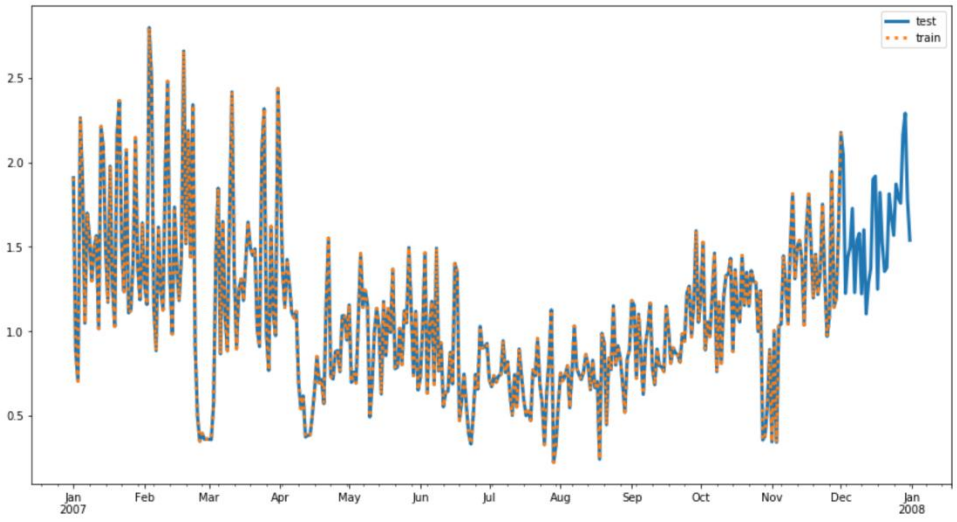
In general, we can create training data by taking each of our complete time series and leaving off the last `prediction_length` data points to create corresponding, training time series.
In code this looks like this:

```
def create_training_series(complete_time_series, prediction_length):
    '''Given a complete list of time series data, create training time series.
       :param complete_time_series: A list of all complete time series.
       :param prediction_length: The number of points we want to predict.
       :return: A list of training time series.
       '''
    # get training series
    time_series_training = []

    for ts in complete_time_series:
        # truncate trailing `prediction_length` pts
        time_series_training.append(ts[:-prediction_length])

    return time_series_training
```

DeepAR will train on the provided data looking at different intervals that are `context_length` number of points as input and the next `prediction_length` number of points as output. It selects the context from the given, truncated training data, which is why it is important to leave off the last `prediction_length` points.

Training and Test Series

We can visualize what these series look like, by plotting the train/test series on the same axis. We should see that the test series contains *all* of our data in a year, and a training series contains all but the last `prediction_length` points. Below are train/test series for 2007.



Test series and train series (truncated, in orange).

Solution: Formatting JSON Lines & DeepAR Estimator

https://www.youtube.com/watch?time_continue=5&v=1Wx-LK9TVWY&feature=emb_logo

Instantiating a DeepAR Estimator

Some estimators have specific, SageMaker constructors, but not all. Instead, you can create a base Estimator and pass in the specific **image** (or container) that holds a specific model. The container for the DeepAR model can be gotten as follows:

```
from sagemaker.amazon.amazon_estimator import get_image_uri

image_name = get_image_uri(boto3.Session().region_name, # get the region
                           'forecasting-deepar') # specify image
```

Now that you have the correct image, you can instantiate an Estimator. You're given the following, in-notebook exercise.

EXERCISE: Instantiate an Estimator

A generic Estimator will be defined by the usual constructor arguments and an `image_name`.
You can take a look at the **estimator source code** to view specifics.
If you complete this task, you can move on to setting DeepAR's model and training hyperparameters and call `.fit()` to start a training job! You're encouraged to keep going, deploying and evaluating the model on your owm; you are welcome to consult the solution videos to see if your answer matches mine.

Solution: Complete Estimator & Hyperparameters

https://www.youtube.com/watch?time_continue=10&v=ah7muNBc3dI&feature=emb_logo

Making Predictions

https://www.youtube.com/watch?time_continue=13&v=BKOYIfgjsq8&feature=emb_logo

Predicting the Future

Recall that we did not give our model any data about 2010, but let's see if it can predict the energy consumption given **no target**, only a known start date!

EXERCISE: Format a request for a "future" prediction

Your task is to create a formatted input to send to the deployed predictor passing in my usual parameters for "configuration". The "instances" will, in this case, just be one instance, defined by the following:

- **start**: The start time will be time stamp that you specify. To predict the first 30 days of 2010, start on Jan. 1st, '2010-01-01'.
- **target**: The target will be an empty list because this year has no, complete associated time series; we specifically withheld that information from our model, for testing purposes. For example:

```
{"start": start_time, "target": []} # empty target
```

You'll see the following code to complete in the main exercise notebook. Complete the `instances` and see if you can generate some future predictions. **Also, remember to delete your model endpoint when you are done making predictions and evaluating your model.**

```
# Starting my prediction at the beginning of 2010
start_date = '2010-01-01'
timestamp = '00:00:00'

# formatting start_date
start_time = start_date +' '+ timestamp

# formatting request_data
```

```
## TODO: fill in instances information
request_data = {"instances": [{"start": None, "target": None}],
                "configuration": {"num_samples": 50,
                                  "output_types": ["quantiles"],
                                  "quantiles": ['0.1', '0.5', '0.9']}
                }

json_input = json.dumps(request_data).encode('utf-8')

print('Requesting prediction for '+start_time)
```

Solution: Predicting Future Data

https://www.youtube.com/watch?time_continue=17&v=HT5xKDOgHYw&feature=emb_logo

Project Review

Deploying a Sentiment Analysis Model

Meets Specifications

Hey! You've done a fantastic job completing the Deploying a Sentiment Analysis Model project! 😀 💯
It is evident that you've given enough time & respect to the project which is wonderful. ✨
Awesome job! 🎉
Keep up the amazing work & All the very best for the future!! 🍻

Files Submitted

✓	**The submission includes all required files, including notebook, python scripts, and html files.**
	Well done submitting all required files. 🎉

Preparing and Processing Data

✓	**Answer describes what the pre-processing method does to a review.**
	You described clearly what the pre-processing method does. Good job!! ✨ 😄
✓	**Answer describes how the processing methods are applied to the training and test data sets and what, if any, issues there may be.**
	Good job on providing a clear answer! 🎉
✓	**Notebook displays the five most frequently appearing words.**
	You did an amazing job in displaying the five most frequently appearing words! Nice! 🎉 💯
✓	**The `build_dict` method is implemented and constructs a valid word dictionary.**
	Yay! You found an optimal way to create a sorted list from a dictionary. Great!! 🍻 💯

Machine Learning Engineer Nanodegree Material

Build and Train a PyTorch Model

✓	**The train method is implemented and can be used to train the PyTorch model.**
	There's a lot of flexibility here in choosing to implement the training method. You performed a good job on it as well! 🎁 ✨
✓	**The RNN is trained using SageMaker's supported PyTorch functionality.**
	Your effort in correctly implementing the train method shows! Nice! 💯

Deploy a Model for Testing

✓	**The trained PyTorch model is successfully deployed.**
	Perfect!! Your trained PyTorch model is successfully deployed!! 🎉 😄

Use the Model for Testing

✓	**Answer describes the differences between the RNN model and the XGBoost model and how they perform on the IMDB data.**
	Good! Your answer is clear & to the point! 💯
✓	**The test review has been processed correctly and stored in the `test_data` variable.**
	Great!! Your prediction function works really well. 🎉
✓	**The `predict_fn()` method in `serve/predict.py` has been implemented.**
	Good job on implementing the method to process the test review correctly. 😄 💯

Deploying a Web App

✓	**The model is deployed and the Lambda / API Gateway integration is complete so that the web app works (make sure to include your modified `index.html`).**
	Amazing! Your index.html file has been edited correctly for it to work!! 🎉

Deploying a Web App

✓	**The model is deployed and the Lambda / API Gateway integration is complete so that the web app works (make sure to include your modified `index.html`).**
	Amazing! Your index.html file has been edited correctly for it to work!! 🎉
✓	**Answer gives a sample review and the resulting predicted sentiment.**
	Good job! Your review looks reasonable! 🎉 😄

Capstone Project

Machine Learning Engineer Nanodegree Material

Meets Specifications

Dear Udacian,
This is a very interesting problem and honestly, it was not an easy task to do. But you have handled the work with utmost care and wit. I love your approach to solve the problem. Overall an amazing report and work. I wish you good luck on your future path. Happy learning and stay safe. Take care.

Hyperparameter and Feature Engineering will improve as you work more on machine learning projects. To improve this skill, select a problem from Kaggle and think about how you want to solve it. Then explore different kernels from Kaggle and see how people are dealing with it. Try to google similar problems and see how others are doing it.
You need to improve your deep learning skills too. So I suggest you go through these free courses:

Prerequisite:

https://github.com/fastai/numerical-linear-algebra/blob/master/README.md (Linear Algebra)
https://www.edx.org/course/introduction-probability-science-mitx-6-041x-2 (Probability)

Newcomer:
1.http://course18.fast.ai/ml
2.http://course18.fast.ai/index.html

Deep learning courses:

http://onlinehub.stanford.edu/cs230 (Deep Learning)
http://onlinehub.stanford.edu/cs224 (NLP)
http://cs231n.stanford.edu/ (Computer Vision)
https://www.youtube.com/playlist?list=PLqYmG7hTraZDNJre23vqCGIVpfZ_K2RZs (RL 1)
http://web.stanford.edu/class/cs234/schedule.html (RL 2) (edited)

Definition

✓	**Student provides a high-level overview of the project in layman's terms. Background information such as the problem domain, the project origin, and related data sets or input data is given.**
	Very good overview of the project is presented along with some nice discussion. This is a very interesting project.

✓	**The problem which needs to be solved is clearly defined. A strategy for solving the problem, including discussion of the expected solution, has been made.**
	The problem is clearly defined and discussed. Solving strategy is well elaborated.

✓	**Metrics used to measure the performance of a model or result are clearly defined. Metrics are justified based on the characteristics of the problem.**
	Good choice of the metrics here and necessary justification is made.

Analysis

✓ **If a dataset is present, features and calculated statistics relevant to the problem have been reported and discussed, along with a sampling of the data. In lieu of a dataset, a thorough description of the input space or input data has been made. Abnormalities or characteristics of the data or input that need to be addressed have been identified.**

Statistical analysis of the dataset is presented

✓ **A visualization has been provided that summarizes or extracts a relevant characteristic or feature about the dataset or input data with thorough discussion. Visual cues are clearly defined.**

✓ **Algorithms and techniques used in the project are thoroughly discussed and properly justified based on the characteristics of the problem.**

You have discussed the algorithms in detail. Good job here.

✓ **Student clearly defines a benchmark result or threshold for comparing performances of solutions obtained.**

Good choice here.

Methodology

✓ **All preprocessing steps have been clearly documented. Abnormalities or characteristics of the data or input that needed to be addressed have been corrected. If no data preprocessing is necessary, it has been clearly justified.**

Stepwise discussion is made.

✓ **The process for which metrics, algorithms, and techniques were implemented with the given datasets or input data has been thoroughly documented. Complications that occurred during the coding process are discussed.**

Nice to see some nice details here.

✓ **The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary.**

Results

✓ **The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem.**

Results

✓ **The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem.**

Necessary comparison is made.

✓ **The final model's qualities—such as parameters—are evaluated in detail. Some type of analysis is used to validate the robustness of the model's solution.**

Very good analysis is done to prove the robustness of the model.
I love your improvements note.

DOWNLOAD PROJECT

https://confirm.udacity.com/2R76NJ67