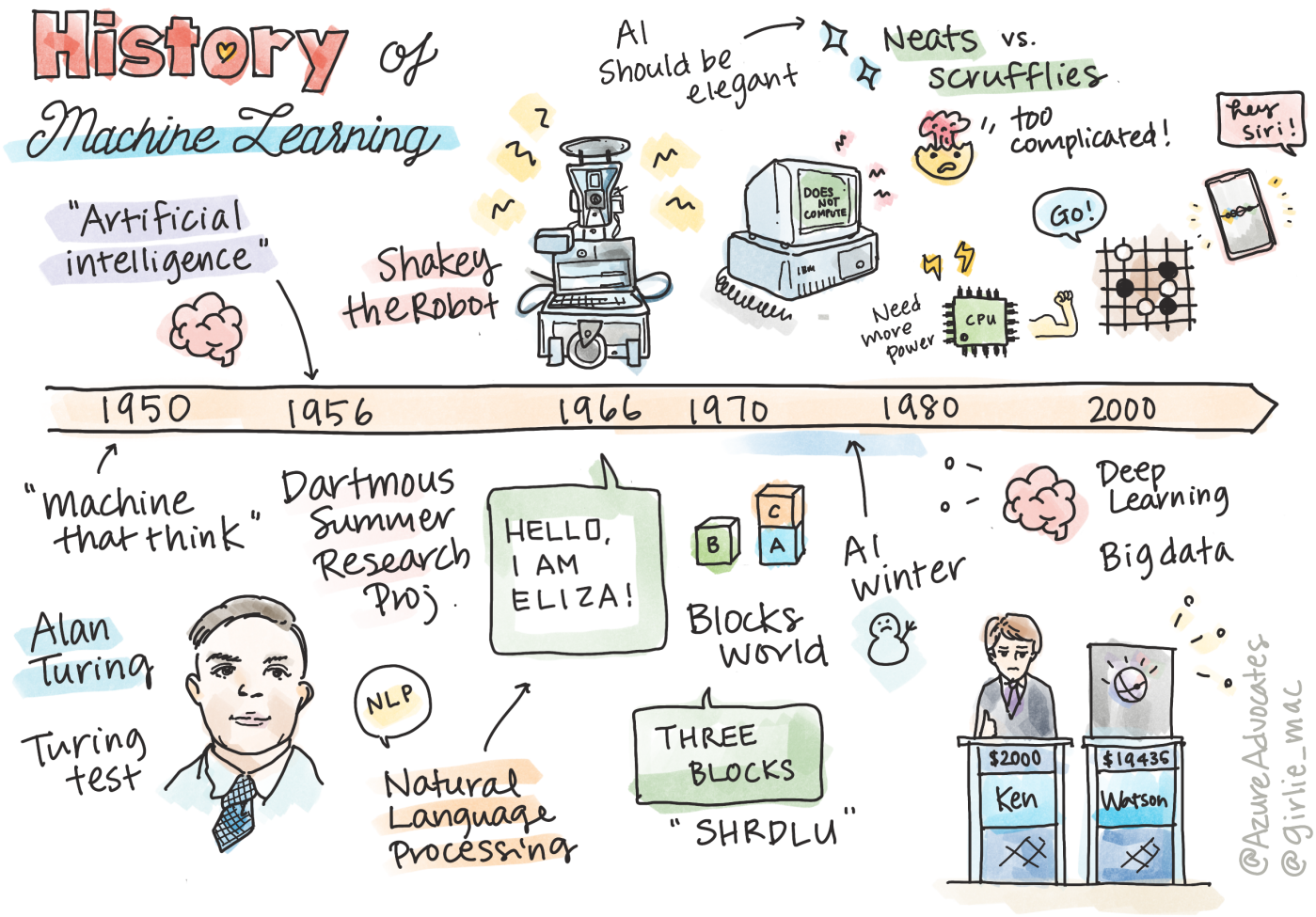
History of machine learning



> Sketchnote by Tomomi Imura

Pre-lecture quiz

In this lesson, we will walk through the major milestones in the history of machine learning and artificial intelligence.

The history of artificial intelligence, AI, as a field is intertwined with the history of machine learning, as the algorithms and computational advances that underpin ML fed into the development of AI. It is useful to remember that, while these fields as distinct areas of inquiry began to crystallize in the 1950s, important algorithmical, statistical, mathematical, computational and technical discoveries predated and overlapped this era. In fact, people have been thinking about these questions for

hundreds of years: this article discusses the historical intellectual underpinnings of the idea of a 'thinking machine.'

Notable discoveries

- 1763, 1812 Bayes Theorem and its predecessors. This theorem and its applications underlie inference, describing the probability of an event occurring based on prior knowledge.
- 1805 Least Square Theory by French mathematician Adrien-Marie Legendre. This theory, which you will learn about in our Regression unit, helps in data fitting.
- 1913 Markov Chains named after Russian mathematician Andrey Markov is used to describe a sequence of possible events based on a previous state.
- 1957 Perceptron is a type of linear classifier invented by American psychologist Frank Rosenblatt that underlies advances in deep learning.
- 1967 Nearest Neighbor is an algorithm originally designed to map routes. In an ML context it is used to detect patterns.
- 1970 Backpropagation is used to train feedforward neural networks.
- 1982 Recurrent Neural Networks are artificial neural networks derived from feedforward neural networks that create temporal graphs.

✅ Do a little research. What other dates stand out as pivotal in the history of ML and AI?

1950: Machines that think

Alan Turing, a truly remarkable person who was voted by the public in 2019 as the greatest scientist of the 20th century, is credited as helping to lay the foundation for the concept of a 'machine that can think.' He grappled with naysayers and his own need for empirical evidence of this concept in part by creating the Turing Test, which you will explore in our NLP lessons.

1956: Dartmouth Summer Research Project

"The Dartmouth Summer Research Project on artificial intelligence was a seminal event for artificial intelligence as a field," and it was here that the term 'artificial intelligence' was coined (source).

> Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.

The lead researcher, mathematics professor John McCarthy, hoped "to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." The participants included another luminary in the field, Marvin Minsky.

The workshop is credited with having initiated and encouraged several discussions including "the rise of symbolic methods, systems focussed on limited domains (early expert systems), and deductive systems versus inductive systems." (source).

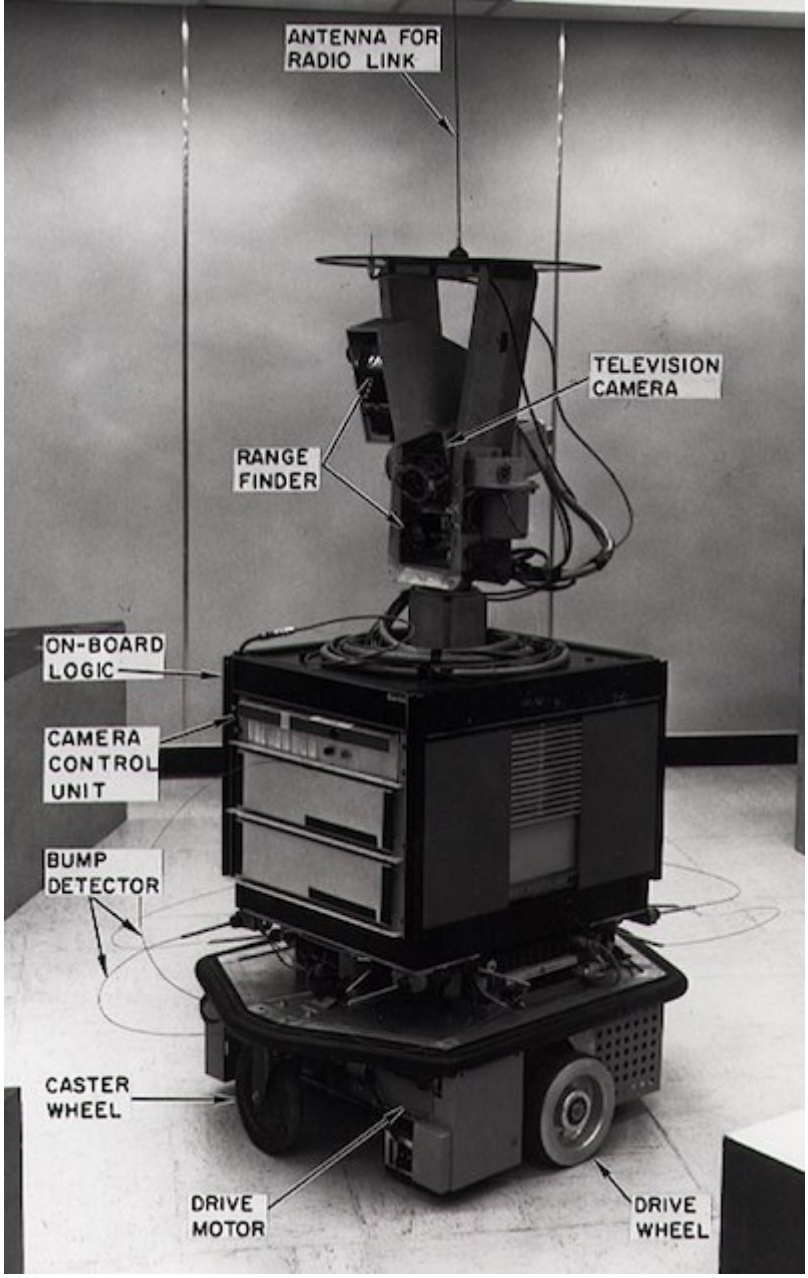
1956 - 1974: "The golden years"

From the 1950s through the mid '70s, optimism ran high in the hope that AI could solve many problems. In 1967, Marvin Minsky stated confidently that "Within a generation ... the problem of creating 'artificial intelligence' will substantially be solved." (Minsky, Marvin (1967), Computation: Finite and Infinite Machines, Englewood Cliffs, N.J.: Prentice-Hall)

natural language processing research flourished, search was refined and made more powerful, and the concept of 'micro-worlds' was created, where simple tasks were completed using plain language instructions.

Research was well funded by government agencies, advances were made in computation and algorithms, and prototypes of intelligent machines were built. Some of these machines include:

- Shakey the robot, who could maneuver and decide how to perform tasks 'intelligently'.



> Shakey in 1972

- Eliza, an early 'chatterbot', could converse with people and act as a primitive 'therapist'. You'll learn more about Eliza in the NLP lessons.

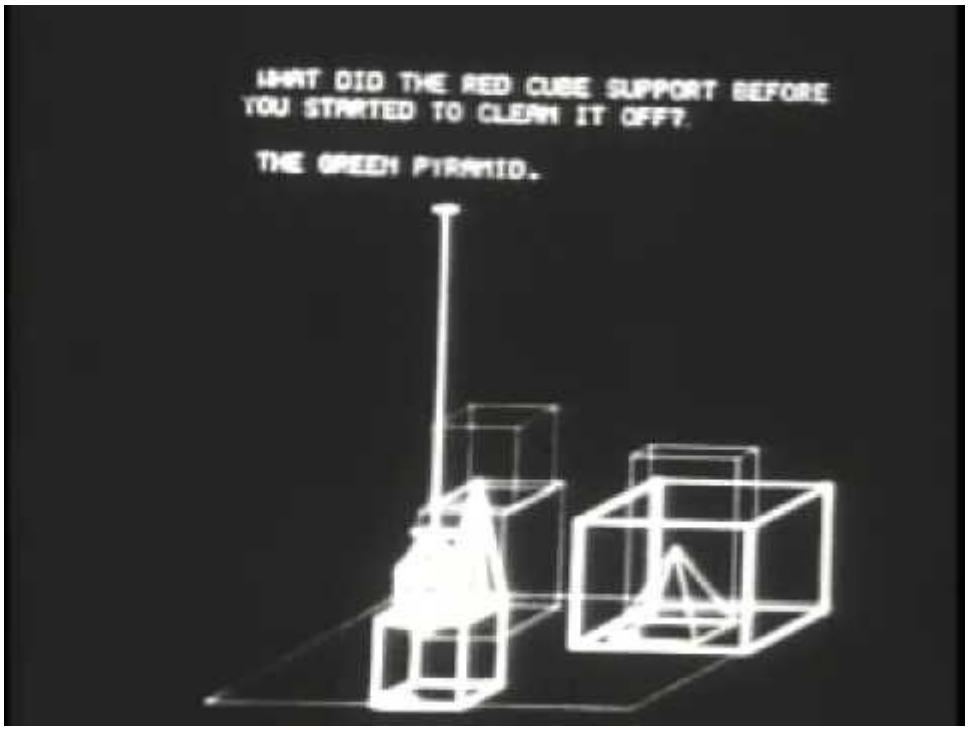
```
  Welcome to
                 EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                 EE      LL       II       ZZ   AA   AA
                 EEEEE   LL       II     ZZZ    AAAAAAA
                 EE      LL       II    ZZ      AA   AA
                 EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

> A version of Eliza, a chatbot

- "Blocks world" was an example of a micro-world where blocks could be stacked and sorted, and experiments in teaching machines to make decisions could be tested. Advances built with libraries such as SHRDLU helped propel language processing forward.



> 🎥 Click the image above for a video: Blocks world with SHRDLU

1974 - 1980: "AI Winter"

By the mid 1970s, it had become apparent that the complexity of making 'intelligent machines' had been understated and that its promise, given the available compute power, had been overblown. Funding dried up and confidence in the field slowed. Some issues that impacted confidence included:

- **Limitations**. Compute power was too limited.
- **Combinatorial explosion**. The amount of parameters needed to be trained grew exponentially as more was asked of computers, without a parallel evolution of compute power and capability.
- **Paucity of data**. There was a paucity of data that hindered the process of testing, developing, and refining algorithms.
- **Are we asking the right questions?**. The very questions that were being asked began to be questioned. Researchers began to field criticism about their approaches:
    - Turing tests came into question by means, among other ideas, of the 'chinese room theory' which posited that, "programming a digital computer may make it appear to understand language but could not produce real understanding." (source)
    - The ethics of introducing artificial intelligences such as the "therapist" ELIZA into society was challenged.

At the same time, various AI schools of thought began to form. A dichotomy was established between "scruffy" vs. "neat AI" practices. *Scruffy* labs tweaked programs for hours until they had the desired results. *Neat* labs "focused on logic and formal problem solving". ELIZA and SHRDLU were well-known *scruffy* systems. In the 1980s, as demand emerged to make ML systems reproducible, the *neat* approach gradually took the forefront as its results are more explainable.

1980s Expert systems

As the field grew, its benefit to business became clearer, and in the 1980s so did the proliferation of 'expert systems'. "Expert systems were among the first truly successful forms of artificial intelligence (AI) software." (source).

This type of system is actually *hybrid*, consisting partially of a rules engine defining business requirements, and an inference engine that leveraged the rules system to deduce new facts.

This era also saw increasing attention paid to neural networks.

1987 - 1993: AI 'Chill'

The proliferation of specialized expert systems hardware had the unfortunate effect of becoming too specialized. The rise of personal computers also competed with these large, specialized, centralized systems. The democratization of computing had begun, and it eventually paved the way for the modern explosion of big data.

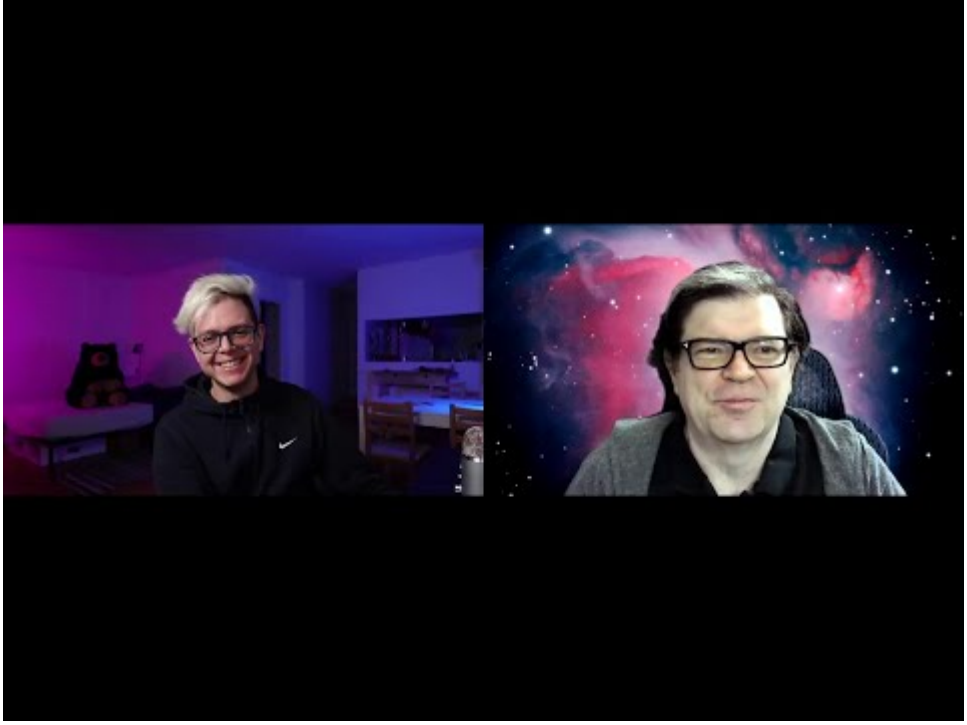
1993 - 2011

This epoch saw a new era for ML and AI to be able to solve some of the problems that had been caused earlier by the lack of data and compute power. The amount of data began to rapidly increase and become more widely available, for better and for worse, especially with the advent of the smartphone around 2007. Compute power expanded exponentially, and algorithms evolved alongside. The field began to gain maturity as the freewheeling days of the past began to crystallize into a true discipline.

Now

Today, machine learning and AI touch almost every part of our lives. This era calls for careful understanding of the risks and potentials effects of these algorithms on human lives. As Microsoft's Brad Smith has stated, "Information technology raises issues that go to the heart of fundamental human-rights protections like privacy and freedom of expression. These issues heighten responsibility for tech companies that create these products. In our view, they also call for thoughtful government regulation and for the development of norms around acceptable uses" (source).

It remains to be seen what the future holds, but it is important to understand these computer systems and the software and algorithms that they run. We hope that this curriculum will help you to gain a better understanding so that you can decide for yourself.



> 📽️ Click the image above for a video: Yann LeCun discusses the history of deep learning in this lecture

---

🚀Challenge

Dig into one of these historical moments and learn more about the people behind them. There are fascinating characters, and no scientific discovery was ever created in a cultural vacuum. What do you discover?

Post-lecture quiz

Review & Self Study

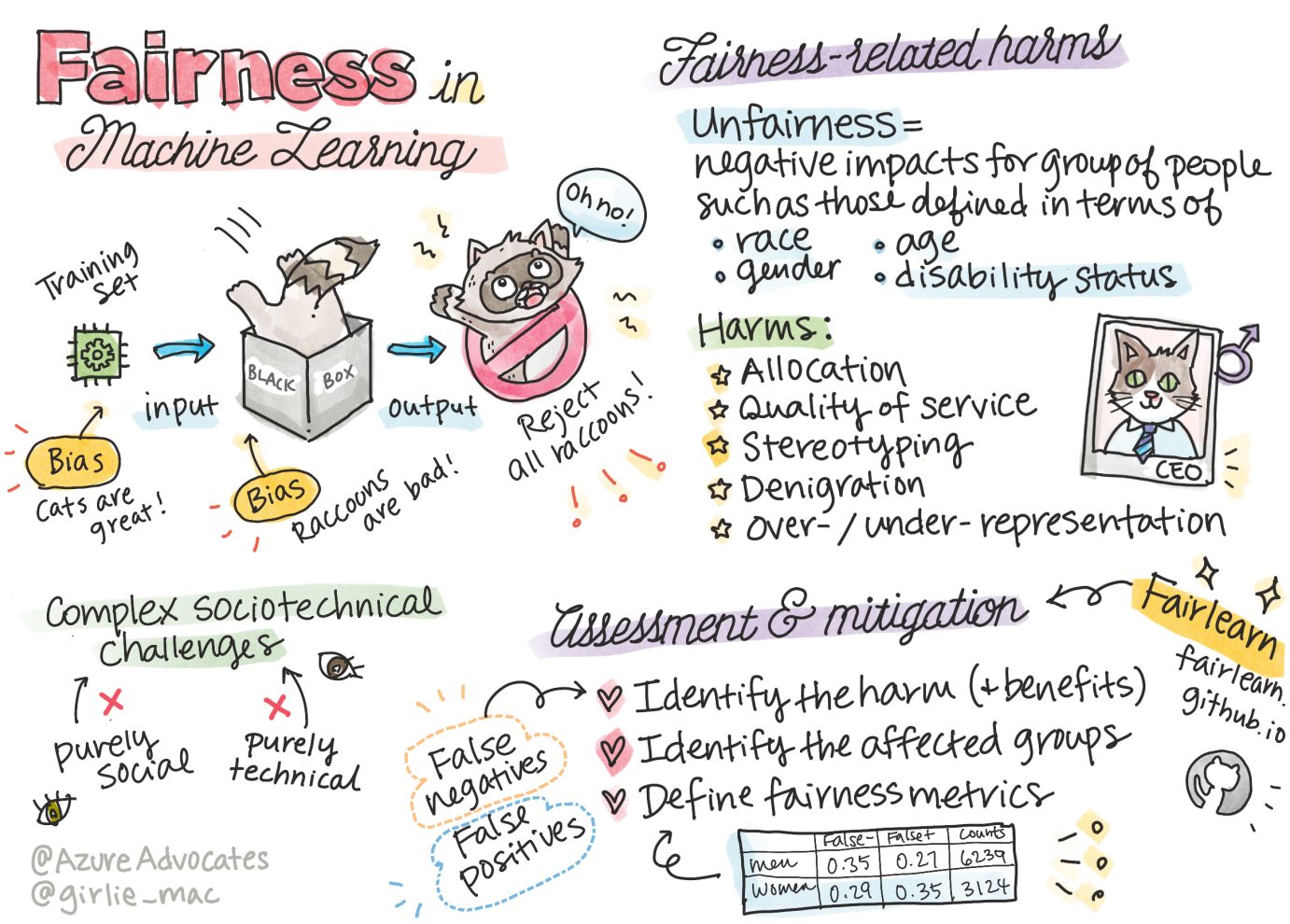
Here are items to watch and listen to:

This podcast where Amy Boyd discusses the evolution of AI

Assignment

Create a timeline

Fairness in Machine Learning



> Sketchnote by Tomomi Imura

Pre-lecture quiz

Introduction

In this curriculum, you will start to discover how machine learning can and is impacting our everyday lives. Even now, systems and models are involved in daily decision-making tasks, such as health care diagnoses or detecting fraud. So it is important that these models work well in order to provide fair outcomes for everyone.

Imagine what can happen when the data you are using to build these models lacks certain demographics, such as race, gender, political view, religion, or disproportionally represents such demographics. What about when the model's output is interpreted to favor some demographic? What is the consequence for the application?

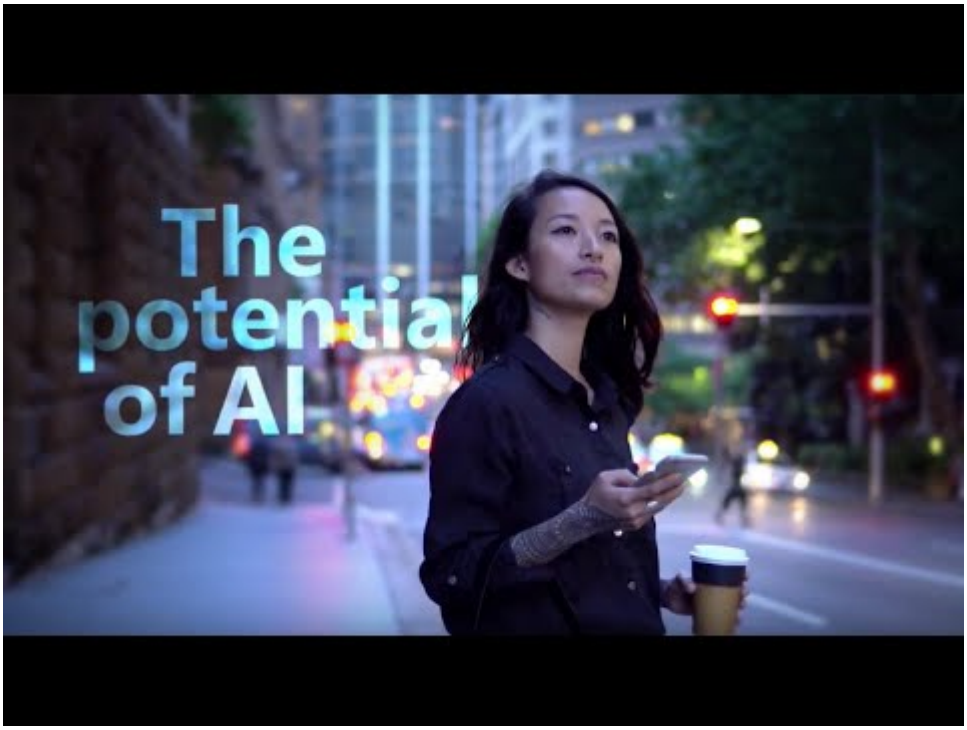
In this lesson, you will:

- Raise your awareness of the importance of fairness in machine learning.
- Learn about fairness-related harms.
- Learn about unfairness assessment and mitigation.

Prerequisite

As a prerequisite, please take the "Responsible AI Principles" Learn Path and watch the video below on the topic:

Learn more about Responsible AI by following this Learning Path



> 🎥 Click the image above for a video: Microsoft's Approach to Responsible AI

Unfairness in data and algorithms

> "If you torture the data long enough, it will confess to anything - Ronald Coase

This statement sounds extreme, but it is true that data can be manipulated to support any conclusion. Such manipulation can sometimes happen unintentionally. As humans, we all have bias,

and it's often difficult to consciously know when you are introducing bias in data.

Guaranteeing fairness in AI and machine learning remains a complex sociotechnical challenge. Meaning that it cannot be addressed from either purely social or technical perspectives.

Fairness-related harms

What do you mean by unfairness? "Unfairness" encompasses negative impacts, or "harms", for a group of people, such as those defined in terms of race, gender, age, or disability status.

The main fairness-related harms can be classified as:

- **Allocation**, if a gender or ethnicity for example is favored over another.
- **Quality of service**. If you train the data for one specific scenario but reality is much more complex, it leads to a poor performing service.
- **Stereotyping**. Associating a given group with pre-assigned attributes.
- **Denigration**. To unfairly criticize and label something or someone.
- **Over- or under- representation**. The idea is that a certain group is not seen in a certain profession, and any service or function that keeps promoting that is contributing to harm.

Let's take a look at the examples.

Allocation

Consider a hypothetical system for screening loan applications. The system tends to pick white men as better candidates over other groups. As a result, loans are withheld from certain applicants.

Another example would be an experimental hiring tool developed by a large corporation to screen candidates. The tool systemically discriminated against one gender by using the models were trained to prefer words associated with another. It resulted in penalizing candidates whose resumes contain words such as "women's rugby team".

✅ Do a little research to find a real-world example of something like this

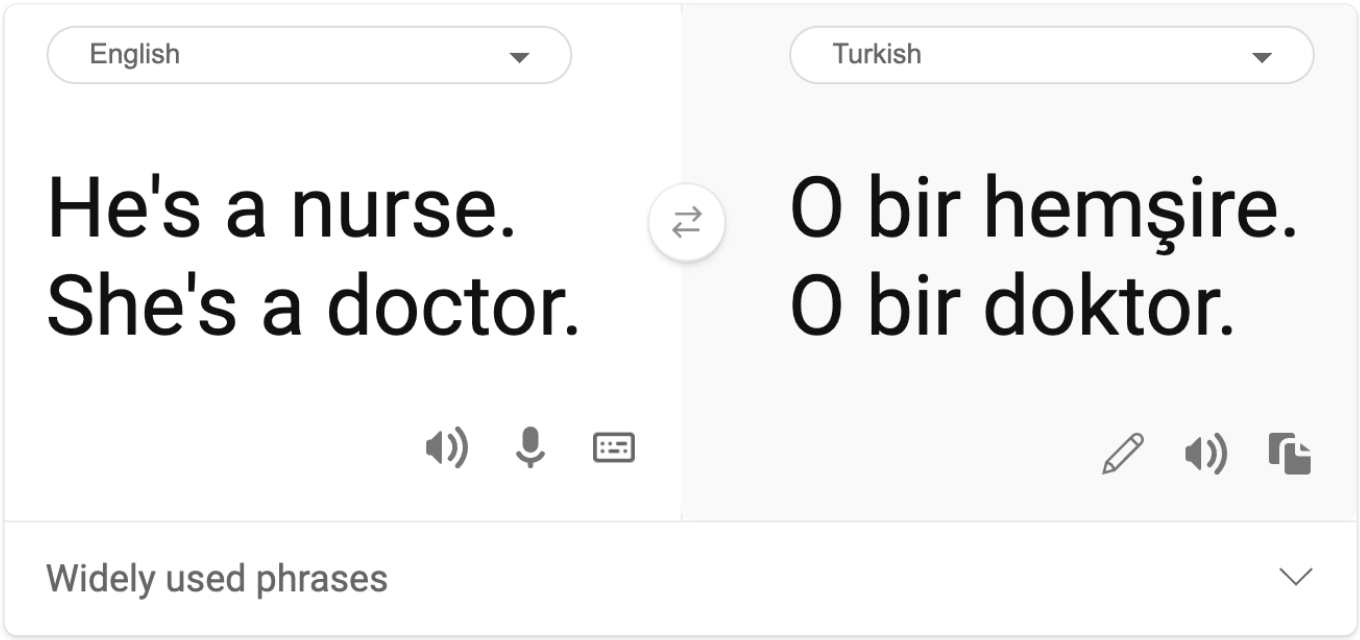
Quality of Service

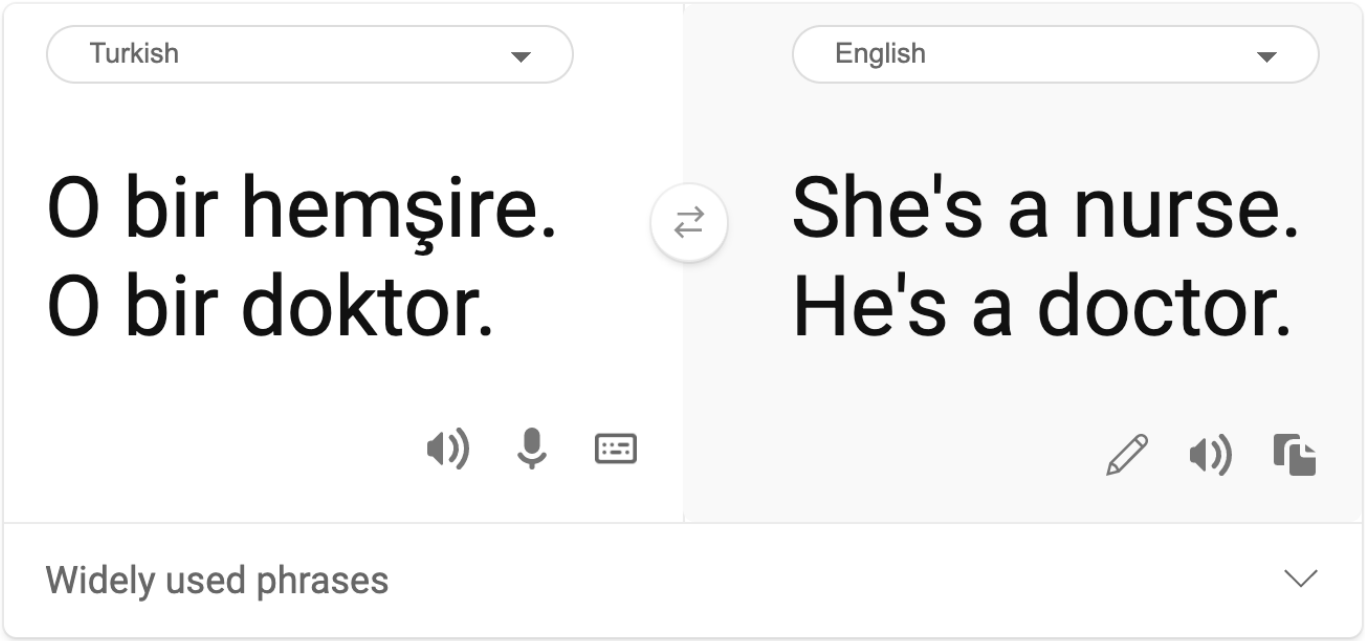
Researchers found that several commercial gender classifiers had higher error rates around images of women with darker skin tones as opposed to images of men with lighter skin tones. Reference

Another infamous example is a hand soap dispenser that could not seem to be able to sense people with dark skin. Reference

Stereotyping

Stereotypical gender view was found in machine translation. When translating “he is a nurse and she is a doctor” into Turkish, problems were encountered. Turkish is a genderless language which has one pronoun, “o” to convey a singular third person, but translating the sentence back from Turkish to English yields the stereotypical and incorrect as “she is a nurse and he is a doctor”.





Denigration

An image labeling technology infamously mislabeled images of dark-skinned people as gorillas. Mislabeling is harmful not just because the system made a mistake because it specifically applied a label that has a long history of being purposefully used to denigrate Black people.

> 📽️ Click the image above for a video: AI, Ain't I a Woman - a performance showing the harm caused by racist denigration by AI

Over- or under- representation

Skewed image search results can be a good example of this harm. When searching images of professions with an equal or higher percentage of men than women, such as engineering, or CEO, watch for results that are more heavily skewed towards a given gender.

> This search on Bing for 'CEO' produces pretty inclusive results

These five main types of harms are not mutually exclusive, and a single system can exhibit more than one type of harm. In addition, each case varies in its severity. For instance, unfairly labeling someone as a criminal is a much more severe harm than mislabeling an image. It's important, however, to remember that even relatively non-severe harms can make people feel alienated or singled out and the cumulative impact can be extremely oppressive.

✅ **Discussion**: Revisit some of the examples and see if they show different harms.

	Allocation	Quality of service	Stereotyping	Denigration	Over- or under-representation
Automated hiring system	x	x	x		x
Machine translation					
Photo labeling					

Detecting unfairness

There are many reasons why a given system behaves unfairly. Social biases, for example, might be reflected in the datasets used to train them. For example, hiring unfairness might have been exacerbated by over reliance on historical data. By using the patterns in resumes submitted to the company over a 10-year period, the model determined that men were more qualified because the majority of resumes came from men, a reflection of past male dominance across the tech industry.

Inadequate data about a certain group of people can be the reason for unfairness. For example, image classifiers a have higher rate of error for images of dark-skinned people because darker skin tones were underrepresented in the data.

Wrong assumptions made during development cause unfairness too. For example, a facial analysis system intended to predict who is going to commit a crime based on images of people's faces can lead to damaging assumptions. This could lead to substantial harms for people who are misclassified.

Understand your models and build in fairness

Although many aspects of fairness are not captured in quantitative fairness metrics, and it is not possible to fully remove bias from a system to guarantee fairness, you are still responsible to detect and to mitigate fairness issues as much as possible.

When you are working with machine learning models, it is important to understand your models by means of assuring their interpretability and by assessing and mitigating unfairness.

Let's use the loan selection example to isolate the case to figure out each factor's level of impact on the prediction.

Assessment methods

1. **Identify harms (and benefits)**. The first step is to identify harms and benefits. Think about how actions and decisions can affect both potential customers and a business itself.
2. **Identify the affected groups**. Once you understand what kind of harms or benefits that can occur, identify the groups that may be affected. Are these groups defined by gender, ethnicity, or social group?
3. **Define fairness metrics**. Finally, define a metric so you have something to measure against in your work to improve the situation.

Identify harms (and benefits)

What are the harms and benefits associated with lending? Think about false negatives and false positive scenarios:

False negatives (reject, but Y=1) - in this case, an applicant who will be capable of repaying a loan is rejected. This is an adverse event because the resources of the loans are withheld from qualified applicants.

False positives (accept, but Y=0) - in this case, the applicant does get a loan but eventually defaults. As a result, the applicant's case will be sent to a debt collection agency which can affect their future loan applications.

Identify affected groups

The next step is to determine which groups are likely to be affected. For example, in case of a credit card application, a model might determine that women should receive much lower credit limits compared with their spouses who share household assets. An entire demographic, defined by gender, is thereby affected.

Define fairness metrics

You have identified harms and an affected group, in this case, delineated by gender. Now, use the quantified factors to disaggregate their metrics. For example, using the data below, you can see that women have the largest false positive rate and men have the smallest, and that the opposite is true for false negatives.

✅ In a future lesson on Clustering, you will see how to build this 'confusion matrix' in code

	False positive rate	False negative rate	count
Women	0.37	0.27	54032
Men	0.31	0.35	28620
Non-binary	0.33	0.31	1266

This table tells us several things. First, we note that there are comparatively few non-binary people in the data. The data is skewed, so you need to be careful how you interpret these numbers.

In this case, we have 3 groups and 2 metrics. When we are thinking about how our system affects the group of customers with their loan applicants, this may be sufficient, but when you want to define larger number of groups, you may want to distill this to smaller sets of summaries. To do that, you can add more metrics, such as the largest difference or smallest ratio of each false negative and false positive.

✅ Stop and Think: What other groups are likely to be affected for loan application?

Mitigating unfairness

To mitigate unfairness, explore the model to generate various mitigated models and compare the tradeoffs it makes between accuracy and fairness to select the most fair model.

This introductory lesson does not dive deeply into the details of algorithmic unfairness mitigation, such as post-processing and reductions approach, but here is a tool that you may want to try.

Fairlearn

Fairlearn is an open-source Python package that allows you to assess your systems' fairness and mitigate unfairness.

The tool helps you to assesses how a model's predictions affect different groups, enabling you to compare multiple models by using fairness and performance metrics, and supplying a set of algorithms to mitigate unfairness in binary classification and regression.

- Learn how to use the different components by checking out the Fairlearn's GitHub
- Explore the user guide, examples
- Try some sample notebooks.
- Learn how to enable fairness assessments of machine learning models in Azure Machine Learning.
- Check out these sample notebooks for more fairness assessment scenarios in Azure Machine Learning.

🚀 Challenge

To prevent biases from being introduced in the first place, we should:

- have a diversity of backgrounds and perspectives among the people working on systems
- invest in datasets that reflect the diversity of our society
- develop better methods for detecting and correcting bias when it occurs

Think about real-life scenarios where unfairness is evident in model-building and usage. What else should we consider?

Post-lecture quiz

Review & Self Study

In this lesson, you have learned some basics of the concepts of fairness and unfairness in machine learning.

Watch this workshop to dive deeper into the topics:

- YouTube: Fairness-related harms in AI systems: Examples, assessment, and mitigation by Hanna Wallach and Miro Dudik Fairness-related harms in AI systems: Examples, assessment, and

mitigation - YouTube

Also, read:

- Microsoft's RAI resource center: Responsible AI Resources – Microsoft AI
- Microsoft's FATE research group: FATE: Fairness, Accountability, Transparency, and Ethics in AI - Microsoft Research

Explore the Fairlearn toolkit

Fairlearn

Read about Azure Machine Learning's tools to ensure fairness

- Azure Machine Learning

Assignment

Explore Fairlearn

Techniques of Machine Learning

The process of building, using, and maintaining machine learning models and the data they use is a very different process from many other development workflows. In this lesson, we will demystify the process, and outline the main techniques you need to know. You will:

- Understand the processes underpinning machine learning at a high level.
- Explore base concepts such as 'models', 'predictions', and 'training data'.

Pre-lecture quiz

Introduction

On a high level, the craft of creating machine learning (ML) processes is comprised of a number of steps:

1. **Decide on the question**. Most ML processes start by asking a question that cannot be answered by a simple conditional program or rules-based engine. These questions often revolve around predictions based on a collection of data.
2. **Collect and prepare data**. To be able to answer your question, you need data. The quality and, sometimes, quantity of your data will determine how well you can answer your initial question. Visualizing data is an important aspect of this phase. This phase also includes splitting the data into a training and testing group to build a model.
3. **Choose a training method**. Depending on your question and the nature of your data, you need to choose how you want to train a model to best reflect your data and make accurate predictions against it. This is the part of your ML process that requires specific expertise and, often, a considerable amount of experimentation.
4. **Train the model**. Using your training data, you'll use various algorithms to train a model to recognize patterns in the data. The model might leverage internal weights that can be adjusted to privilege certain parts of the data over others to build a better model.
5. **Evaluate the model**. You use never before seen data (your testing data) from your collected set to see how the model is performing.
6. **Parameter tuning**. Based on the performance of your model, you can redo the process using different parameters, or variables, that control the behavior of the algorithms used to train the model.
7. **Predict**. Use new inputs to test the accuracy of your model.

What question to ask

Computers are particularly skilled at discovering hidden patterns in data. This utility is very helpful for researchers who have questions about a given domain that cannot be easily answered by creating a conditionally-based rules engine. Given an actuarial task, for example, a data scientist might be able to construct handcrafted rules around the mortality of smokers vs non-smokers.

When many other variables are brought into the equation, however, a ML model might prove more efficient to predict future mortality rates based on past health history. A more cheerful example might be making weather predictions for the month of April in a given location based on data that includes latitude, longitude, climate change, proximity to the ocean, patterns of the jet stream, and more.

✅ This slide deck on weather models offers a historical perspective for using ML in weather analysis.

Pre-building tasks

Before starting to build your model, there are several tasks you need to complete. To test your question and form a hypothesis based on a model's predictions, you need to identify and configure several elements.

Data

To be able to answer your question with any kind of certainty, you need a good amount of data of the right type. There are two things you need to do at this point:

- **Collect data**. Keeping in mind the previous lesson on fairness in data analysis, collect your data with care. Be aware of the sources of this data, any inherent biases it might have, and document its origin.
- **Prepare data**. There are several steps in the data preparation process. You might need to collate data and normalize it if it comes from diverse sources. You can improve the data's quality and quantity through various methods such as converting strings to numbers (as we do in Clustering). You might also generate new data, based on the original (as we do in Classification). You can clean and edit the data (as we did prior to the Web App lesson). Finally, you might also need to randomize it and shuffle it, depending on your training techniques.

✅ After collecting and processing your data, take a moment to see if its shape will allow you to address your intended question. It may be that the data will not perform well in your given task, as we discover in our Clustering lessons!

Selecting your feature variable

A feature is a measurable property of your data. In many datasets it is expressed as a column heading like 'date' 'size' or 'color'. Your feature variable, usually represented as `y` in code, represents the answer to the question you are trying to ask of your data: in December, what **color** pumpkins will be cheapest? in San Francisco, what neighborhoods will have the best real estate **price**?

🎓 **Feature Selection and Feature Extraction** How do you know which variable to choose when building a model? You'll probably go through a process of feature selection or feature extraction to choose the right variables for the most performant model. They're not the same thing, however: "Feature extraction creates new features from functions of the original features, whereas feature selection returns a subset of the features." (source)

Visualize your data

An important aspect of the data scientist's toolkit is the power to visualize data using several excellent libraries such as Seaborn or MatPlotLib. Representing your data visually might allow you to

uncover hidden correlations that you can leverage. Your visualizations might also help you to uncover bias or unbalanced data (as we discover in Classification).

Split your dataset

Prior to training, you need to split your dataset into two or more parts of unequal size that still represent the data well.

- **Training**. This part of the dataset is fit to your model to train it. This set constitutes the majority of the original dataset.
- **Testing**. A test dataset is an independent group of data, often gathered from the original data, that you use to confirm the performance of the built model.
- **Validating**. A validation set is a smaller independent group of examples that you use to tune the model's hyperparameters, or architecture, to improve the model. Depending on your data's size and the question you are asking, you might not need to build this third set (as we note in Time Series Forecasting).

Building a model

Using your training data, your goal is to build a model, or a statistical representation of your data, using various algorithms to **train** it. Training a model exposes it to data and allows it to make assumptions about perceived patterns it discovers, validates, and accepts or rejects.

Decide on a training method

Depending on your question and the nature of your data, your will choose a method to train it. Stepping through Scikit-learn's documentation - which we use in this course - you can explore many ways to train a model. Depending on your experience, you might have to try several different methods to build the best model. You are likely to go through a process whereby data scientists evaluate the performance of a model by feeding it unseen data, checking for accuracy, bias, and other quality-degrading issues, and selecting the most appropriate training method for the task at hand.

Train a model

Armed with your training data, you are ready to 'fit' it to create a model. You will notice that in many ML libraries you will find the code 'model.fit' - it is at this time that you send in your data as an array of values (usually 'X') and a feature variable (usually 'y').

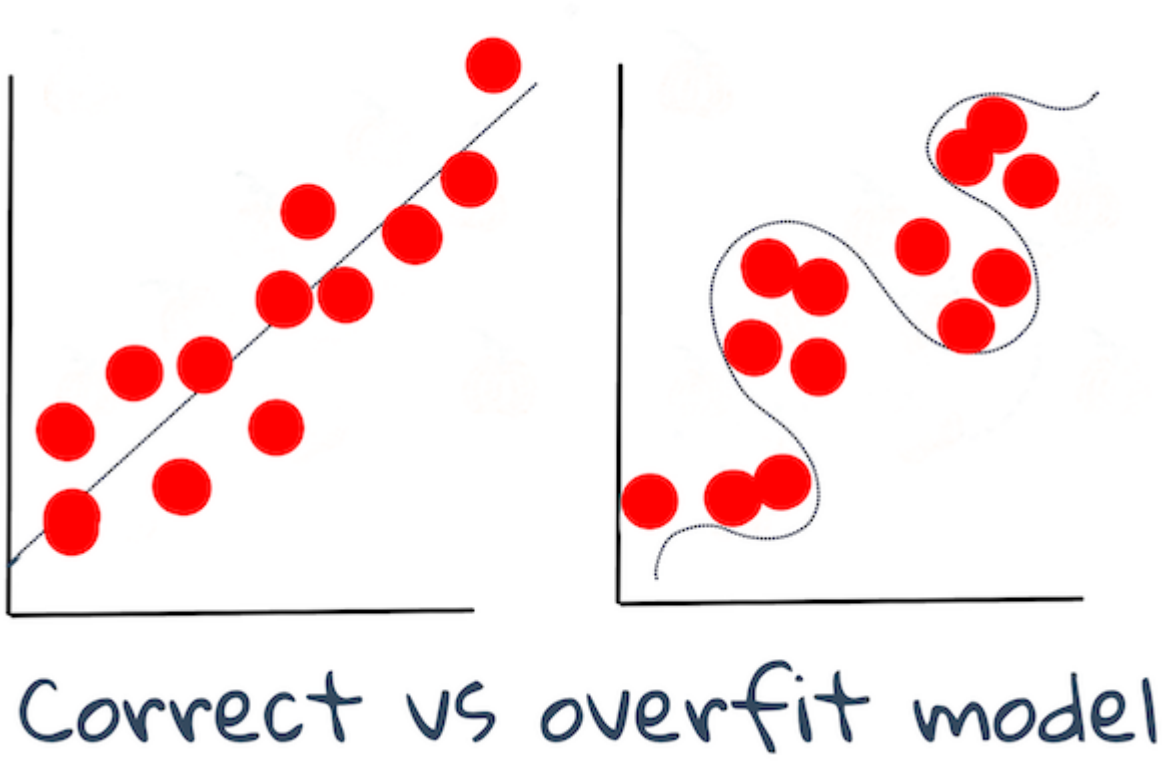
Evaluate the model

Once the training process is complete (it can take many iterations, or 'epochs', to train a large model), you will be able to evaluate the model's quality by using test data to gauge its performance. This data is a subset of the original data that the model has not previously analyzed. You can print out a table of metrics about your model's quality.

🎓 **Model fitting**

In the context of machine learning, model fitting refers to the accuracy of the model's underlying function as it attempts to analyze data with which it is not familiar.

🎓 **Underfitting** and **overfitting** are common problems that degrade the quality of the model, as the model fits either not well enough or too well. This causes the model to make predictions either too closely aligned or too loosely aligned with its training data. An overfit model predicts training data too well because it has learned the data's details and noise too well. An underfit model is not accurate as it can neither accurately analyze its training data nor data it has not yet 'seen'.



> Infographic by Jen Looper

Parameter tuning

Once your initial training is complete, observe the quality of the model and consider improving it by tweaking its 'hyperparameters'. Read more about the process in the documentation.

Prediction

This is the moment where you can use completely new data to test your model's accuracy. In an 'applied' ML setting, where you are building web assets to use the model in production, this process might involve gathering user input (a button press, for example) to set a variable and send it to the model for inference, or evaluation.

In these lessons, you will discover how to use these steps to prepare, build, test, evaluate, and predict - all the gestures of a data scientist and more, as you progress in your journey to become a 'full stack' ML engineer.

🚀Challenge

Draw a flow chart reflecting the steps of a ML practitioner. Where do you see yourself right now in the process? Where do you predict you will find difficulty? What seems easy to you?

Post-lecture quiz

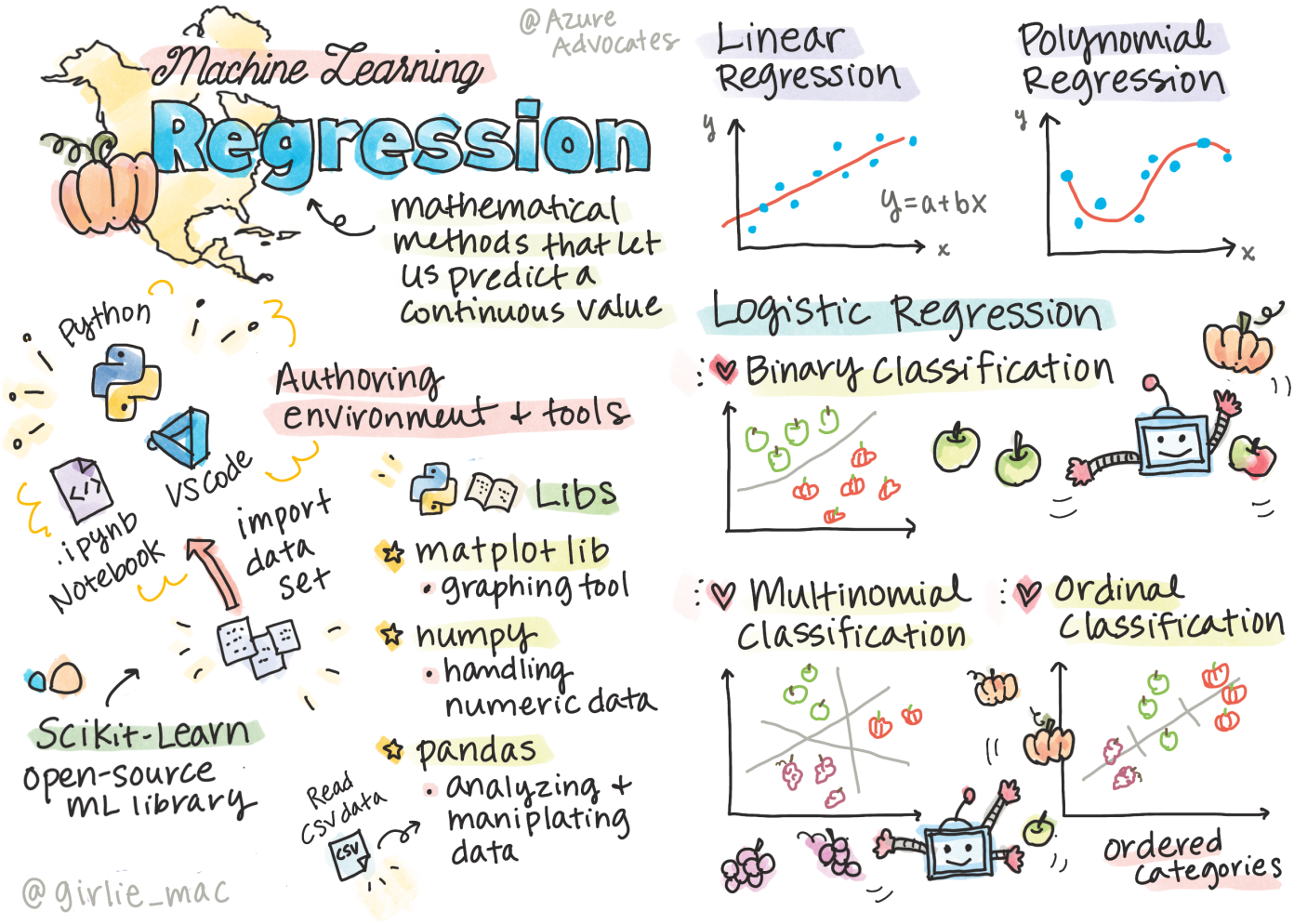
Review & Self Study

Search online for interviews with data scientists who discuss their daily work. Here is one.

Assignment

Interview a data scientist

Get started with Python and Scikit-learn for regression models



> Sketchnote by Tomomi Imura

Pre-lecture quiz

Introduction

In these four lessons, you will discover how to build regression models. We will discuss what these are for shortly. But before you do anything, make sure you have the right tools in place to start the process!

In this lesson, you will learn how to:

- Configure your computer for local machine learning tasks.
- Work with Jupyter notebooks.
- Use Scikit-learn, including installation.
- Explore linear regression with a hands-on exercise.

Installations and configurations

> 📽️ Click the image above for a video: using Python within VS Code.

1. **Install Python**. Ensure that Python is installed on your computer. You will use Python for many data science and machine learning tasks. Most computer systems already include a Python installation. There are useful Python Coding Packs available as well, to ease the setup for some users.

   Some usages of Python, however, require one version of the software, whereas others require a different version. For this reason, it's useful to work within a virtual environment.

2. **Install Visual Studio Code**. Make sure you have Visual Studio Code installed on your computer. Follow these instructions to install Visual Studio Code for the basic installation. You are going to use Python in Visual Studio Code in this course, so you might want to brush up on how to configure Visual Studio Code for Python development.

> Get comfortable with Python by working through this collection of Learn modules

3. **Install Scikit-learn**, by following these instructions. Since you need to ensure that you use Python 3, it's recommended that you use a virtual environment. Note, if you are installing this library on a M1 Mac, there are special instructions on the page linked above.
4. **Install Jupyter Notebook**. You will need to install the Jupyter package.

Your ML authoring environment

You are going to use **notebooks** to develop your Python code and create machine learning models. This type of file is a common tool for data scientists, and they can be identified by their suffix or extension `.ipynb`.

Notebooks are an interactive environment that allow the developer to both code and add notes and write documentation around the code which is quite helpful for experimental or research-oriented projects.

Exercise - work with a notebook

In this folder, you will find the file *notebook.ipynb*.

1. Open *notebook.ipynb* in Visual Studio Code.

   A Jupyter server will start with Python 3+ started. You will find areas of the notebook that can be `run`, pieces of code. You can run a code block, by selecting the icon that looks like a play button.

2. Select the `md` icon and add a bit of markdown, and the following text **# Welcome to your notebook**.

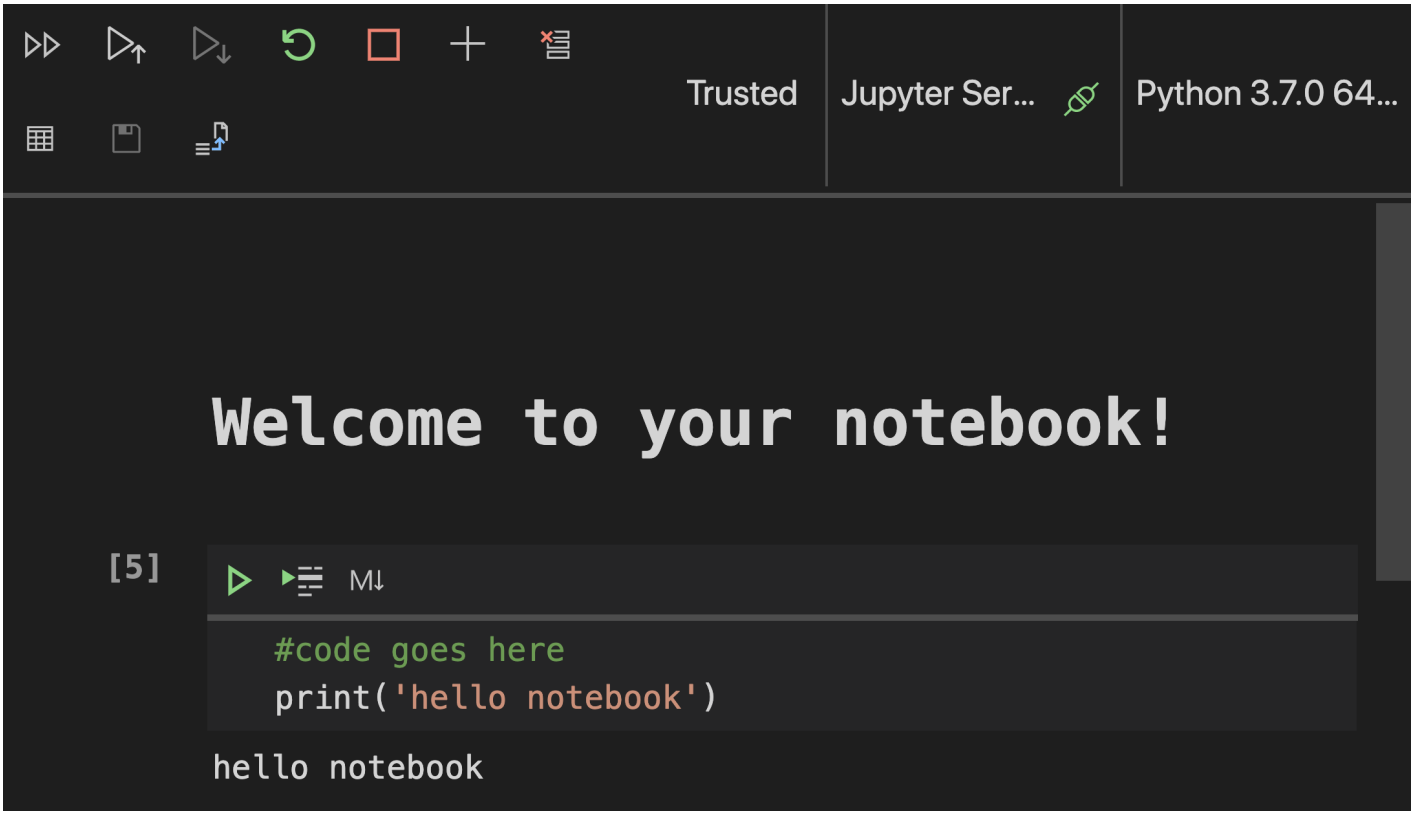
   Next, add some Python code.

3. Type **print("hello notebook'")** in the code block.

4. Select the arrow to run the code.

   You should see the printed statement:

```
output
hello notebook
```



You can interleaf your code with comments to self-document the notebook.

✅ Think for a minute how different a web developer's working environment is versus that of a data scientist.

Up and running with Scikit-learn

Now that Python is set up in your local environment, and you are comfortable with Jupyter notebooks, let's get equally comfortable with Scikit-learn (pronounce it `sci` as in `science`). Scikit-learn provides an extensive API to help you perform ML tasks.

According to their website, "Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities."

In this course, you will use Scikit-learn and other tools to build machine learning models to perform what we call 'traditional machine learning' tasks. We have deliberately avoided neural networks and deep learning, as they are better covered in our forthcoming 'AI for Beginners' curriculum.

Scikit-learn makes it straightforward to build models and evaluate them for use. It is primarily focused on using numeric data and contains several ready-made datasets for use as learning tools. It also includes pre-built models for students to try. Let's explore the process of loading prepackaged data and using a built in estimator first ML model with Scikit-learn with some basic data.

Exercise - your first Scikit-learn notebook

> This tutorial was inspired by the linear regression example on Scikit-learn's web site.

In the *notebook.ipynb* file associated to this lesson, clear out all the cells by pressing the 'trash can' icon.

In this section, you will work with a small dataset about diabetes that is built into Scikit-learn for learning purposes. Imagine that you wanted to test a treatment for diabetic patients. Machine Learning models might help you determine which patients would respond better to the treatment, based on combinations of variables. Even a very basic regression model, when visualized, might show information about variables that would help you organize your theoretical clinical trials.

✅ There are many types of regression methods, and which one you pick depends on the answer you're looking for. If you want to predict the probable height for a person of a given age, you'd use linear regression, as you're seeking a **numeric value**. If you're interested in discovering whether a type of cuisine should be considered vegan or not, you're looking for a **category assignment** so you would use logistic regression. You'll learn more about logistic regression later. Think a bit about some questions you can ask of data, and which of these methods would be more appropriate.

Let's get started on this task.

Import libraries

For this task we will import some libraries:

- **matplotlib**. It's a useful graphing tool and we will use it to create a line plot.
- **numpy**. numpy is a useful library for handling numeric data in Python.
- **sklearn**. This is the Scikit-learn library.

Import some libraries to help with your tasks.

1. Add imports by typing the following code:

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, model_selection
```

Above you are importing `matplottlib`, `numpy` and you are importing `datasets`, `linear_model` and `model_selection` from `sklearn`. `model_selection` is used for splitting data into training and test sets.

The diabetes dataset

The built-in diabetes dataset includes 442 samples of data around diabetes, with 10 feature variables, some of which include:

age: age in years bmi: body mass index bp: average blood pressure s1 tc: T-Cells (a type of white blood cells)

✅ This dataset includes the concept of 'sex' as a feature variable important to research around diabetes. Many medical datasets include this type of binary classification. Think a bit about how categorizations such as this might exclude certain parts of a population from treatments.

Now, load up the X and y data.

> 🎓 Remember, this is supervised learning, and we need a named 'y' target.

In a new code cell, load the diabetes dataset by calling `load_diabetes()`. The input `return_X_y=True` signals that `X` will be a data matrix, and `y` will be the regression target.

1. Add some print commands to show the shape of the data matrix and its first element:

```python
X, y = datasets.load_diabetes(return_X_y=True)
print(X.shape)
print(X[0])
```

What you are getting back as a response, is a tuple. What you are doing is to assign the two first values of the tuple to `X` and `y` respectively. Learn more about tuples.

You can see that this data has 442 items shaped in arrays of 10 elements:

```text
(442, 10)
[ 0.03807591  0.05068012  0.06169621  0.02187235 -0.0442235  -0.03482076
-0.04340085 -0.00259226  0.01990842 -0.01764613]
```

✅ Think a bit about the relationship between the data and the regression target. Linear regression predicts relationships between feature X and target variable y. Can you find the target for the diabetes dataset in the documentation? What is this dataset demonstrating, given that target?

2. Next, select a portion of this dataset to plot by arranging it into a new array using numpy's `newaxis` function. We are going to use linear regression to generate a line between values in this data, according to a pattern it determines.

```python
X = X[:, np.newaxis, 2]
```

✅ At any time, print out the data to check its shape.

3. Now that you have data ready to be plotted, you can see if a machine can help determine a logical split between the numbers in this dataset. To do this, you need to split both the data (X) and the target (y) into test and training sets. Scikit-learn has a straightforward way to do this; you can split your test data at a given point.

```python
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y
```

4. Now you are ready to train your model! Load up the linear regression model and train it with your X and y training sets using `model.fit()` :

```python
model = linear_model.LinearRegression()
model.fit(X_train, y_train)
```

✅ `model.fit()` is a function you'll see in many ML libraries such as TensorFlow

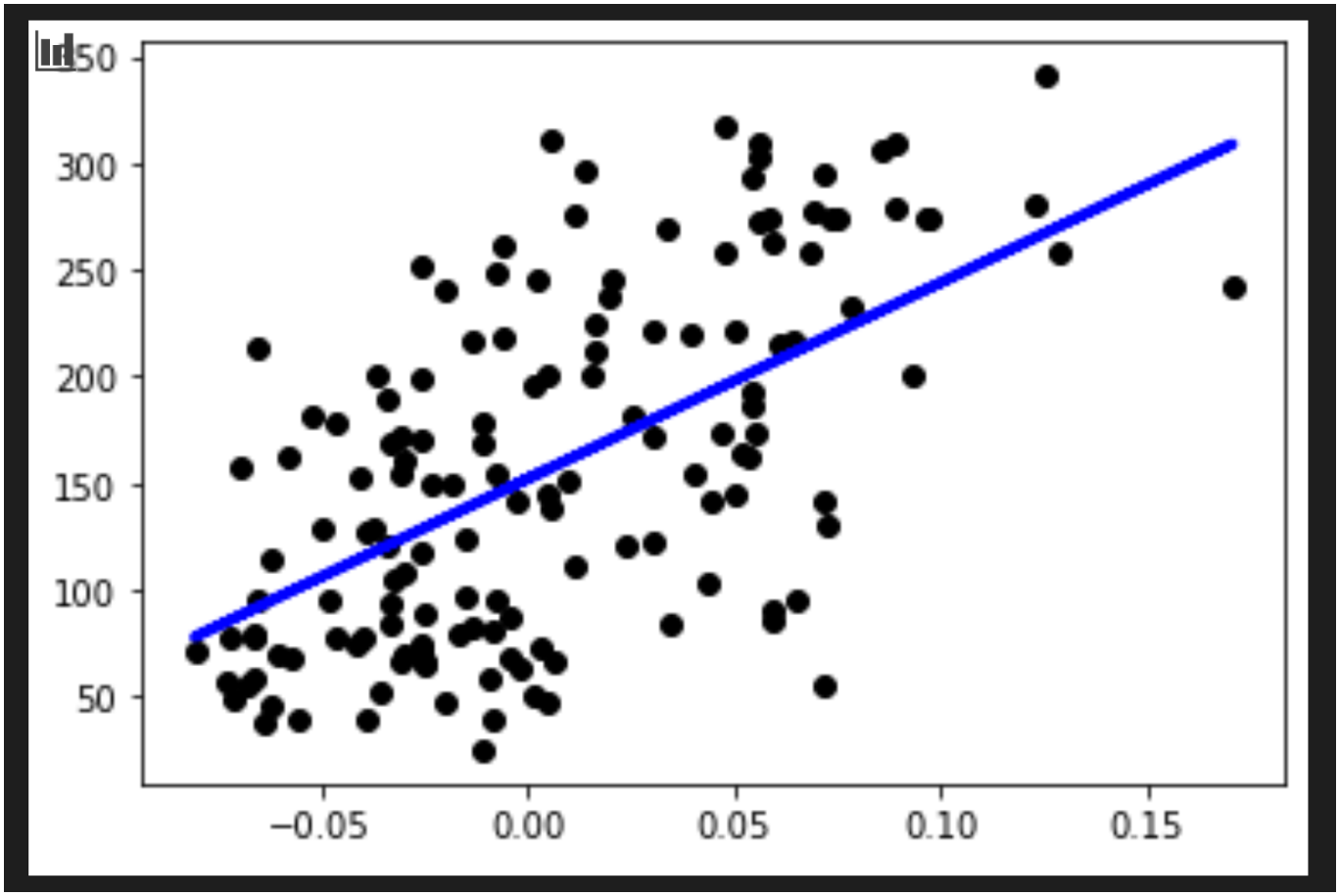
5. Then, create a prediction using test data, using the function `predict()` . This will be used to draw the line between data groups

```python
y_pred = model.predict(X_test)
```

6. Now it's time to show the data in a plot. Matplotlib is a very useful tool for this task. Create a scatterplot of all the X and y test data, and use the prediction to draw a line in the most appropriate place, between the model's data groupings.

python

```
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.show()
```



✅ Think a bit about what's going on here. A straight line is running through many small dots of data, but what is it doing exactly? Can you see how you should be able to use this line to predict where a new, unseen data point should fit in relationship to the plot's y axis? Try to put into words the practical use of this model.

Congratulations, you built your first linear regression model, created a prediction with it, and displayed it in a plot!

🚀Challenge

Plot a different variable from this dataset. Hint: edit this line: `X = X[:, np.newaxis, 2]` . Given this dataset's target, what are you able to discover about the progression of diabetes as a disease?

Post-lecture quiz

Review & Self Study

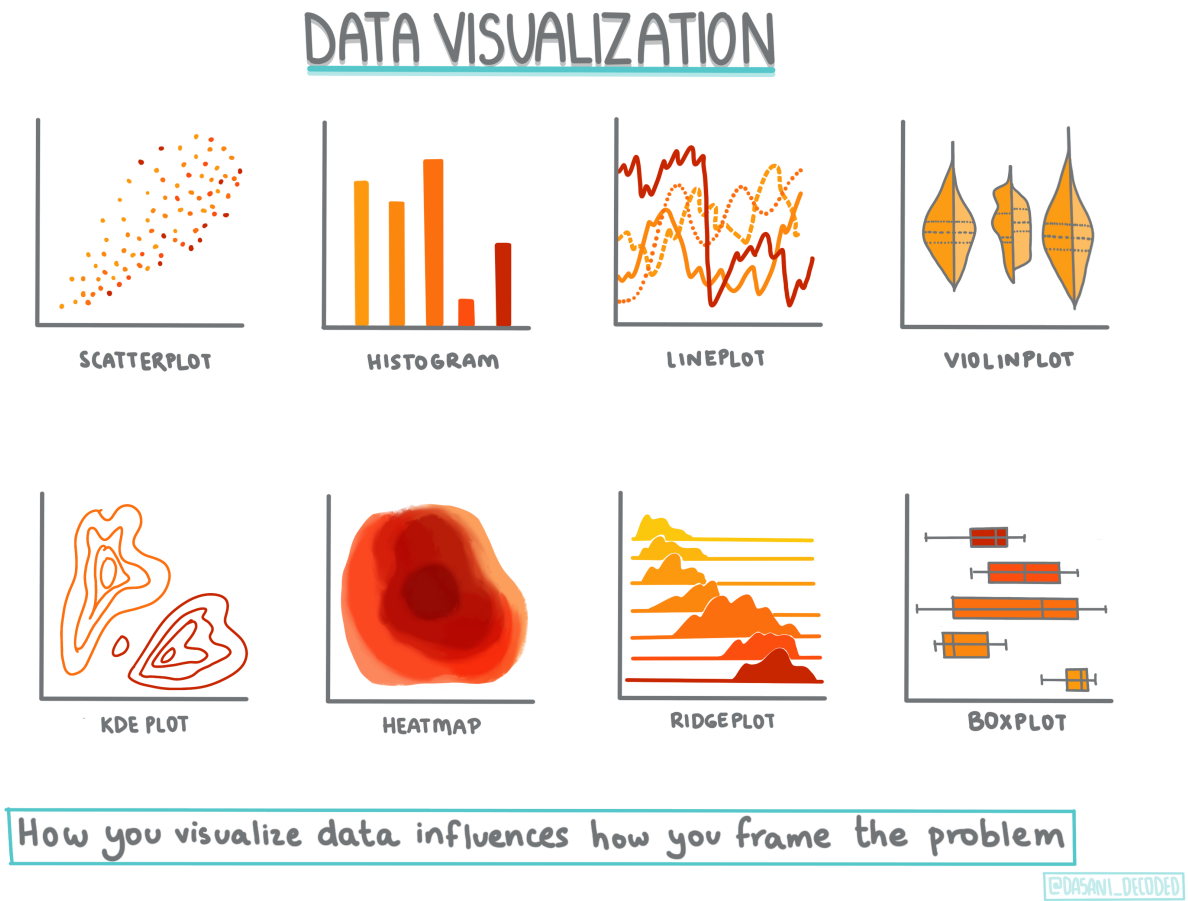
In this tutorial, you worked with simple linear regression, rather than univariate or multiple linear regression. Read a little about the differences between these methods, or take a look at this video

Read more about the concept of regression and think about what kinds of questions can be answered by this technique. Take this tutorial to deepen your understanding.

Assignment

A different dataset

Build a regression model using Scikit-learn: prepare and visualize data



Infographic by Dasani Madipalli

Pre-lecture quiz

Introduction

Now that you are set up with the tools you need to start tackling machine learning model building with Scikit-learn, you are ready to start asking questions of your data. As you work with data and apply ML solutions, it's very important to understand how to ask the right question to properly unlock the potentials of your dataset.

In this lesson, you will learn:

- How to prepare your data for model-building.
- How to use Matplotlib for data visualization.

Asking the right question of your data

The question you need answered will determine what type of ML algorithms you will leverage. And the quality of the answer you get back will be heavily dependent on the nature of your data.

Take a look at the data provided for this lesson. You can open this .csv file in VS Code. A quick skim immediately shows that there are blanks and a mix of strings and numeric data. There's also a strange column called 'Package' where the data is a mix between 'sacks', 'bins' and other values. The data, in fact, is a bit of a mess.

In fact, it is not very common to be gifted a dataset that is completely ready to use to create a ML model out of the box. In this lesson, you will learn how to prepare a raw dataset using standard Python libraries. You will also learn various techniques to visualize the data.

Case study: 'the pumpkin market'

In this folder you will find a .csv file in the root `data` folder called US-pumpkins.csv which includes 1757 lines of data about the market for pumpkins, sorted into groupings by city. This is raw data extracted from the Specialty Crops Terminal Markets Standard Reports distributed by the United States Department of Agriculture.

Preparing data

This data is in the public domain. It can be downloaded in many separate files, per city, from the USDA web site. To avoid too many separate files, we have concatenated all the city data into one spreadsheet, thus we have already *prepared* the data a bit. Next, let's take a closer look at the data.

The pumpkin data - early conclusions

What do you notice about this data? You already saw that there is a mix of strings, numbers, blanks and strange values that you need to make sense of.

What question can you ask of this data, using a Regression technique? What about "Predict the price of a pumpkin for sale during a given month". Looking again at the data, there are some changes you need to make to create the data structure necessary for the task.

Exercise - analyze the pumpkin data

Let's use Pandas, (the name stands for `Python Data Analysis`) a tool very useful for shaping data, to analyze and prepare this pumpkin data.

First, check for missing dates

You will first need to take steps to check for missing dates:

1. Convert the dates to a month format (these are US dates, so the format is `MM/DD/YYYY`).
2. Extract the month to a new column.

Open the *notebook.ipynb* file in Visual Studio Code and import the spreadsheet in to a new Pandas dataframe.

1. Use the `head()` function to view the first five rows.

```python
import pandas as pd
pumpkins = pd.read_csv('../../data/US-pumpkins.csv')
pumpkins.head()
```

✅ What function would you use to view the last five rows?

2. Check if there is missing data in the current dataframe:

```python
pumpkins.isnull().sum()
```

There is missing data, but maybe it won't matter for the task at hand.

3. To make your dataframe easier to work with, drop several of its columns, using `drop()`, keeping only the columns you need:

```python
new_columns = ['Package', 'Month', 'Low Price', 'High Price', 'Date']
pumpkins = pumpkins.drop([c for c in pumpkins.columns if c not in new_cc
```

Second, determine average price of pumpkin

Think about how to determine the average price of a pumpkin in a given month. What columns would you pick for this task? Hint: you'll need 3 columns.

Solution: take the average of the `Low Price` and `High Price` columns to populate the new Price column, and convert the Date column to only show the month. Fortunately, according to the check above, there is no missing data for dates or prices.

1. To calculate the average, add the following code:

```python
price = (pumpkins['Low Price'] + pumpkins['High Price']) / 2

month = pd.DatetimeIndex(pumpkins['Date']).month
```

✅ Feel free to print any data you'd like to check using `print(month)`.

2. Now, copy your converted data into a fresh Pandas dataframe:

```python
new_pumpkins = pd.DataFrame({'Month': month, 'Package': pumpkins['Packag
```

Printing out your dataframe will show you a clean, tidy dataset on which you can build your new regression model.

But wait! There's something odd here

If you look at the `Package` column, pumpkins are sold in many different configurations. Some are sold in '1 1/9 bushel' measures, and some in '1/2 bushel' measures, some per pumpkin, some per

pound, and some in big boxes with varying widths.

> Pumpkins seem very hard to weigh consistently

Digging into the original data, it's interesting that anything with `Unit of Sale` equalling 'EACH' or 'PER BIN' also have the `Package` type per inch, per bin, or 'each'. Pumpkins seem to be very hard to weigh consistently, so let's filter them by selecting only pumpkins with the string 'bushel' in their `Package` column.

1. Add a filter at the top of the file, under the initial .csv import:

```python
pumpkins = pumpkins[pumpkins['Package'].str.contains('bushel', case=True
```

If you print the data now, you can see that you are only getting the 415 or so rows of data containing pumpkins by the bushel.

But wait! There's one more thing to do

Did you notice that the bushel amount varies per row? You need to normalize the pricing so that you show the pricing per bushel, so do some math to standardize it.

1. Add these lines after the block creating the new_pumpkins dataframe:

```python
new_pumpkins.loc[new_pumpkins['Package'].str.contains('1 1/9'), 'Price']

new_pumpkins.loc[new_pumpkins['Package'].str.contains('1/2'), 'Price'] =
```

✅ According to The Spruce Eats, a bushel's weight depends on the type of produce, as it's a volume measurement. "A bushel of tomatoes, for example, is supposed to weigh 56 pounds... Leaves and greens take up more space with less weight, so a bushel of spinach is only 20 pounds." It's all pretty complicated! Let's not bother with making a bushel-to-pound conversion, and instead price by the bushel. All this study of bushels of pumpkins, however, goes to show how very important it is to understand the nature of your data!

Now, you can analyze the pricing per unit based on their bushel measurement. If you print out the data one more time, you can see how it's standardized.

✅ Did you notice that pumpkins sold by the half-bushel are very expensive? Can you figure out why? Hint: little pumpkins are way pricier than big ones, probably because there are so many more of them per bushel, given the unused space taken by one big hollow pie pumpkin.

Visualization Strategies

Part of the data scientist's role is to demonstrate the quality and nature of the data they are working with. To do this, they often create interesting visualizations, or plots, graphs, and charts, showing different aspects of data. In this way, they are able to visually show relationships and gaps that are otherwise hard to uncover.

Visualizations can also help determine the machine learning technique most appropriate for the data. A scatterplot that seems to follow a line, for example, indicates that the data is a good candidate for a linear regression exercise.

One data visualization libary that works well in Jupyter notebooks is Matplotlib (which you also saw in the previous lesson).

> Get more experience with data visualization in these tutorials.

Exercise - experiment with Matplotlib

Try to create some basic plots to display the new dataframe you just created. What would a basic line plot show?

1. Import Matplotlib at the top of the file, under the Pandas import:

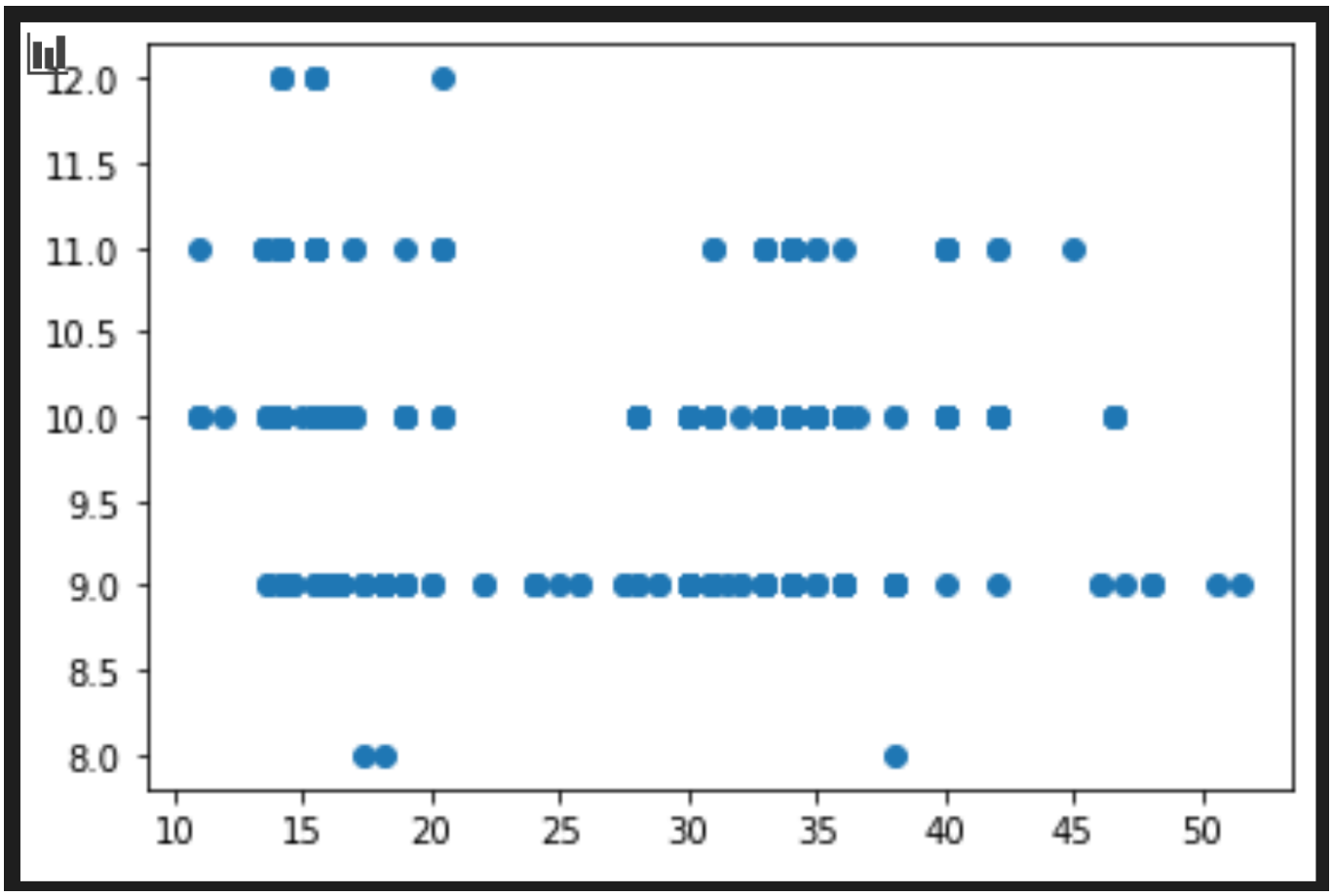
```python
import matplotlib.pyplot as plt
```

2. Rerun the entire notebook to refresh.

3. At the bottom of the notebook, add a cell to plot the data as a box:

```python
price = new_pumpkins.Price
month = new_pumpkins.Month
```

```
plt.scatter(price, month)
plt.show()
```



Is this a useful plot? Does anything about it surprise you?

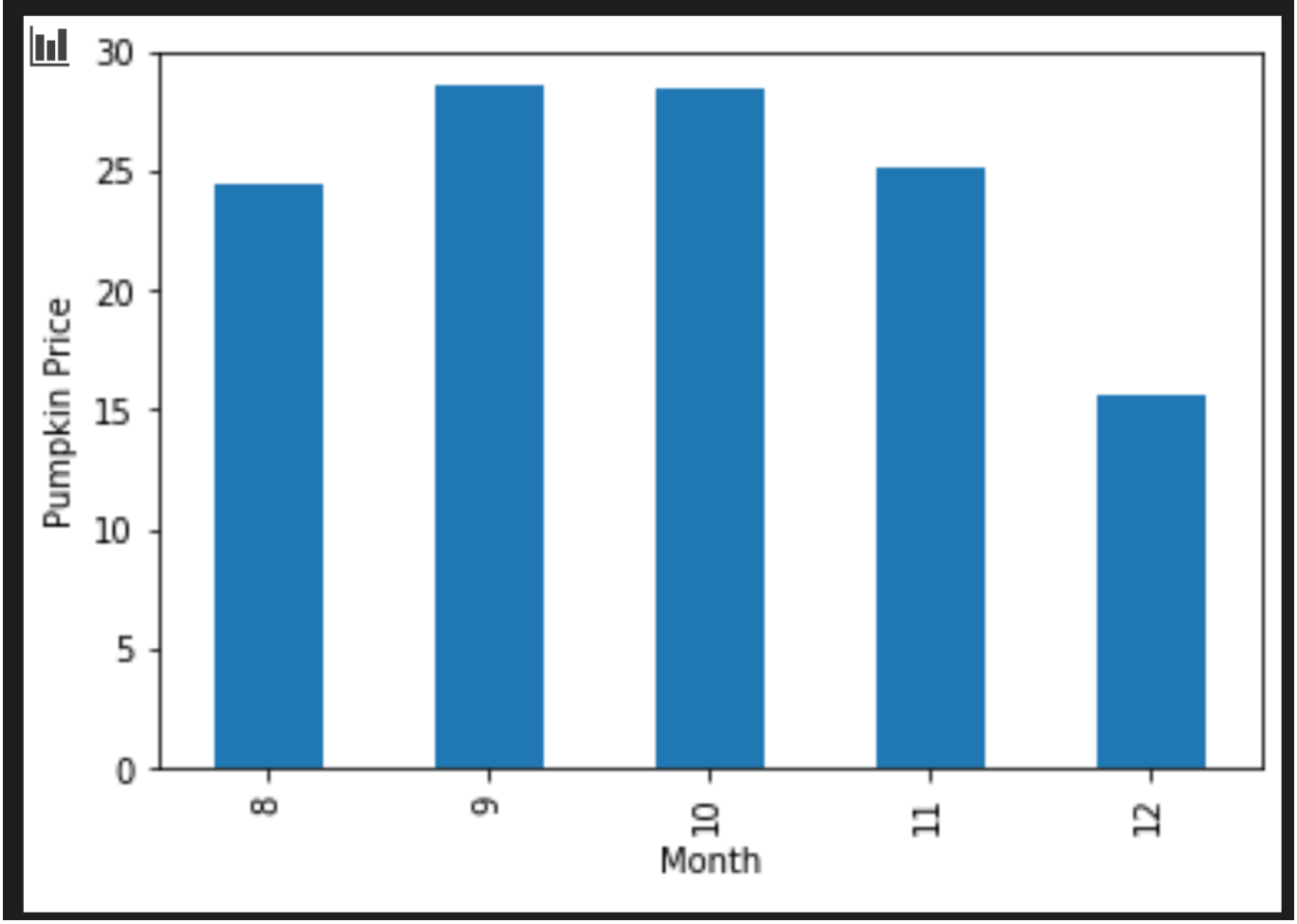
It's not particularly useful as all it does is display in your data as a spread of points in a given month.

Make it useful

To get charts to display useful data, you usually need to group the data somehow. Let's try creating a plot where the y axis shows the months and the data demonstrates the distribution of data.

1. Add a cell to create a grouped bar chart:

```python
new_pumpkins.groupby(['Month'])['Price'].mean().plot(kind='bar')
plt.ylabel("Pumpkin Price")
```



This is a more useful data visualization! It seems to indicate that the highest price for pumpkins occurs in September and October. Does that meet your expectation? Why or why not?

🚀Challenge

Explore the different types of visualization that M Matplotlib offers. Which types are most appropriate for regression problems?

Post-lecture quiz

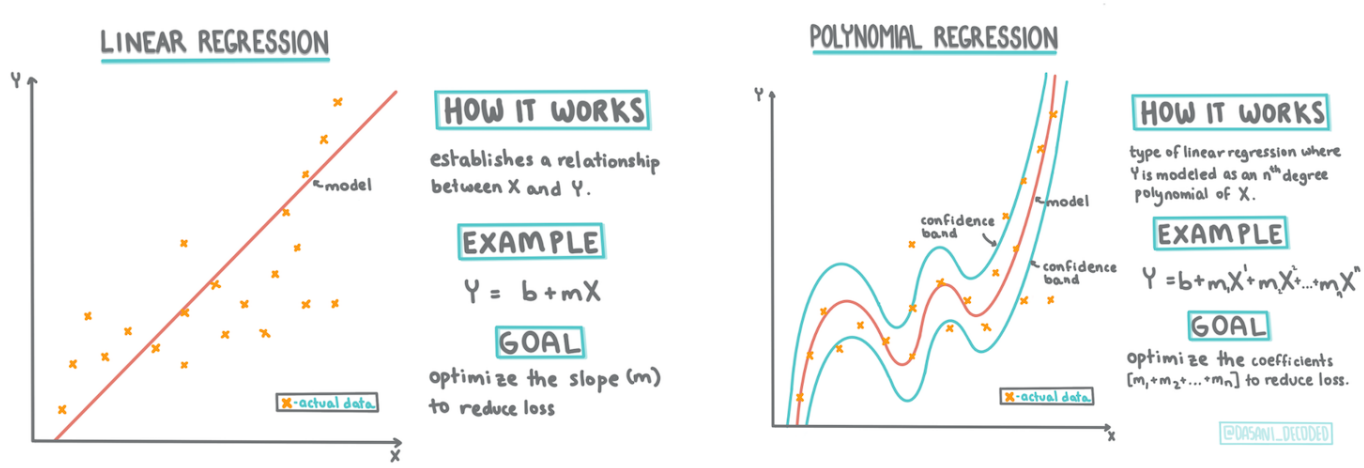
Review & Self Study

Take a look at the many ways to visualize data. Make a list of the various libraries available and note which are best for given types of tasks, for example 2D visualizations vs. 3D visualizations. What do you discover?

Assignment

Exploring visualization

Build a regression model using Scikit-learn: regression two ways



> Infographic by Dasani Madipalli

Pre-lecture quiz

Introduction

So far you have explored what regression is with sample data gathered from the pumpkin pricing dataset that we will use throughout this lesson. You have also visualized it using Matplotlib.

Now you are ready to dive deeper into regression for ML. In this lesson, you will learn more about two types of regression: *basic linear regression* and *polynomial regression*, along with some of the math underlying these techniques.

> Throughout this curriculum, we assume minimal knowledge of math, and seek to make it accessible for students coming from other fields, so watch for notes, callouts, diagrams,

> and other learning tools to aid in comprehension.

Prerequisite

You should be familiar by now with the structure of the pumpkin data that we are examining. You can find it preloaded and pre-cleaned in this lesson's *notebook.ipynb* file. In the file, the pumpkin price is displayed per bushel in a new dataframe. Make sure you can run these notebooks in kernels in Visual Studio Code.

Preparation

As a reminder, you are loading this data so as to ask questions of it.

- When is the best time to buy pumpkins?
- What price can I expect of a case of miniature pumpkins?
- Should I buy them in half-bushel baskets or by the 1 1/9 bushel box? Let's keep digging into this data.

In the previous lesson, you created a Pandas dataframe and populated it with part of the original dataset, standardizing the pricing by the bushel. By doing that, however, you were only able to gather about 400 datapoints and only for the fall months.

Take a look at the data that we preloaded in this lesson's accompanying notebook. The data is preloaded and an initial scatterplot is charted to show month data. Maybe we can get a little more detail about the nature of the data by cleaning it more.

A linear regression line

As you learned in Lesson 1, the goal of a linear regression exercise is to be able to plot a line to:

- **Show variable relationships**. Show the relationship between variables
- **Make predictions**. Make accurate predictions on where a new datapoint would fall in relationship to that line.

  It is typical of **Least-Squares Regression** to draw this type of line. The term 'least-squares' means that all the datapoints surrounding the regression line are squared and then added up. Ideally, that final sum is as small as possible, because we want a low number of errors, or `least-squares`.

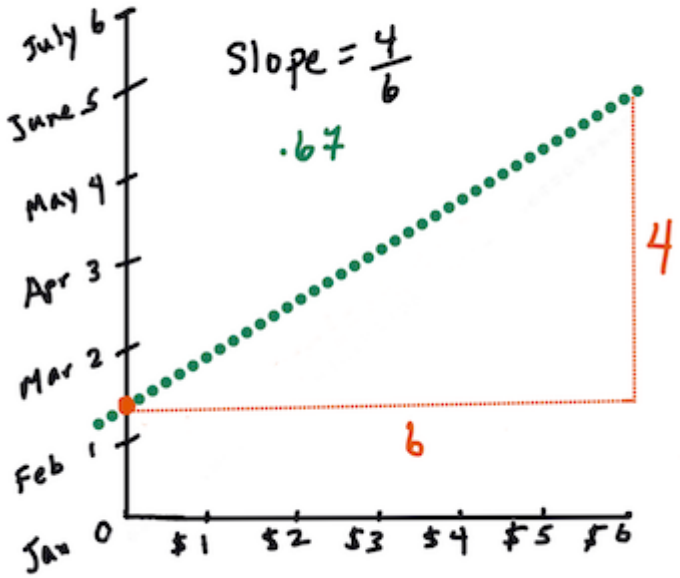
We do so since we want to model a line that has the least cumulative distance from all of our data points. We also square the terms before adding them since we are concerned with its magnitude rather than its direction.

Show me the math

This line, called the line of best fit can be expressed by an equation:

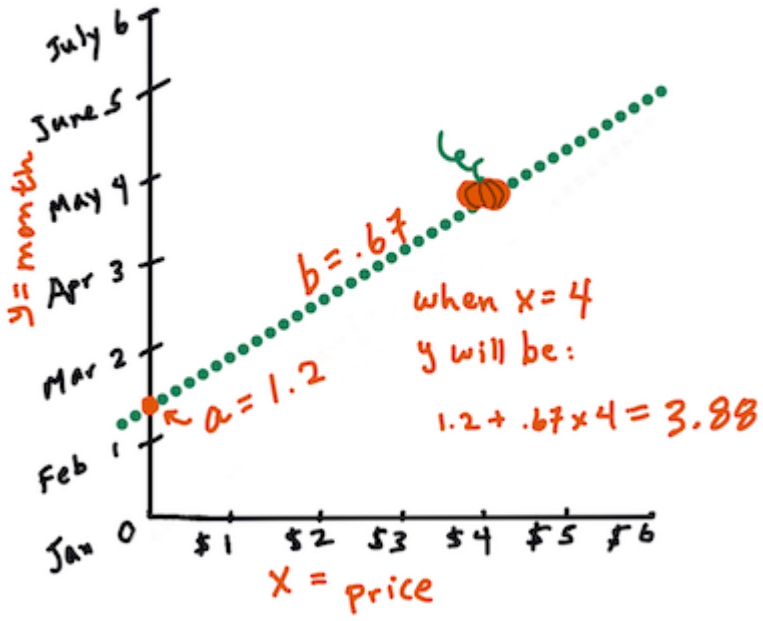
```
Y = a + bX
```

`X` is the 'explanatory variable'. `Y` is the 'dependent variable'. The slope of the line is `b` and `a` is the y-intercept, which refers to the value of `Y` when `X = 0`.



First, calculate the slope `b`. Infographic by Jen Looper

In other words, and referring to our pumpkin data's original question: "predict the price of a pumpkin per bushel by month", `X` would refer to the price and `Y` would refer to the month of sale.



Calculate the value of Y. If you're paying around $4, it must be April! Infographic by Jen Looper

The math that calculates the line must demonstrate the slope of the line, which is also dependent on the intercept, or where `Y` is situated when `X = 0`.

You can observe the method of calculation for these values on the Math is Fun web site. Also visit this Least-squares calculator to watch how the numbers' values impact the line.

Correlation

One more term to understand is the **Correlation Coefficient** between given X and Y variables. Using a scatterplot, you can quickly visualize this coefficient. A plot with datapoints scattered in a neat line have high correlation, but a plot with datapoints scattered everywhere between X and Y have a low correlation.

A good linear regression model will be one that has a high (nearer to 1 than 0) Correlation Coefficient using the Least-Squares Regression method with a line of regression.

✅ Run the notebook accompanying this lesson and look at the City to Price scatterplot. Does the data associating City to Price for pumpkin sales seem to have high or low correlation, according to your visual interpretation of the scatterplot?

Prepare your data for regression

Now that you have an understanding of the math behind this exercise, create a Regression model to see if you can predict which package of pumpkins will have the best pumpkin prices. Someone buying pumpkins for a holiday pumpkin patch might want this information to be able to optimize their purchases of pumpkin packages for the patch.

Since you'll use Scikit-learn, there's no reason to do this by hand (although you could!). In the main data-processing block of your lesson notebook, add a library from Scikit-learn to automatically convert all string data to numbers:

```python
from sklearn.preprocessing import LabelEncoder

new_pumpkins.iloc[:, 0:-1] = new_pumpkins.iloc[:, 0:-1].apply(LabelEncoder
new_pumpkins.iloc[:, 0:-1] = new_pumpkins.iloc[:, 0:-1].apply(LabelEncoder
```

If you look at the new_pumpkins dataframe now, you see that all the strings are now numeric. This makes it harder for you to read but much more intelligible for Scikit-learn! Now you can make more educated decisions (not just based on eyeballing a scatterplot) about the data that is best suited to regression.

Try to find a good correlation between two points of your data to potentially build a good predictive model. As it turns out, there's only weak correlation between the City and Price:

```python
print(new_pumpkins['City'].corr(new_pumpkins['Price']))
0.32363971816089226
```

However there's a bit better correlation between the Package and its Price. That makes sense, right? Normally, the bigger the produce box, the higher the price.

```python
print(new_pumpkins['Package'].corr(new_pumpkins['Price']))
0.6061712937226021
```

A good question to ask of this data will be: 'What price can I expect of a given pumpkin package?'

Let's build this regression model

Building a linear model

Before building your model, do one more tidy-up of your data. Drop any null data and check once more what the data looks like.

python

```
new_pumpkins.dropna(inplace=True)
new_pumpkins.info()
```

Then, create a new dataframe from this minimal set and print it out:

python

```
new_columns = ['Package', 'Price']
lin_pumpkins = new_pumpkins.drop([c for c in new_pumpkins.columns if c not

lin_pumpkins
```

output

```
    Package    Price
70    0    13.636364
71    0    16.363636
72    0    16.363636
73    0    15.454545
74    0    13.636364
...    ...    ...
1738    2    30.000000
1739    2    28.750000
1740    2    25.750000
1741    2    24.000000
1742    2    24.000000
415 rows × 2 columns
```

1. Now you can assign your X and y coordinate data:

python

```
X = lin_pumpkins.values[:, :1]
y = lin_pumpkins.values[:, 1:2]
```

✅ What's going on here? You're using Python slice notation to create arrays to populate `X` and `y`.

2. Next, start the regression model-building routines:

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
lin_reg = LinearRegression()
lin_reg.fit(X_train,y_train)

pred = lin_reg.predict(X_test)

accuracy_score = lin_reg.score(X_train,y_train)
print('Model Accuracy: ', accuracy_score)
```

Because the correlation isn't particularly good, the model produced isn't terribly accurate.

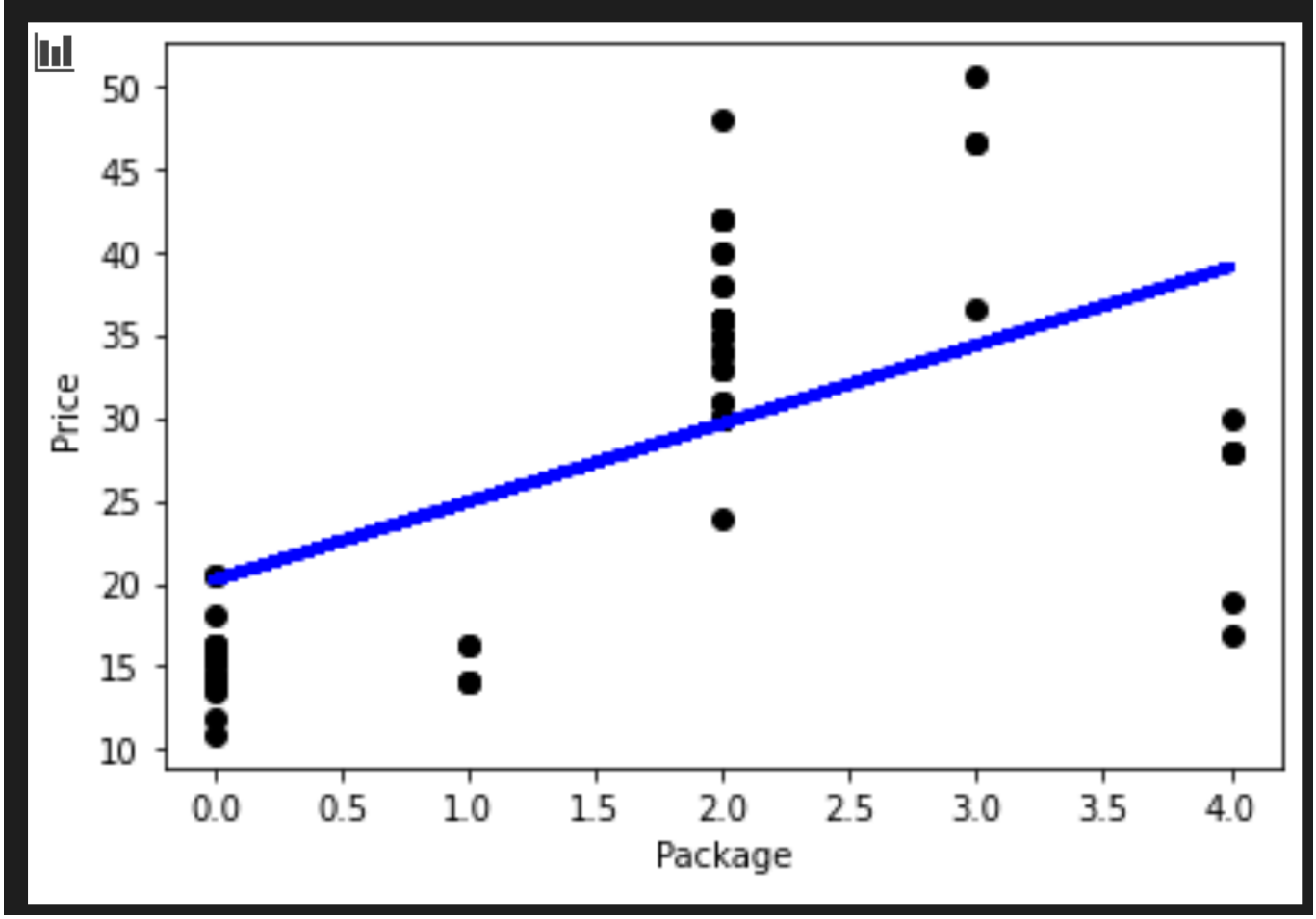
```output
Model Accuracy:  0.3315342327998987
```

3. You can visualize the line that's drawn in the process:

```python
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, pred, color='blue', linewidth=3)

plt.xlabel('Package')
plt.ylabel('Price')

plt.show()
```



4. Test the model against a hypothetical variety:

```python
lin_reg.predict( np.array([ [2.75] ]) )
```

The returned price for this mythological Variety is:

```output
array([[33.15655975]])
```

That number makes sense, if the logic of the regression line holds true.

🎃 Congratulations, you just created a model that can help predict the price of a few varieties of pumpkins. Your holiday pumpkin patch will be beautiful. But you can probably create a better model!

Polynomial regression

Another type of linear regression is polynomial regression. While sometimes there's a linear relationship between variables - the bigger the pumpkin in volume, the higher the price - sometimes these relationships can't be plotted as a plane or straight line.

✅ Here are some more examples of data that could use polynomial regression

Take another look at the relationship between Variety to Price in the previous plot. Does this scatterplot seem like it should necessarily be analyzed by a straight line? Perhaps not. In this case, you can try polynomial regression.

✅ Polynomials are mathematical expressions that might consist of one or more variables and coefficients

Polynomial regression creates a curved line to better fit nonlinear data.

1. Let's recreate a dataframe populated with a segment of the original pumpkin data:

```python
new_columns = ['Variety', 'Package', 'City', 'Month', 'Price']
poly_pumpkins = new_pumpkins.drop([c for c in new_pumpkins.columns if c

poly_pumpkins
```

A good way to visualize the correlations between data in dataframes is to display it in a 'coolwarm' chart:

2. Use the `Background_gradient()` method with `coolwarm` as its argument value:

```python
corr = poly_pumpkins.corr()
corr.style.background_gradient(cmap='coolwarm')
```

This code creates a heatmap:

	Month	Variety	City	Package	Price
Month	1.000000	0.171330	-0.188728	-0.144847	-0.148783
Variety	0.171330	1.000000	-0.248441	-0.614855	-0.863479
City	-0.188728	-0.248441	1.000000	0.301604	0.323640
Package	-0.144847	-0.614855	0.301604	1.000000	0.606171
Price	-0.148783	-0.863479	0.323640	0.606171	1.000000

Looking at this chart, you can visualize the good correlation between Package and Price. So you should be able to create a somewhat better model than the last one.

Create a pipeline

Scikit-learn includes a helpful API for building polynomial regression models - the `make_pipeline` API. A 'pipeline' is created which is a chain of estimators. In this case, the pipeline includes polynomial features, or predictions that form a nonlinear path.

1. Build out the X and y columns:

```python
X=poly_pumpkins.iloc[:,3:4].values
y=poly_pumpkins.iloc[:,4:5].values
```

2. Create the pipeline by calling the `make_pipeline()` method:

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(PolynomialFeatures(4), LinearRegression())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

pipeline.fit(np.array(X_train), y_train)

y_pred=pipeline.predict(X_test)
```

Create a sequence

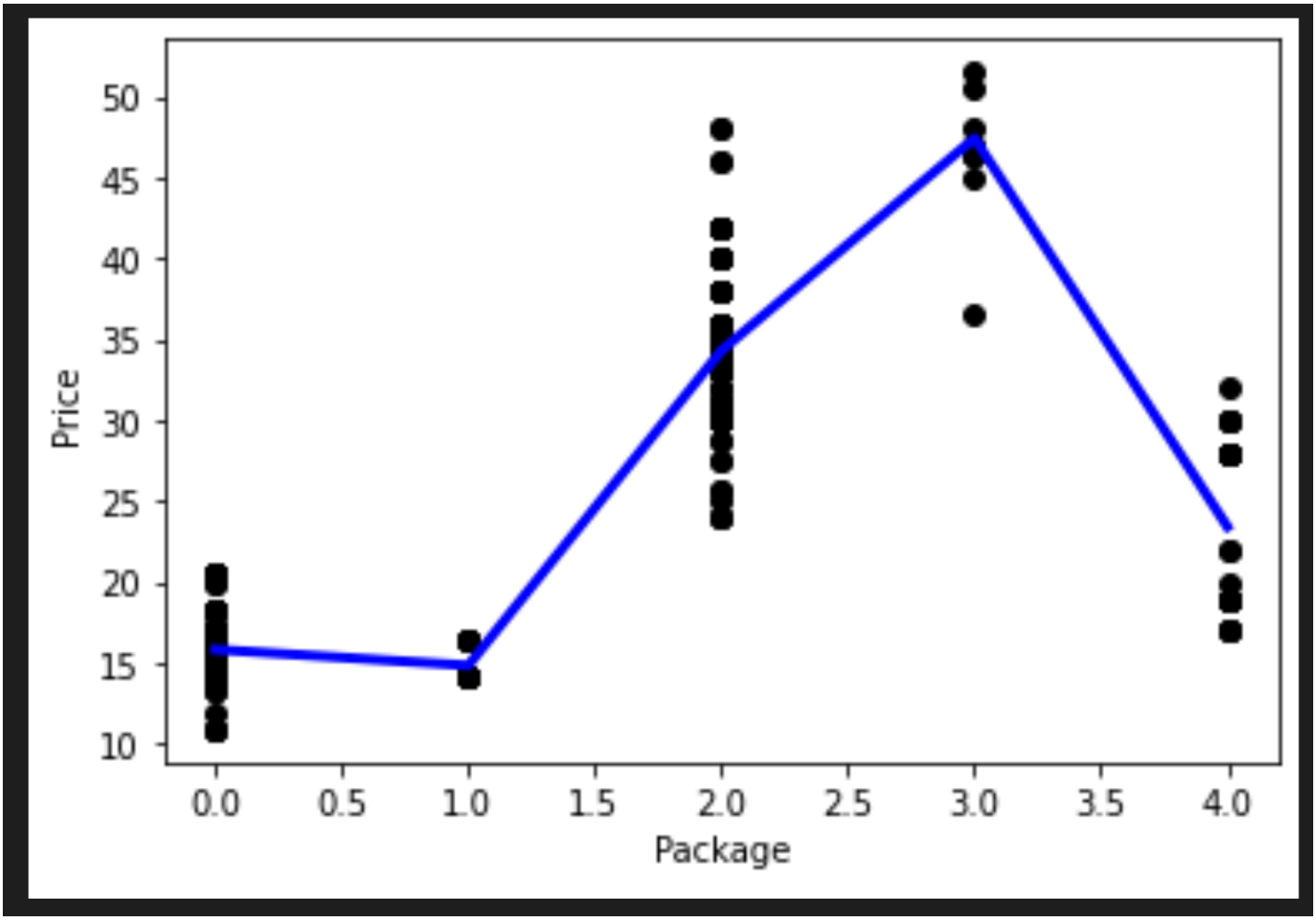
At this point, you need to create a new dataframe with *sorted* data so that the pipeline can create a sequence.

Add the following code:

```python
df = pd.DataFrame({'x': X_test[:,0], 'y': y_pred[:,0]})
df.sort_values(by='x',inplace = True)
points = pd.DataFrame(df).to_numpy()

plt.plot(points[:, 0], points[:, 1],color="blue", linewidth=3)
plt.xlabel('Package')
plt.ylabel('Price')
plt.scatter(X,y, color="black")
plt.show()
```

You created a new dataframe by calling `pd.DataFrame` . Then you sorted the values by calling `sort_values()` . Finally you created a polynomial plot:



You can see a curved line that fits your data better.

Let's check the model's accuracy:

```python
accuracy_score = pipeline.score(X_train,y_train)
print('Model Accuracy: ', accuracy_score)
```

And voila!

```output
Model Accuracy:  0.8537946517073784
```

That's better! Try to predict a price:

Do a prediction

Can we input a new value and get a prediction?

Call `predict()` to make a prediction:

```python
pipeline.predict( np.array([ [2.75] ]) )
```

You are given this prediction:

```output
array([[46.34509342]])
```

It does make sense, given the plot! And, if this is a better model than the previous one, looking at the same data, you need to budget for these more expensive pumpkins!

🏆 Well done! You created two regression models in one lesson. In the final section on regression, you will learn about logistic regression to determine categories.

🚀Challenge

Test several different variables in this notebook to see how correlation corresponds to model accuracy.

Post-lecture quiz

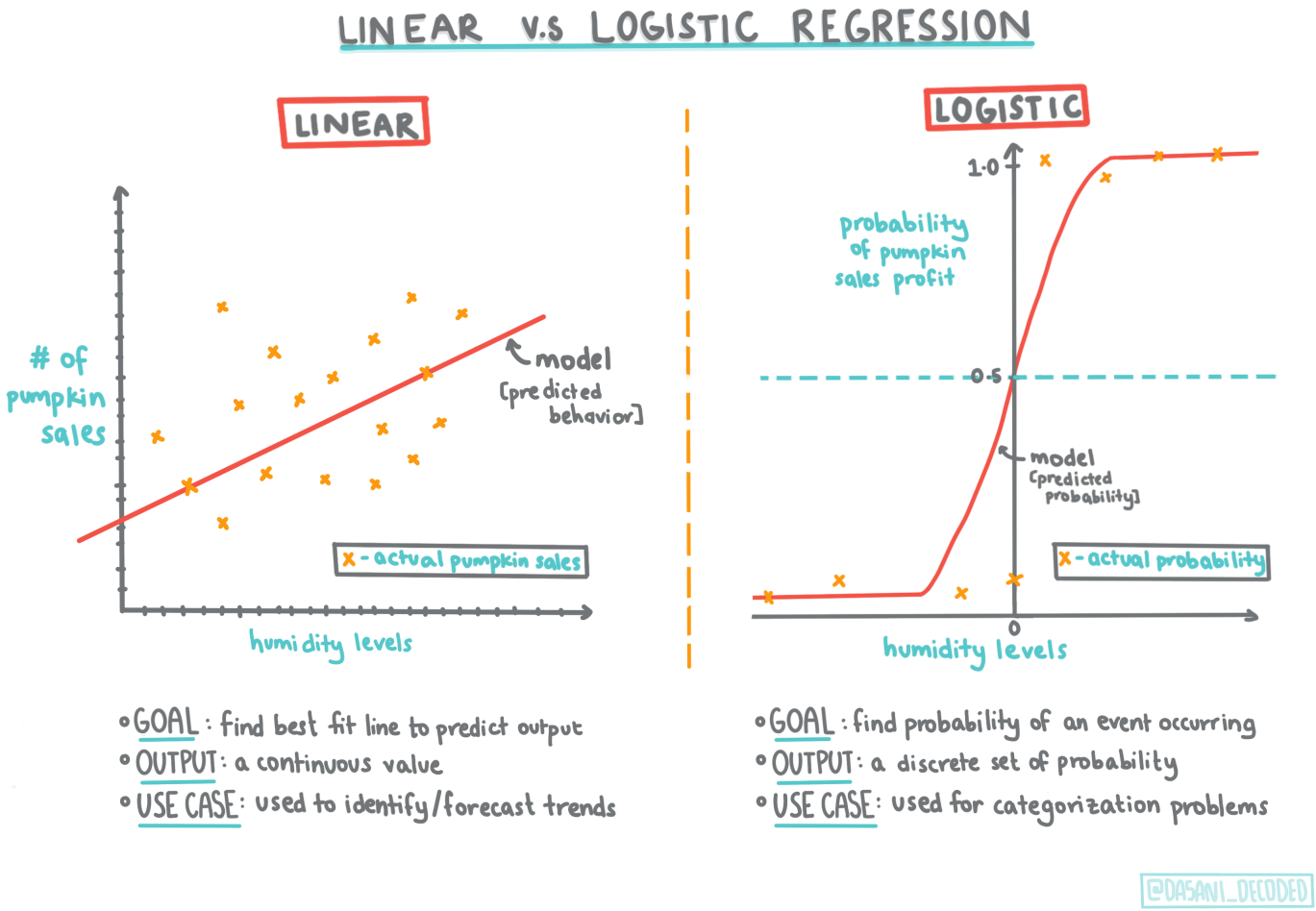
Review & Self Study

In this lesson we learned about Linear Regression. There are other important types of Regression. Read about Stepwise, Ridge, Lasso and Elasticnet techniques. A good course to study to learn more is the Stanford Statistical Learning course

Assignment

Build a Model

Logistic regression to predict categories



> Infographic by Dasani Madipalli

Pre-lecture quiz

Introduction

In this final lesson on Regression, one of the basic *classic* ML techniques, we will take a look at Logistic Regression. You would use this technique to discover patterns to predict binary categories. Is this candy chocolate or not? Is this disease contagious or not? Will this customer choose this product or not?

In this lesson, you will learn:

- A new library for data visualization

- Techniques for logistic regression

✅ Deepen your understanding of working with this type of regression in this Learn module

Prerequisite

Having worked with the pumpkin data, we are now familiar enough with it to realize that there's one binary category that we can work with: `Color` .

Let's build a logistic regression model to predict that, given some variables, *what color a given pumpkin is likely to be* (orange 🎃 or white 👻).

> Why are we talking about binary classification in a lesson grouping about regression? Only for linguistic convenience, as logistic regression is really a classification method, albeit a linear-based one. Learn about other ways to classify data in the next lesson group.

Define the question

For our purposes, we will express this as a binary: 'Orange' or 'Not Orange'. There is also a 'striped' category in our dataset but there are few instances of it, so we will not use it. It disappears once we remove null values from the dataset, anyway.

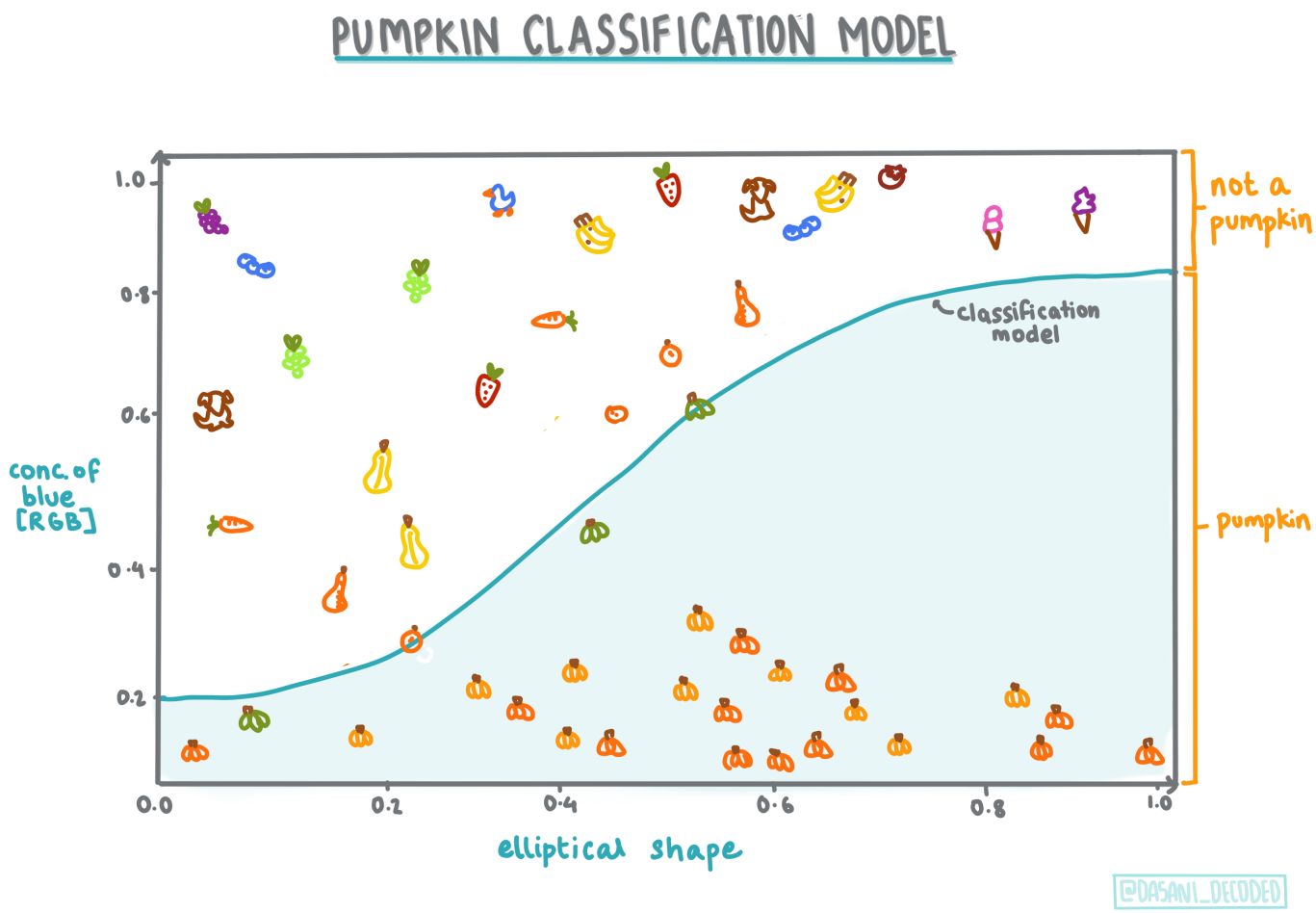
> 🎃 Fun fact, we sometimes call white pumpkins 'ghost' pumpkins. They aren't very easy to carve, so they aren't as popular as the orange ones but they are cool looking!

About logistic regression

Logistic regression differs from linear regression, which you learned about previously, in a few important ways.

Binary classification

Logistic regression does not offer the same features as linear regression. The former offers a prediction about a binary category ("orange or not orange") whereas the latter is capable of predicting continual values, for example given the origin of a pumpkin and the time of harvest, *how much its price will rise*.

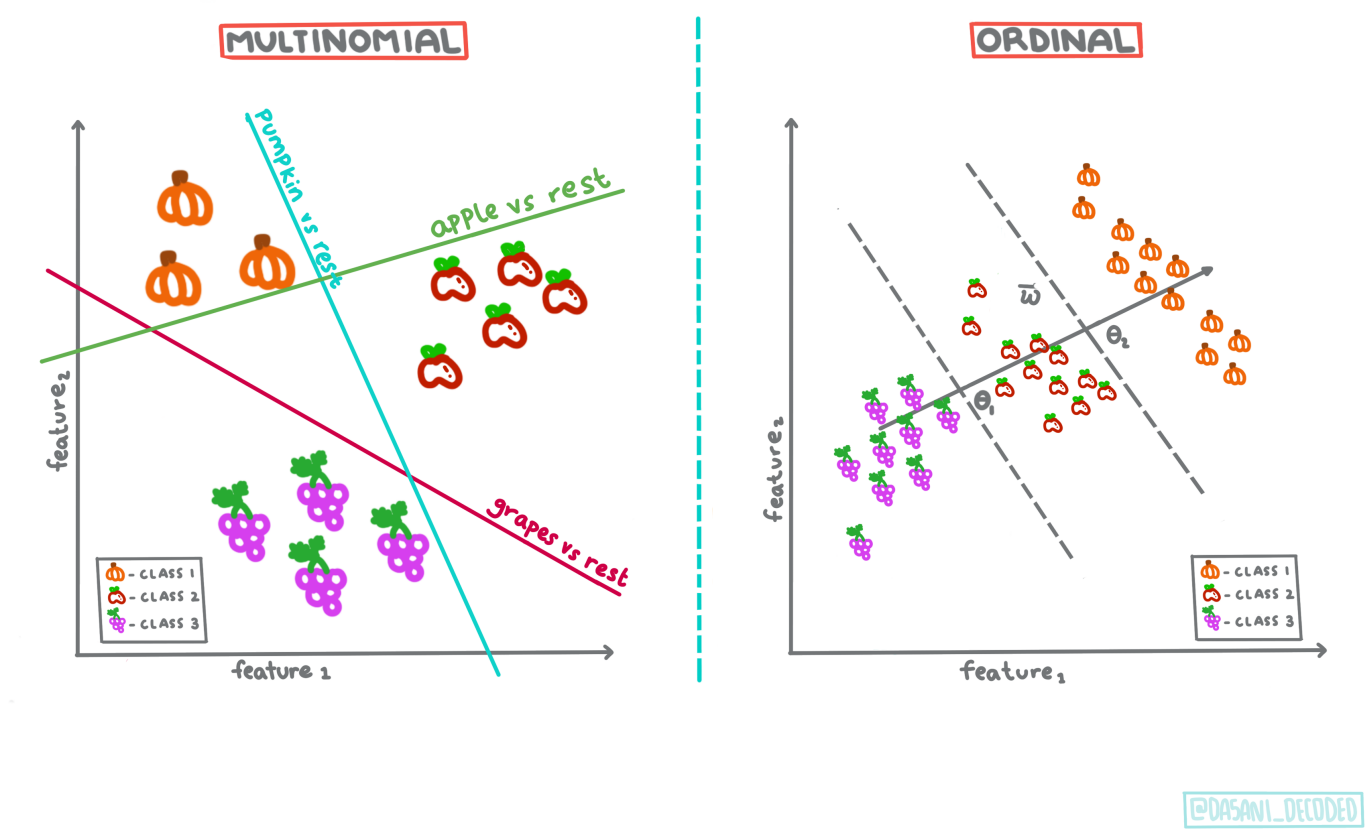


> Infographic by Dasani Madipalli

Other classifications

There are other types of logistic regression, including multinomial and ordinal:

- **Multinomial**, which involves having more than one category - "Orange, White, and Striped".
- **Ordinal**, which involves ordered categories, useful if we wanted to order our outcomes logically, like our pumpkins that are ordered by a finite number of sizes (mini,sm,med,lg,xl,xxl).



> Infographic by Dasani Madipalli

It's still linear

Even though this type of Regression is all about 'category predictions', it still works best when there is a clear linear relationship between the dependent variable (color) and the other independent variables (the rest of the dataset, like city name and size). It's good to get an idea of whether there is any linearity dividing these variables or not.

Variables DO NOT have to correlate

Remember how linear regression worked better with more correlated variables? Logistic regression is the opposite - the variables don't have to align. That works for this data which has somewhat weak correlations.

You need a lot of clean data