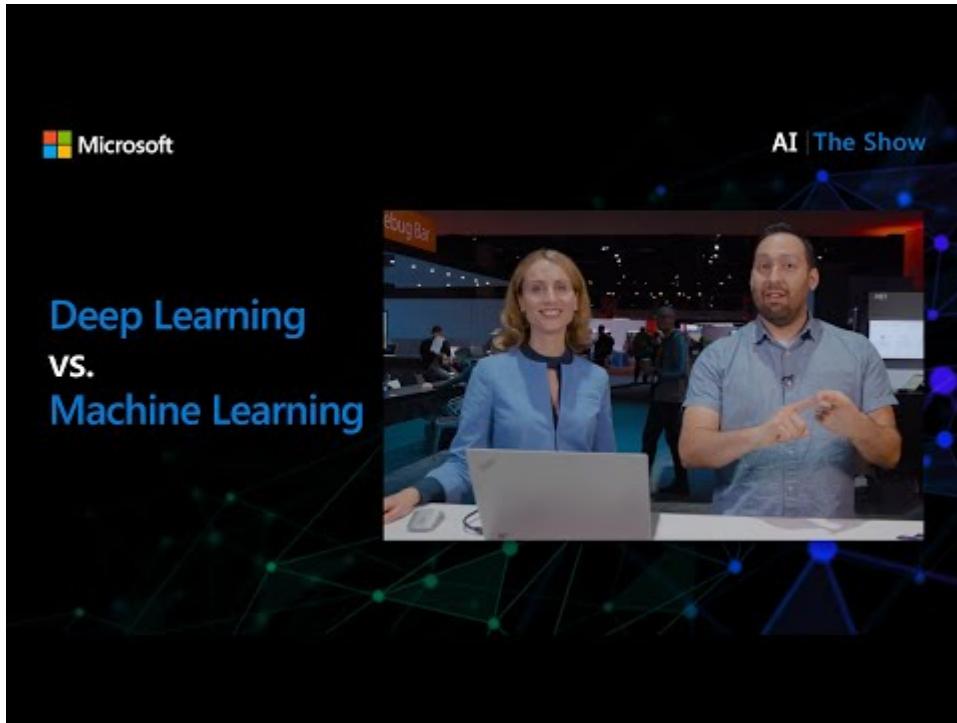


# Introduction to Machine Learning



🎥 Click the image above for a video discussing the difference between Machine Learning, AI, and Deep Learning.

## Pre-lecture quiz

### Introduction

Welcome to this course on classical machine learning for beginners! Whether you're completely new to this topic, or an experienced ML practitioner looking to brush up on an area, we're happy to have you join us! We want to create a friendly launching spot for your ML learning and would be happy to evaluate, respond to, and incorporate your feedback.



🎥 Click the image above for a video: MIT's John Guttag introduces Machine Learning

## Getting started with Machine Learning

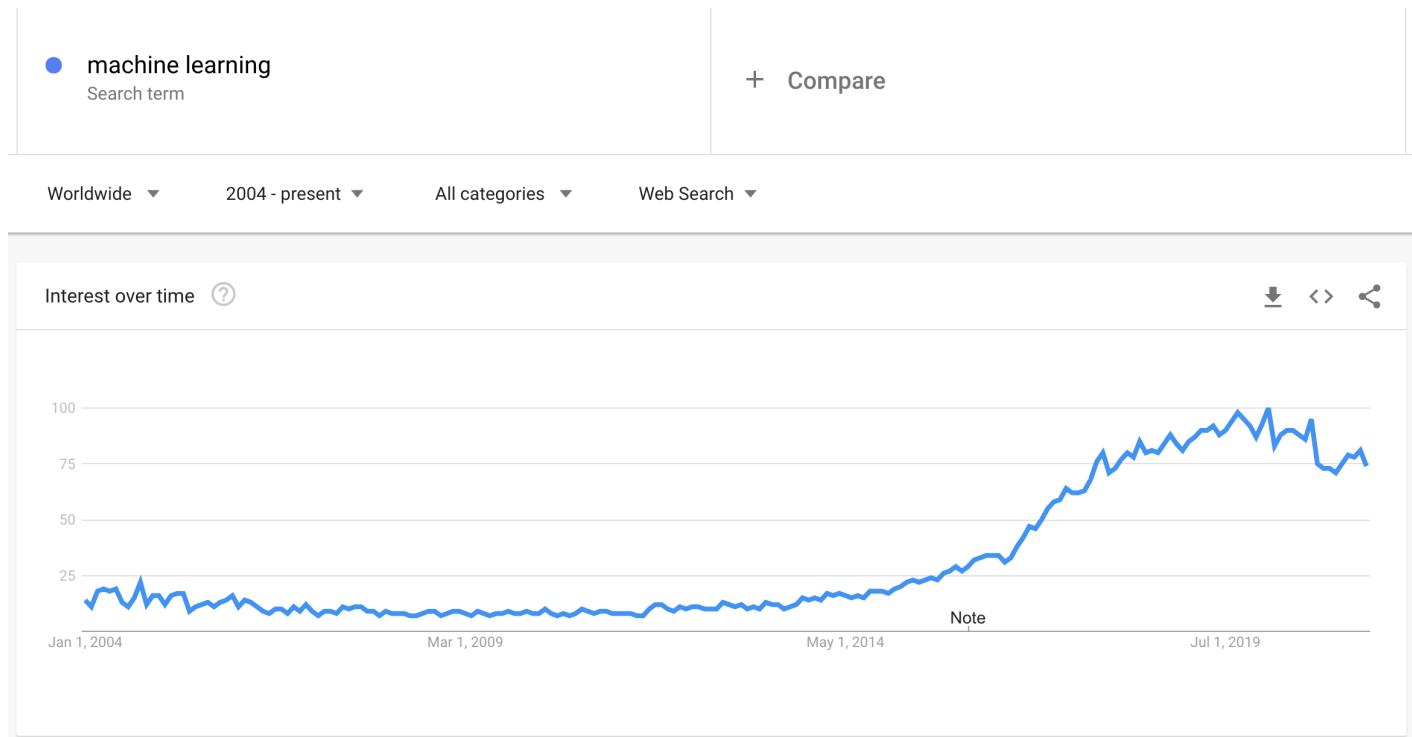
Before starting with this curriculum, you need to have your computer set up and ready to run notebooks locally.

- **Configure your machine with these videos.** Learn more about how to set up your machine in this [set of videos](#).
- **Learn Python.** It's also recommended to have a basic understanding of [Python](#), a programming language useful for data scientists that we use in this course.
- **Learn Node.js and JavaScript.** We also use JavaScript a few times in this course when building web apps, so you will need to have [node](#) and [npm](#) installed, as well as [Visual Studio Code](#) available for both Python and JavaScript development.
- **Create a GH account.** Since you found us here on [GitHub](#), you might already have an account, but if not, create one and then fork this curriculum to use on your own. (Feel free to give us a star, too :))
- **Explore Scikit-Learn.** Familiarize yourself with [Scikit-Learn](#), a set of ML libraries that we reference in these lessons.

## What is Machine Learning?

The term 'Machine Learning' is one of the most popular and frequently used terms of today. There is a nontrivial possibility that you have heard this term at least once if you have some sort of familiarity

with technology, no matter what domain you work in. The mechanics of Machine Learning, however, are a mystery to most people. For a Machine Learning beginner, the subject can sometimes feel overwhelming. Therefore, it is important to understand what Machine Learning actually is, and to learn about it step by step, through practical examples.



Google Trends shows the recent 'hype curve' of the term 'machine learning'

We live in a universe full of unusual and interesting mysteries. Great scientists such as Stephen Hawking, Albert Einstein, and many more have devoted their lives in search of meaningful information that uncovers the mysteries of the world around us. This is the human condition of learning: a human child learns new things and uncovers the structure of their world year by year as they grow to adulthood.

A child's brain and senses perceive the facts of their surroundings and gradually learn the hidden patterns of life which help the child to craft logical rules to identify learned patterns. The learning process of the human brain makes humans the most sophisticated living creature of this world. Learning continuously by discovering hidden patterns and then innovating on those patterns enables us to make ourselves better and better throughout our lifetime. This learning capacity and evolving capability is related to a concept called brain plasticity. Superficially, we can draw some motivational similarities between the learning process of the human brain and the concepts of machine learning.

The human brain perceives things from the real world, processes the perceived information, makes rational decisions, and performs certain actions based on circumstances. This is what we called behaving intelligently. When we program a facsimile of the intelligent behavioral process to a machine, it is called Artificial Intelligence (AI). Although the terms can be confused, Machine Learning

(ML) is an important subset of Artificial Intelligence. ML is concerned with using specialized algorithms fetching meaningful information and finding hidden patterns from perceived data to corroborate the rational decision-making process.

## What you will learn in this course

---

In this curriculum, we are going to cover only the core concepts of Machine Learning that a beginner must know. We cover what we call 'Classical Machine Learning'. To understand broader concepts of Artificial Intelligence or Deep Learning, a strong fundamental knowledge of Machine Learning is indispensable, and so we would like to offer it here. You will additionally learn the basics of Regression, Classification, Clustering, Natural Language Processing, Time Series, and Reinforcement Learning, as well as real-world applications, the history of ML, ML and Fairness, and how to use your model in a web app.

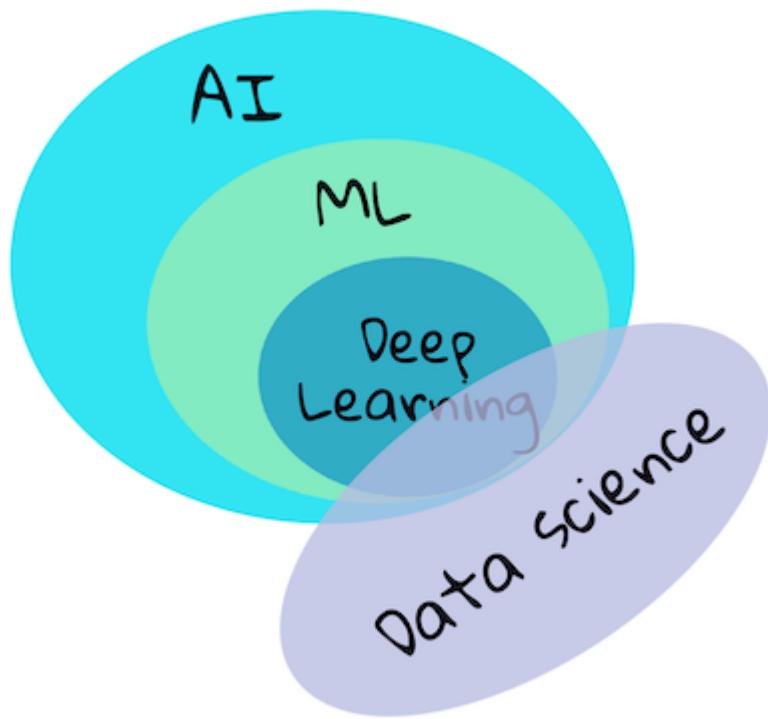
In this course you will learn:

- Core concepts of Machine Learning
- The definition of "Classical Machine Learning"
- Regression
- Classification
- Clustering
- Natural Language Processing
- Time series
- Reinforcement learning
- Real world applications
- History of ML and ML and fairness

## We will not cover

To make for a better learning experience, we will avoid the complexities of neural networks, 'Deep Learning' - many-layered model-building - and AI, which we will discuss in a different curriculum.

- Deep Learning
- AI



A diagram showing the relationships between AI, ML, Deep Learning, and Data Science.  
Infographic by [Jen Looper](#) inspired by [this graphic](#)

## Why learn Machine Learning

---

Machine Learning is defined as the creation of automated systems that can learn hidden patterns from data to infer intelligent decisions.

The major motivation behind leveraging Machine Learning is to create automated systems that can learn hidden patterns from data to infer intelligent decisions. This motivation seems to be loosely inspired by how the human brain learns certain things based on the data it perceives from the outside world.

- Think for a minute why a business would want to try to use Machine Learning strategies vs. creating a hard-coded rules-based engine.

## Applications of Machine Learning

Applications of Machine Learning are now almost everywhere, and are as ubiquitous as the data that is flowing around our societies, generated by our smart phones, connected devices, and other systems. Considering the immense potential of state-of-the-art Machine Learning algorithms,

researchers have been exploring their capability to solve multi-dimensional and multi-disciplinary real-life problems with great positive outcomes.

You can use Machine Learning in many ways:

- Predict the likelihood of disease from a patient's medical history or reports.
- Leverage weather data to predict weather events.
- Understand the sentiment of a text.
- Detect fake news to stop the spread of propaganda.

Finance, economics, earth science, space exploration, biomedical engineering, cognitive science, and even fields in the humanities have adapted Machine Learning to solve the arduous, data-processing heavy problems of their domain.

Machine Learning automates the process of pattern-discovery by finding meaningful insights from real-world or generated data. It has proven itself to be highly valuable in business, health, and financial applications, among others.

In the near future, understanding the basics of Machine Learning is going to be a must for people from any domain due to its widespread adoption.

## Challenge

---

Sketch, on paper or using an online app like [Excalidraw](#), your understanding of the differences between AI, ML, Deep Learning, and Data Science. Add some ideas of problems that each of these techniques are good at solving.

## Post-lecture quiz

---

## Review & Self Study

---

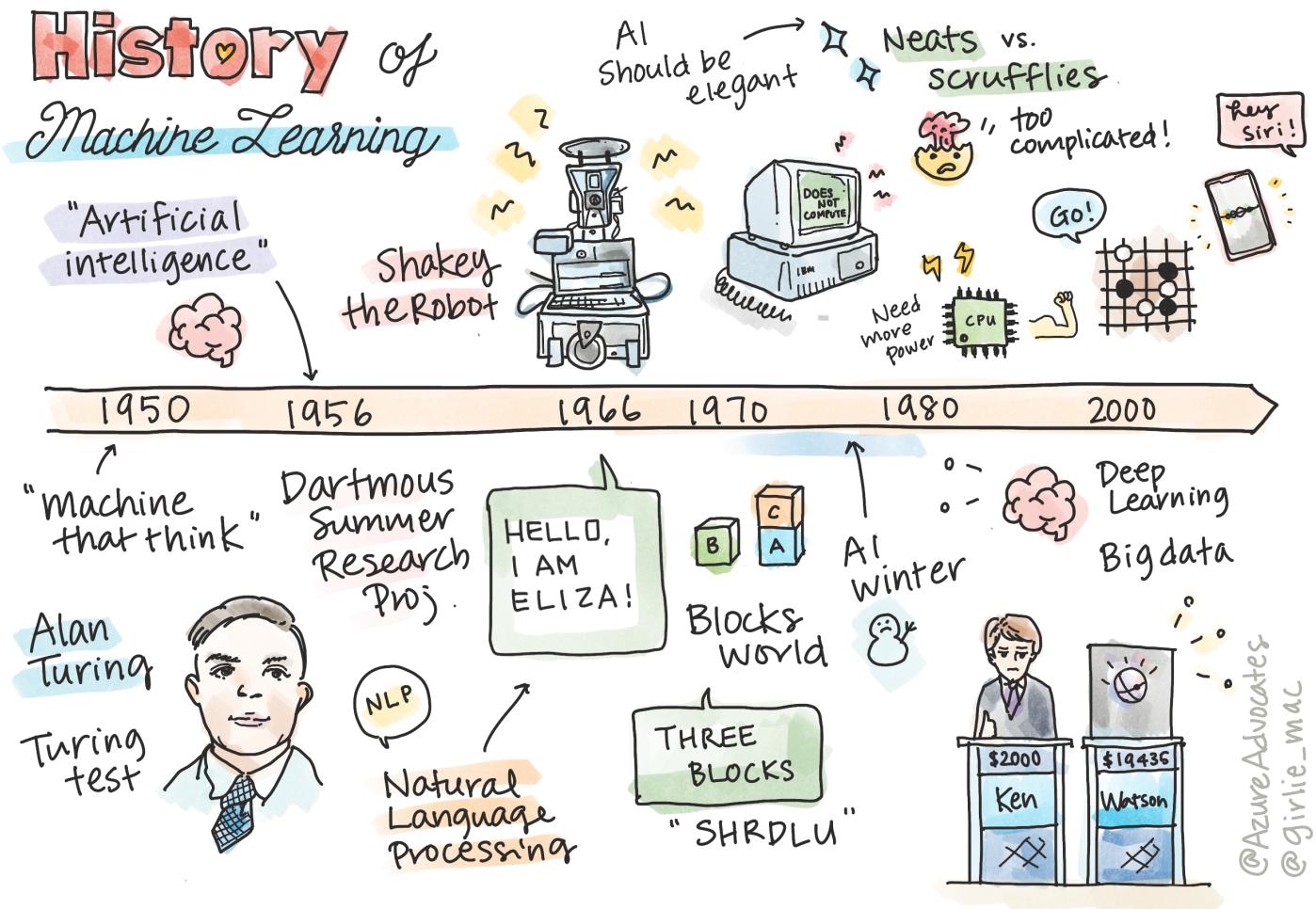
To learn more about how you can work with ML algorithms in the cloud, follow this [Learning Path](#).

## Assignment

---

[Get up and running](#)

# 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 predicated 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"

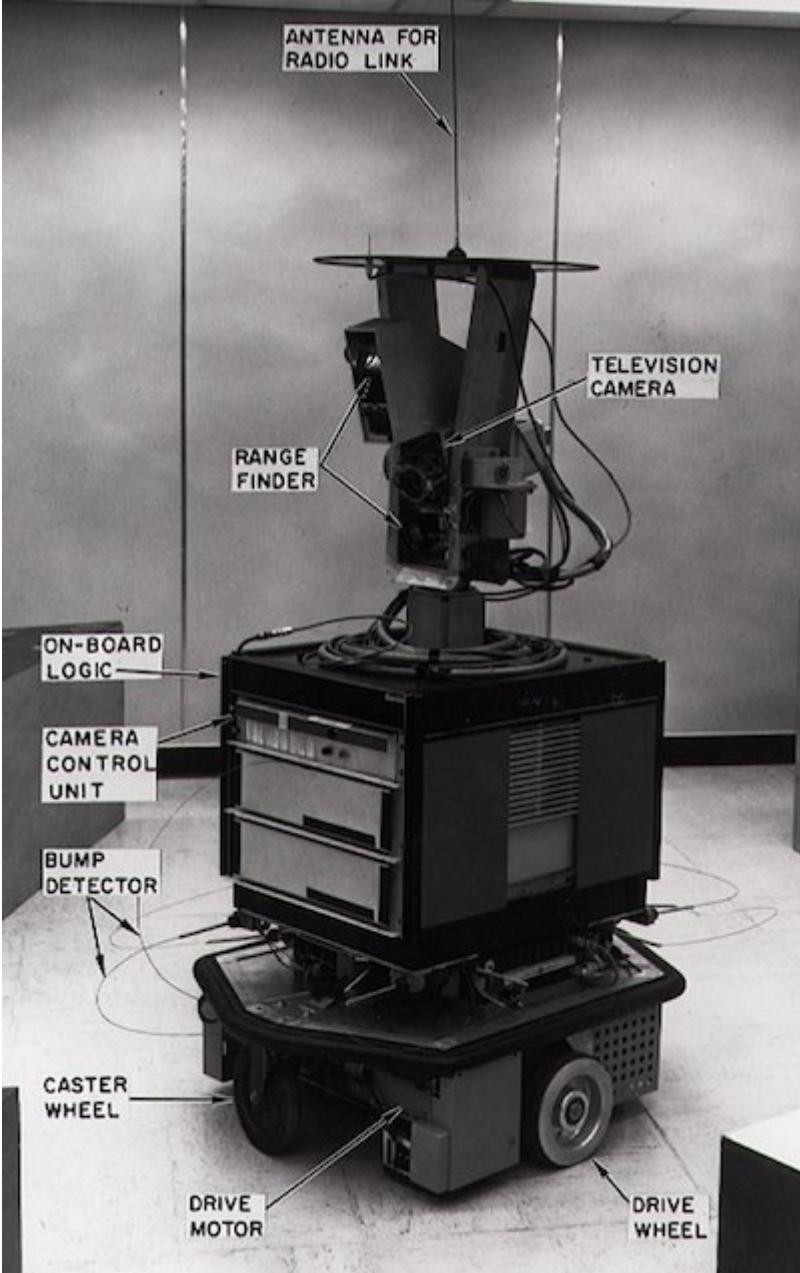
---

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	AAAAAA
EE	LL	II	ZZ	AA AA
EEEEEE	LL	II	ZZZ	AAAAAAA
EE	LL	II	ZZ	AA AA
EEEEEE	LLLLL	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.

markup

[![blocks world with SHRDLU](<https://img.youtube.com/vi/QAJz4YKUwqw/0.jpg>)]

🎥 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

---

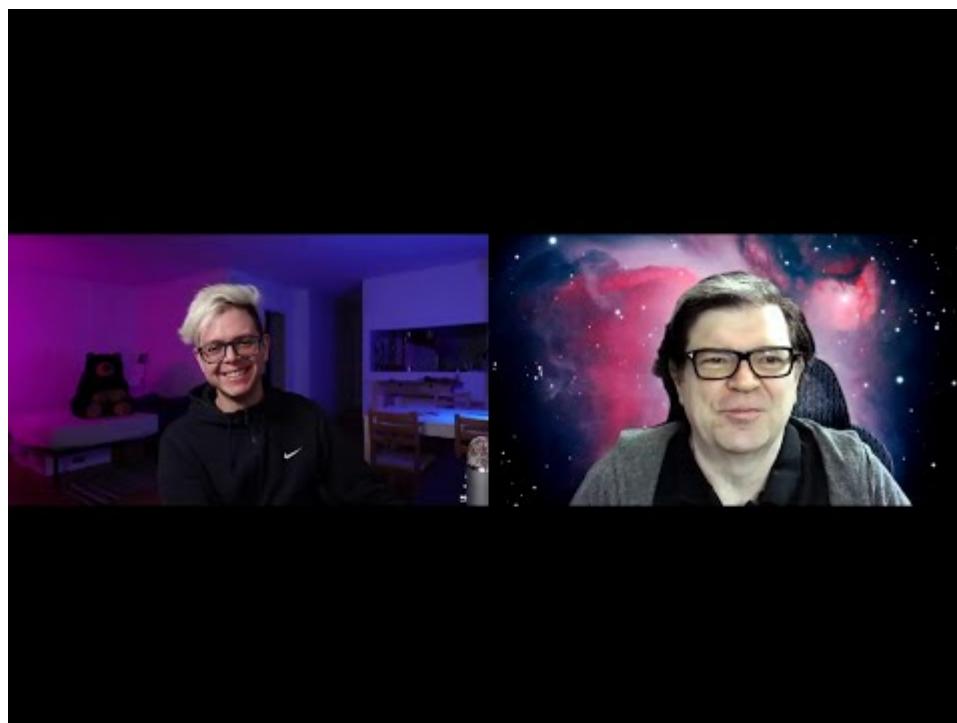
This epoch saw a new era for ML and AI to be able to solve some of the problems that had been caused earlier by the lack of data and compute power. The amount of data began to rapidly increase and become more widely available, for better and for worse, especially with the advent of the smartphone around 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

---

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](#)



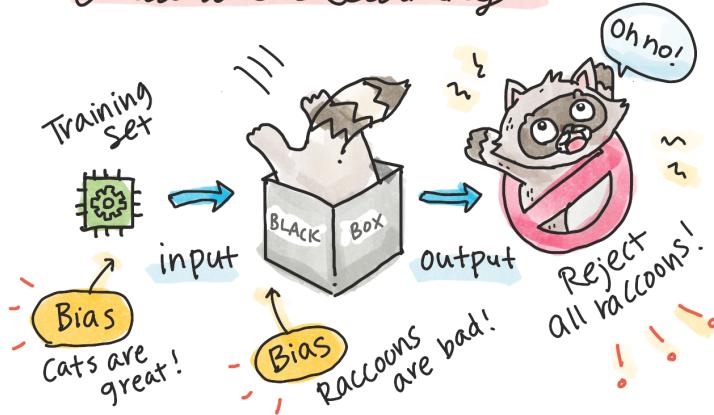
## Assignment

---

[Create a timeline](#)

## Fairness in Machine Learning

# Fairness in Machine Learning



## Complex sociotechnical challenges

purely social      purely technical

@Azure Advocates  
@girlie\_mac

## Fairness-related harms

Unfairness = negative impacts for group of people such as those defined in terms of

- race
- gender
- age
- disability status

### Harms:

- ★ Allocation
- ★ Quality of service
- ★ Stereotyping
- ★ Denigration
- ★ over-/under-representation



## Assessment & mitigation

- ♡ Identify the harm (+benefits)
- ♡ Identify the affected groups
- ♡ Define fairness metrics

	False-	False+	Counts
men	0.35	0.21	6239
women	0.79	0.35	3124



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 disproportionately 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:

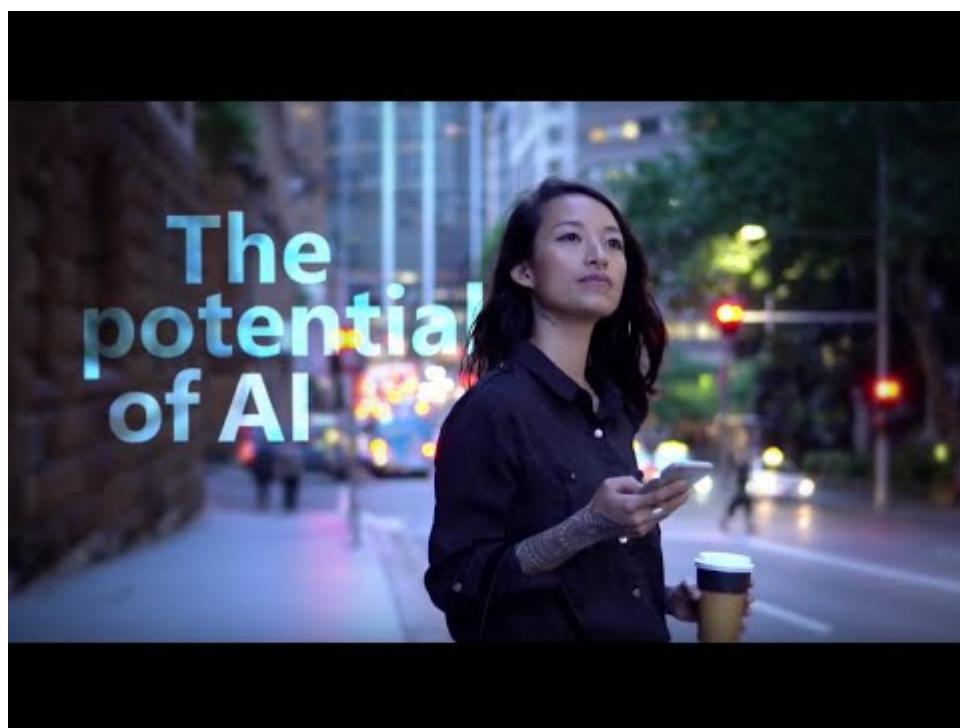
- Raise your awareness of the importance of fairness in machine learning.
- Learn about fairness-related harms.
- Learn about unfairness assessment and mitigation.

## 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".

This screenshot shows a machine translation interface. At the top, there are two dropdown menus: "English" on the left and "Turkish" on the right. Below these, the English phrase "He's a nurse.  
She's a doctor." is displayed on the left, and its Turkish translation "O bir hemşire.  
O bir doktor." is displayed on the right. A circular icon with a double-headed arrow is positioned between the two phrases. Below the phrases are three small icons: a speaker for audio, a microphone for voice input, and a keyboard. At the bottom of the interface, the text "Widely used phrases" is visible.

This screenshot shows a machine translation interface. At the top, there are two dropdown menus: "Turkish" on the left and "English" on the right. Below these, the Turkish phrase "O bir hemşire.  
O bir doktor." is displayed on the left, and its English translation "She's a nurse.  
He's a doctor." is displayed on the right. A circular icon with a double-headed arrow is positioned between the two phrases. Below the phrases are three small icons: a speaker for audio, a microphone for voice input, and a keyboard. At the bottom of the interface, the text "Widely used phrases" is visible.

## 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 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 assess 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](#)

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 process very different from many other development workflows. For web developers, techniques of machine learning can initially seem very strange. In this lesson, we will demystify the process by outlining it. 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 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 input 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, but representing the data well.

- **Training**, this part of the dataset goes into your model to train it. The size of this chunk constitutes the majority of the original dataset.
- **Testing**. A test dataset is another 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 noted 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, you 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, selecting the most appropriate training method for the task at hand.

### Train

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'.

## 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 [here](#).

## 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

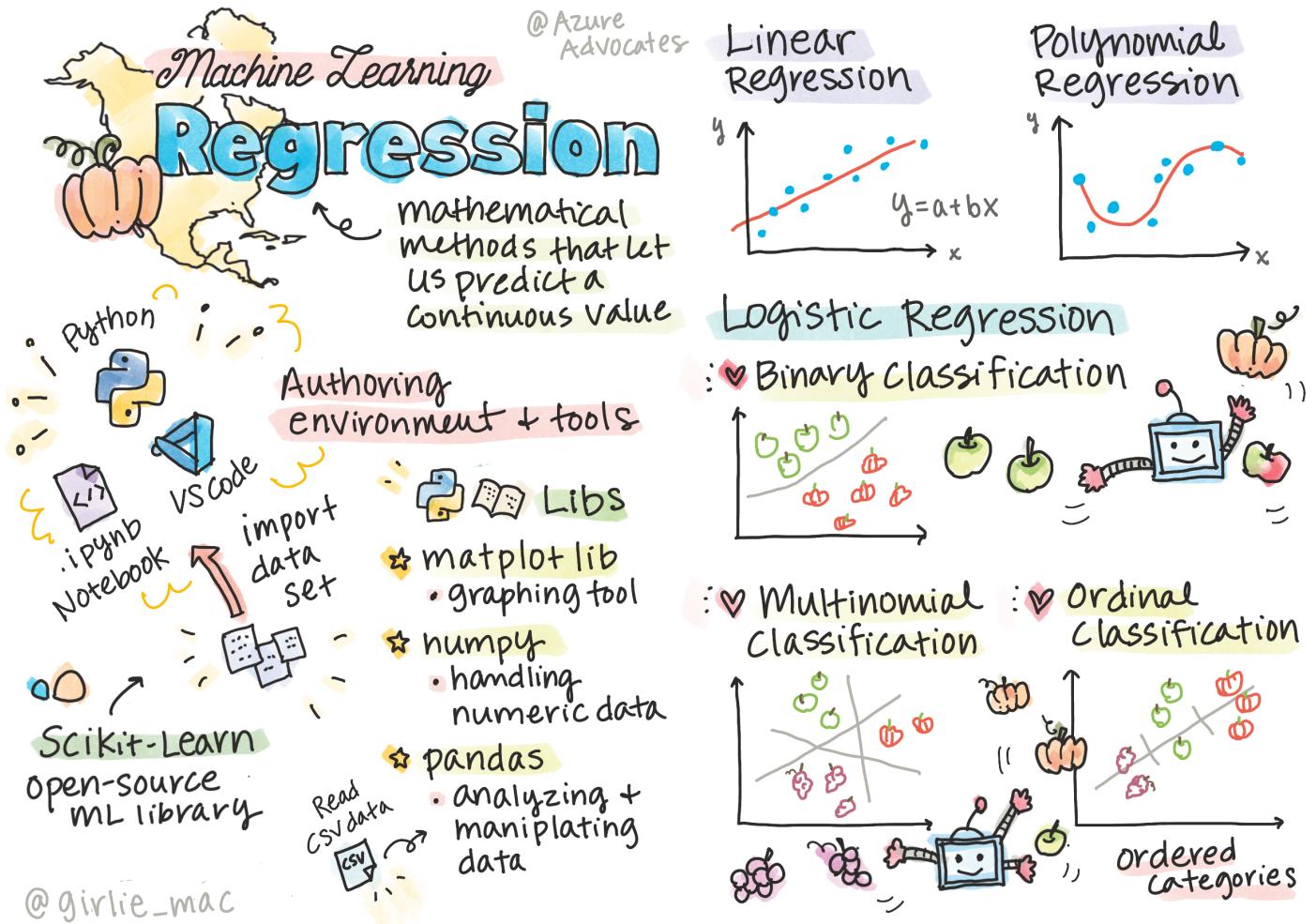
## Review & Self Study

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



## 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.

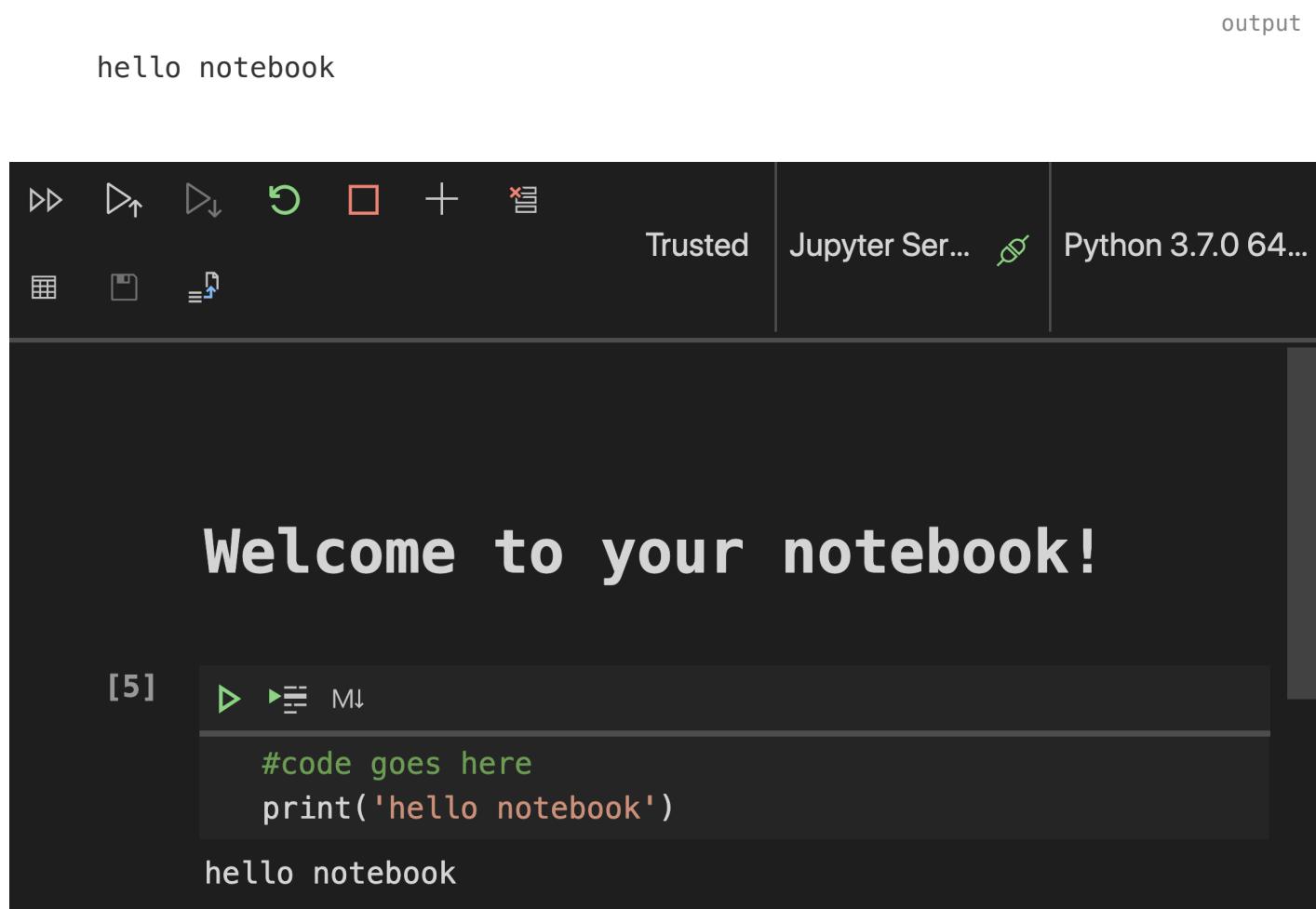
## 2. Select the `md` icon and add a bit of markdown, and the following text `# Welcome to your notebook.`

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:



The screenshot shows a Jupyter Notebook interface in Visual Studio Code. At the top, there's a toolbar with various icons for navigation and cell operations. To the right of the toolbar, it says "Trusted" and "Jupyter Ser...". Below the toolbar, the text "Welcome to your notebook!" is displayed in large, bold, white font. Underneath that, a code cell is shown with the number "[5]" and a play icon. The cell contains the Python code: `#code goes here` and `print('hello notebook')`. The output of the cell, "hello notebook", is displayed below the code. The background of the interface is dark.

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 `matplotlib` , `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 housing dataset

The built-in diabetes housing 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

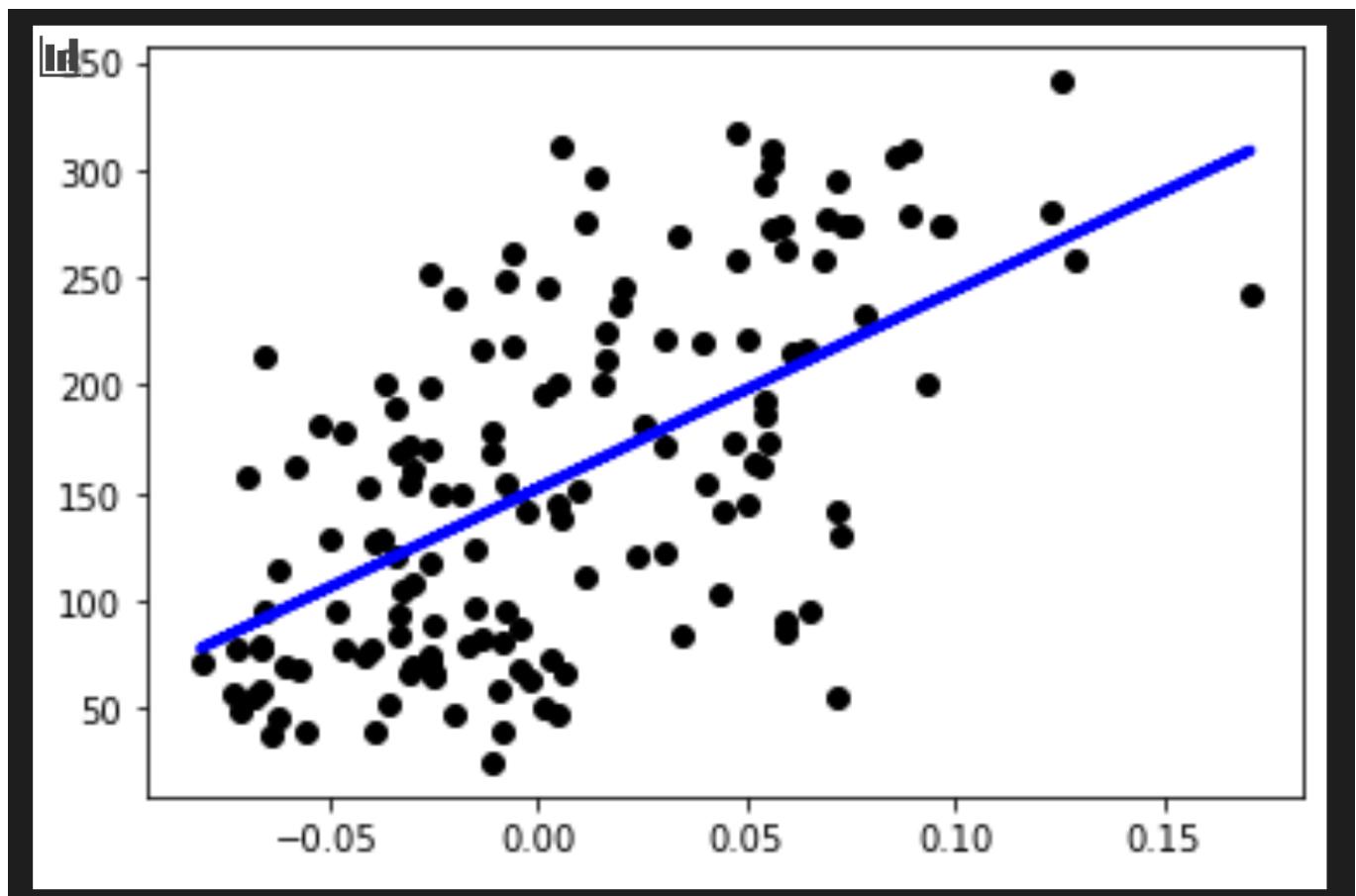
python

```
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!



---

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.

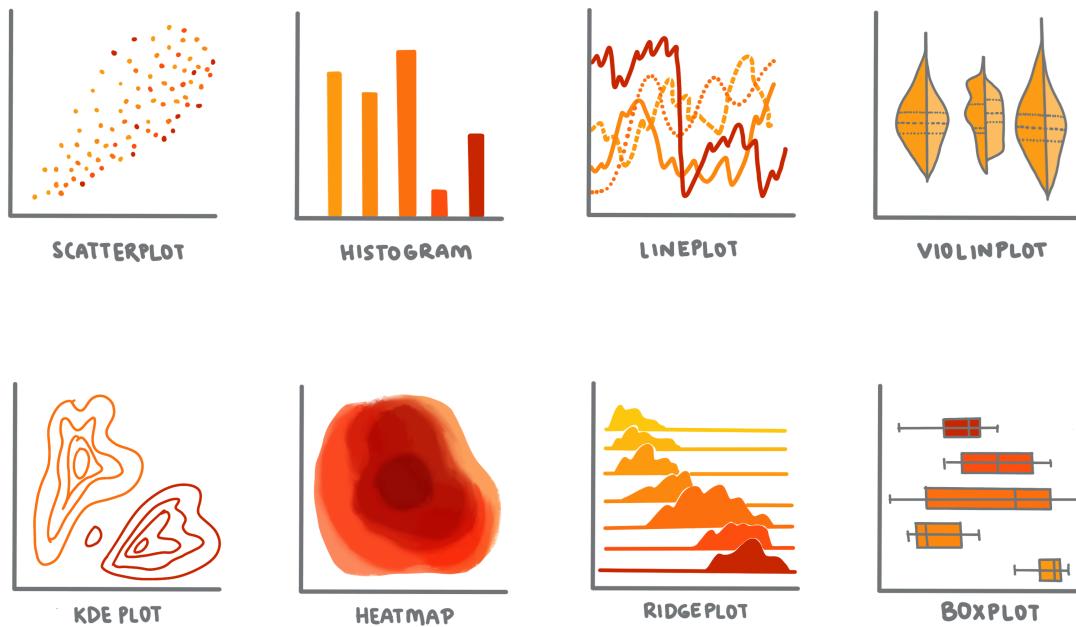
## Assignment

---

[A different dataset](#)

## Build a Regression Model using Scikit-Learn: Prepare and Visualize Data

# DATA VISUALIZATION



How you visualize data influences how you frame the problem

@DASANI\_DECODED

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:

- Preparing your data for model-building.
- Using Matplotlib for data visualization.

## Asking the Right Question

The question you need answered, will determine what type of ML algorithms you will leverage.

Consider the following two examples:

- **Cars and trucks.** For example, do you need to determine the differences between cars and trucks as they cruise down a highway via a video feed? You will need some kind of highly performant classification model to make that differentiation. It will need to be able to perform object detection, probably by showing bounding boxes around detected cars and trucks.
- **Age to height.** What if you are trying to correlate two points of data - like age to height? You can use a linear regression model, as shown in the previous lesson, to draw the classical straight line through the scatterplot of points to show how, with age, height tends to increase. Thus you can predict, for a given group of people, their height given their age.

But it's not very common to be gifted a dataset that is completely ready to use to create a ML model. 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 pumpkin market, 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 *prepared* the data. Next, let's take a closer look at the data.

### The Pumpkin data - early conclusions

What do you notice about this data? First, you see that it is a mix of text and numeric data. There are also dates. Second, you see that there's a considerable amount of missing and mixed data. To build a good model, you will need to handle that.

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

At a high level, you will need to take the following steps to check for missing dates.

1. Convert the dates to a month format (these are US dates, so the format is currently `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: `print(month)` for example.

2. Now, append 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 out 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 library 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

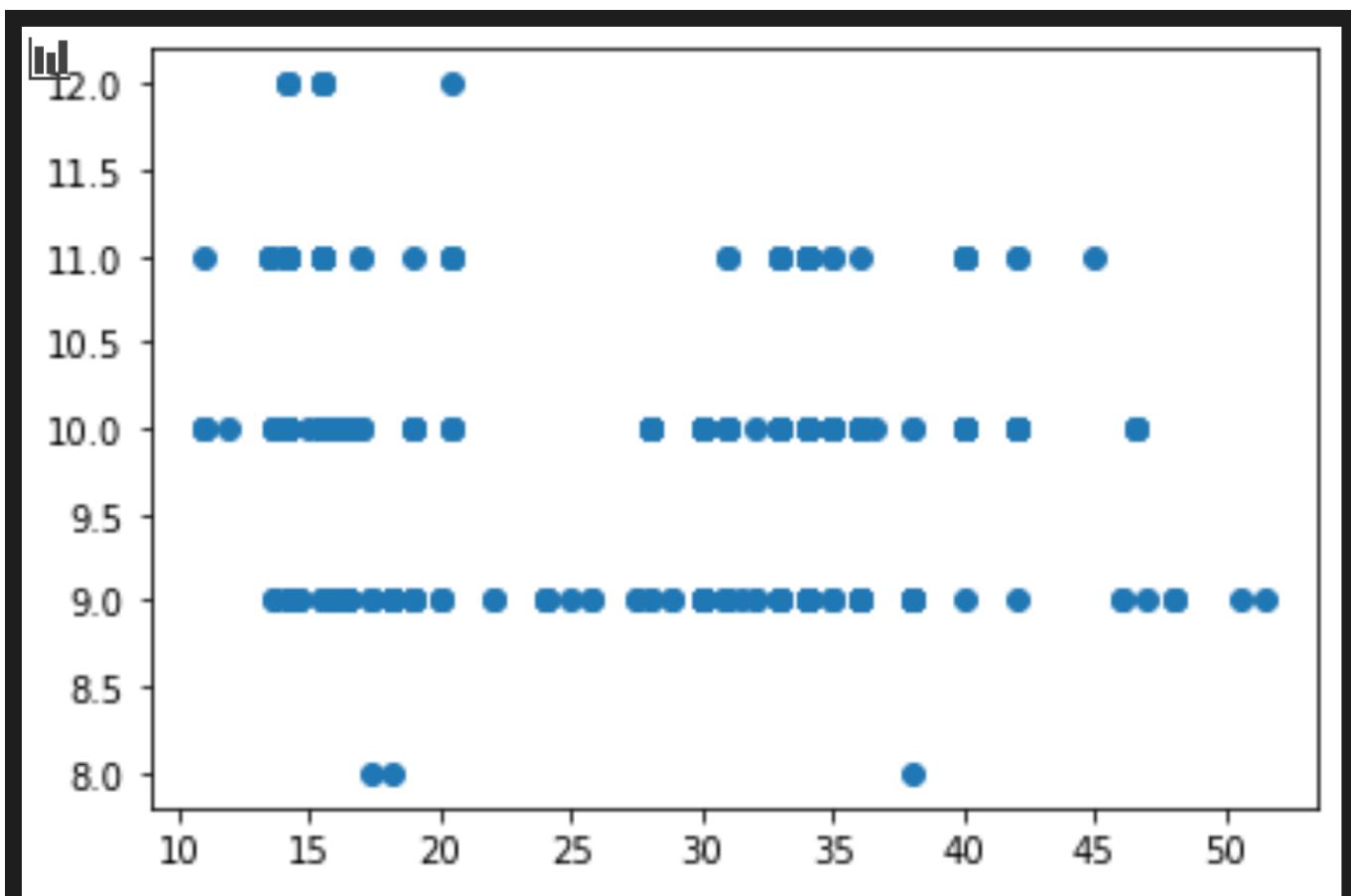
```
import matplotlib.pyplot as plt
```

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.

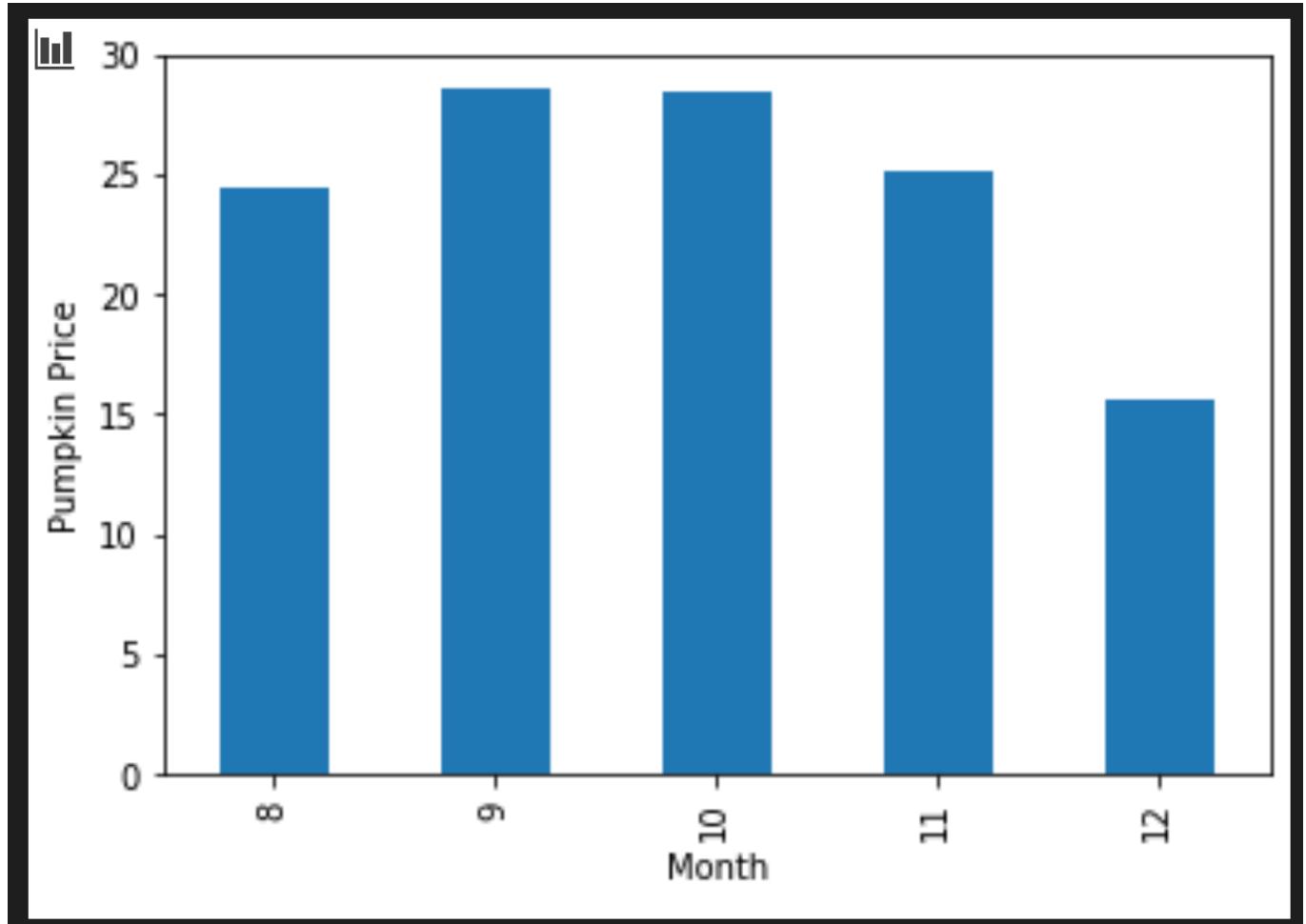
## Make it useful

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?

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

## Post-lecture quiz

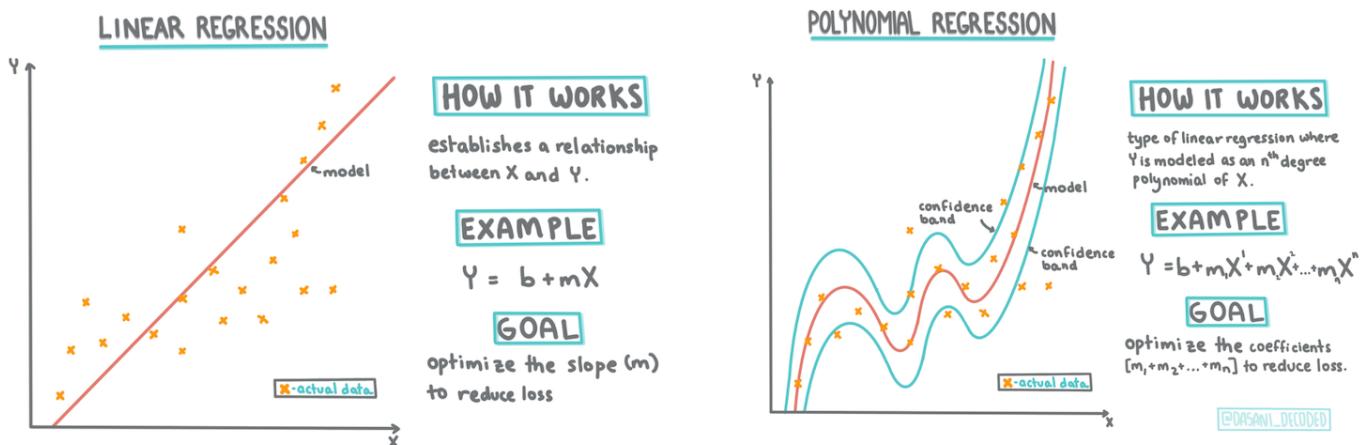
### Review & Self Study

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 unit. 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 files, with the pumpkin price displayed per bushel in a new dataframe. Make sure you can run these notebooks in kernels in VS 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 the relationship between variables and make accurate predictions on where a new datapoint would fall in relationship to that line.

### Show me the math

This line has an equation:  $Y = a + bX$ . It is typical of **Least-Squares Regression** to draw this type of line.

$X$  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  $X = 0$ .

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. 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.

A common method of regression is **Least-Squares Regression** which 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.

One more term to understand is the **Correlation Coefficient** between given X and Y variables. For 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?

## Create a Linear Regression Model correlating Pumpkin Datapoints

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  
in new_columns])
```

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`.

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_error  
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, random_state=42)  
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.

Model Accuracy: 0.3315342327998987

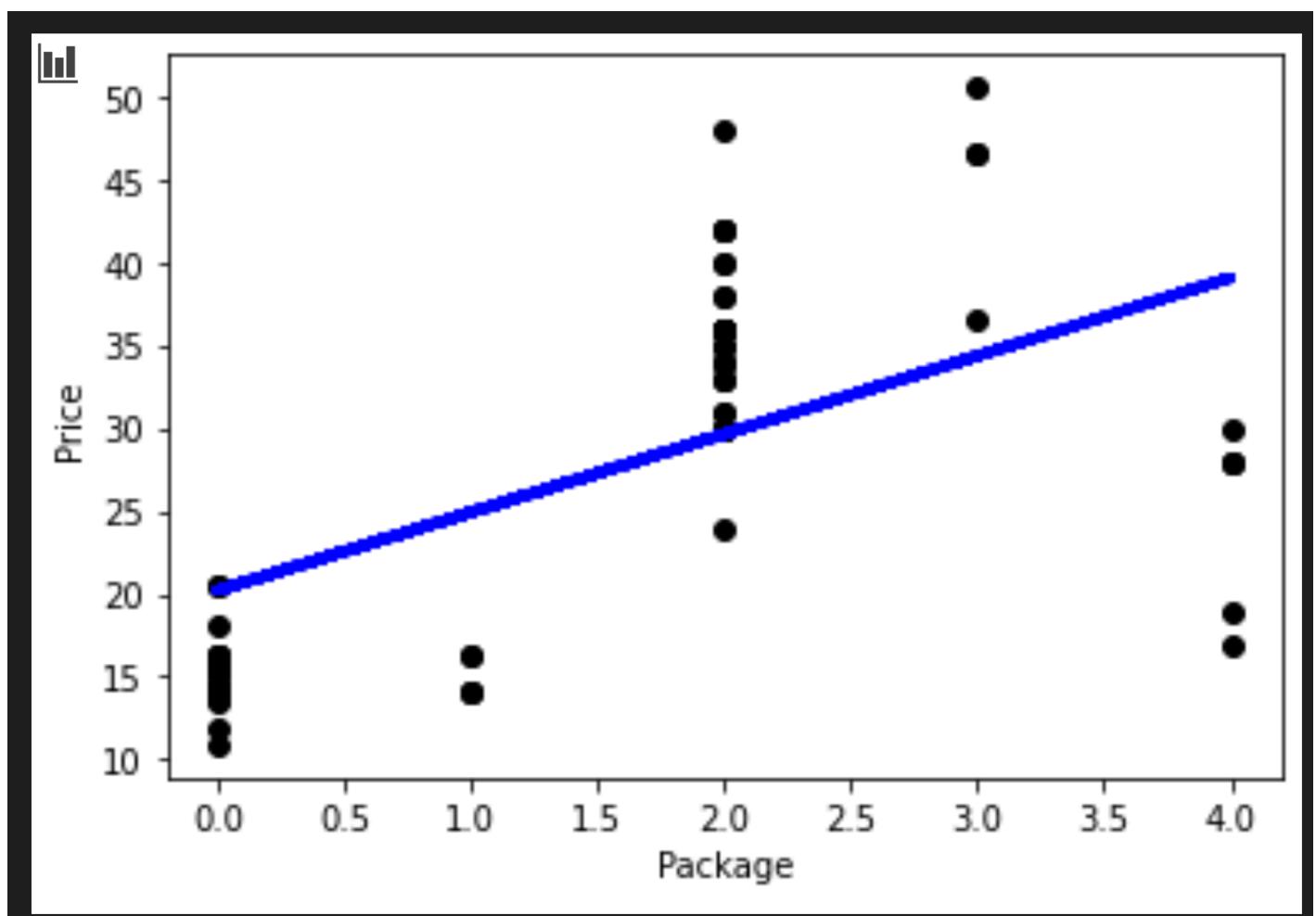
You can visualize the line that's drawn in the process:

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()
```



And you can test the model against a hypothetical variety:

python

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

The returned price for this mythological Variety is:

```
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. 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 no-
poly_pumpkins
```

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

python

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

	<b>Month</b>	<b>Variety</b>	<b>City</b>	<b>Package</b>	<b>Price</b>
<b>Month</b>	1.000000	0.171330	-0.188728	-0.144847	-0.148783
<b>Variety</b>	0.171330	1.000000	-0.248441	-0.614855	-0.863479
<b>City</b>	-0.188728	-0.248441	1.000000	0.301604	0.323640
<b>Package</b>	-0.144847	-0.614855	0.301604	1.000000	0.606171
<b>Price</b>	-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.

Build out the X and y columns:

python

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

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.

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, random_state=42)

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

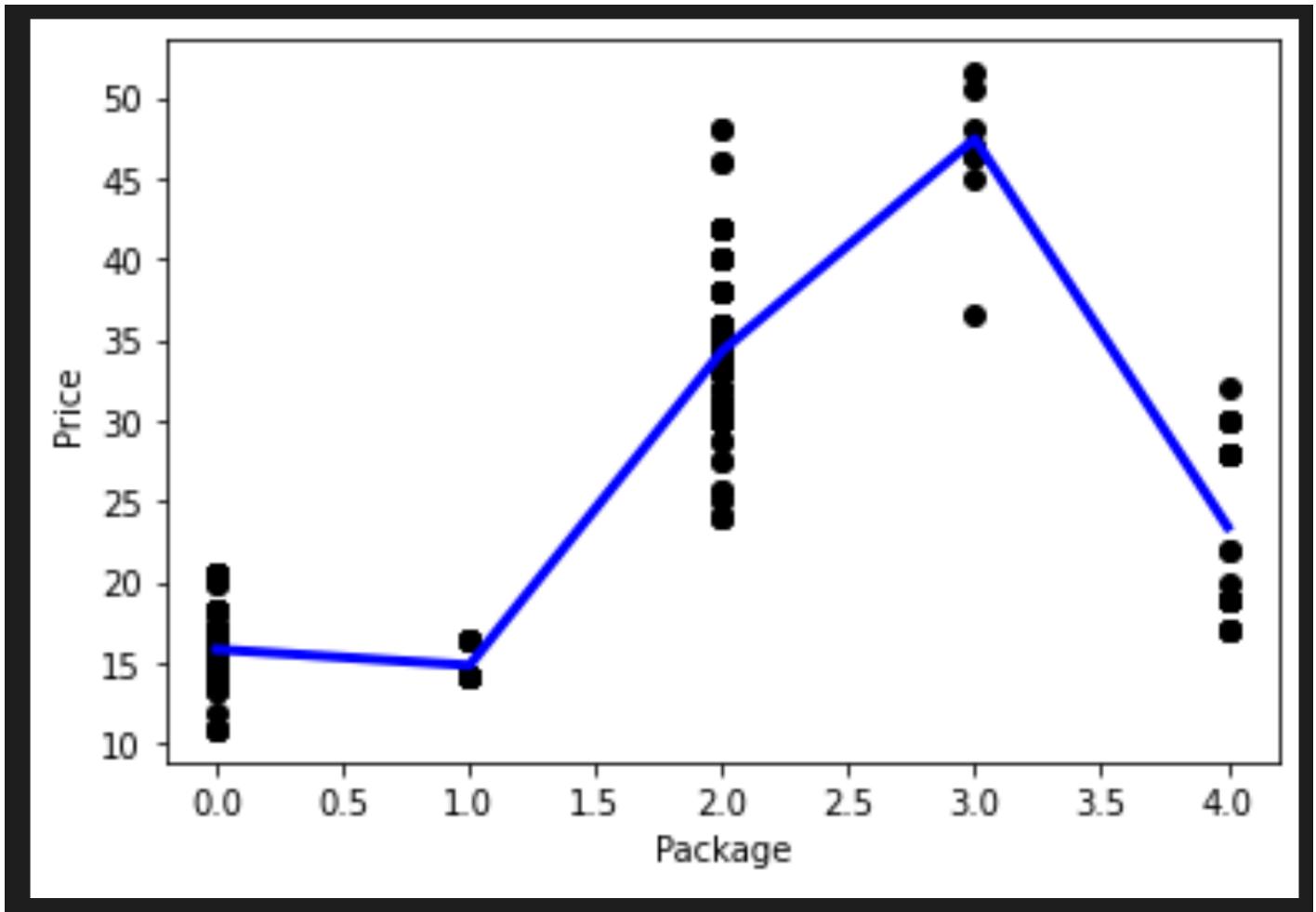
y_pred=pipeline.predict(X_test)
```

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

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 can see a curved line that fits your data better. Let's check the model's accuracy:

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

python

And voila!

Model Accuracy: 0.8537946517073784

That's better! Try to predict a price:

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

python

You are given this prediction:

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

It does make sense! 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](#)

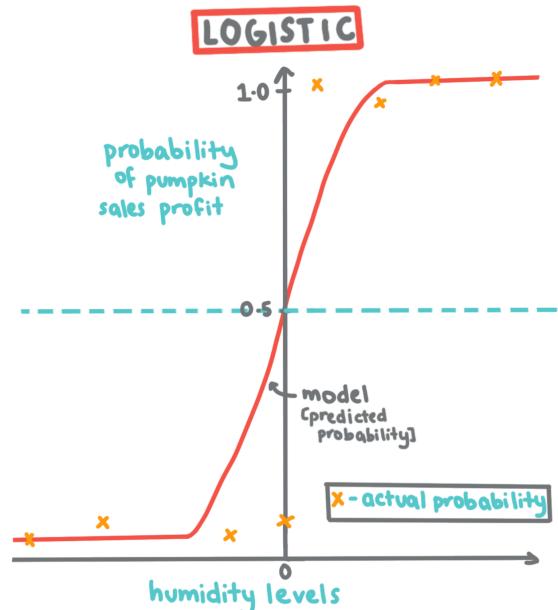
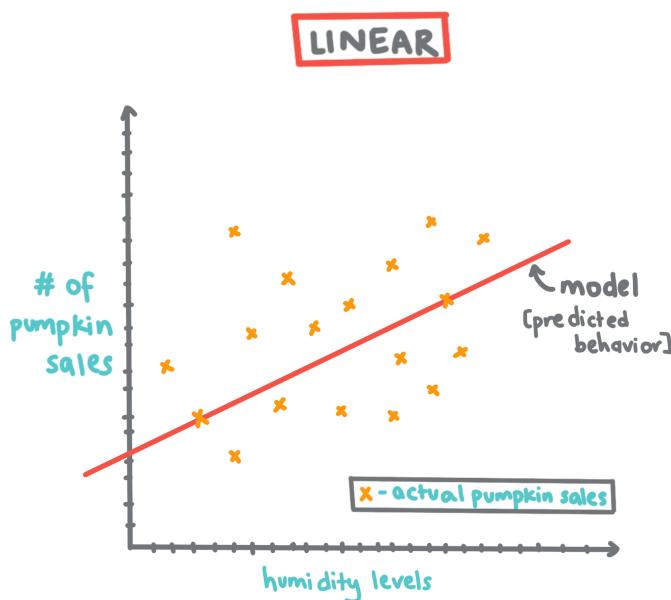
## Assignment

---

[Build a Model](#)

## Logistic Regression to Predict Categories

## LINEAR V.S LOGISTIC REGRESSION



- GOAL: find best fit line to predict output
- OUTPUT: a continuous value
- USE CASE: used to identify/forecast trends

- GOAL: find probability of an event occurring
- OUTPUT: a discrete set of probability
- USE CASE: used for categorization problems

@DASANI\_DECODED

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 will be (orange 🎃 or white 🧟).

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

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

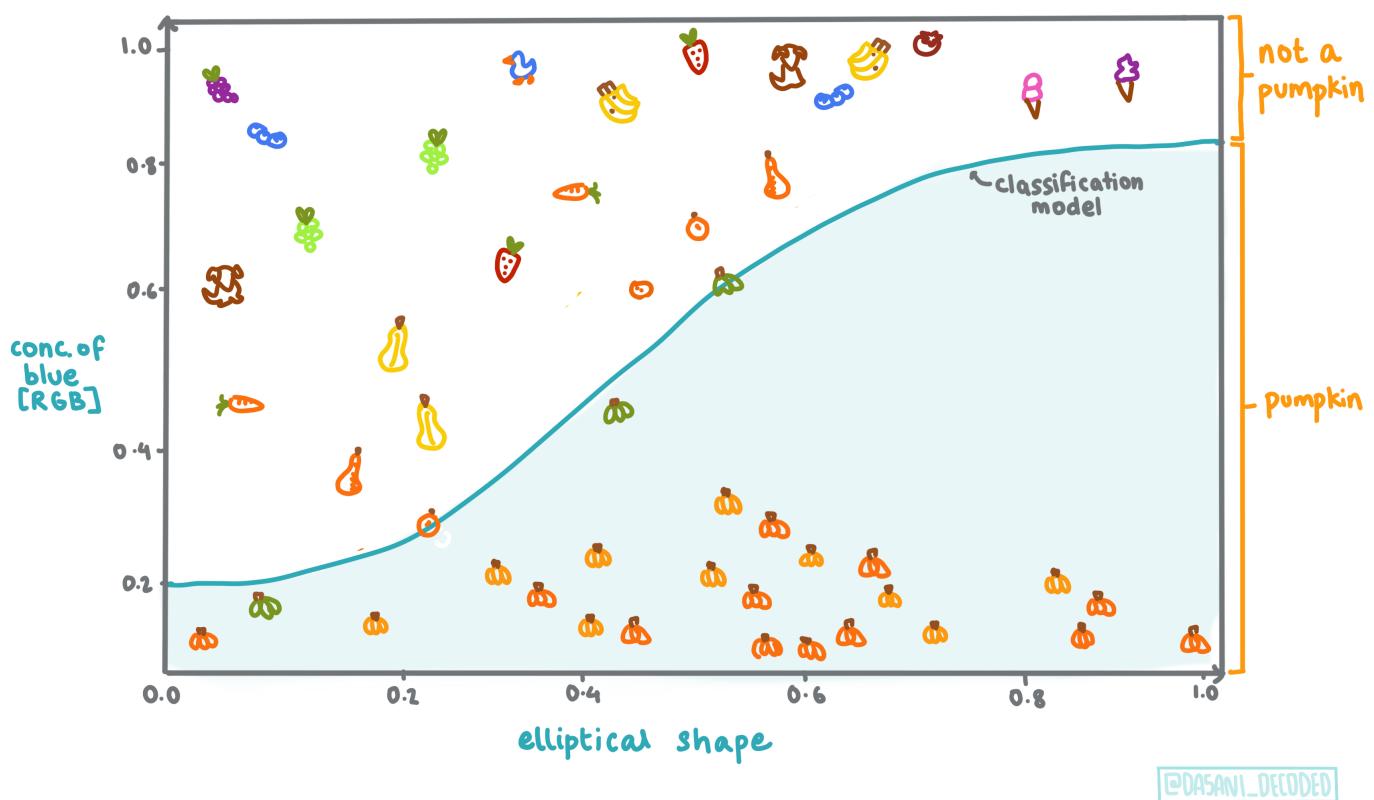
---

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.

# PUMPKIN CLASSIFICATION MODEL

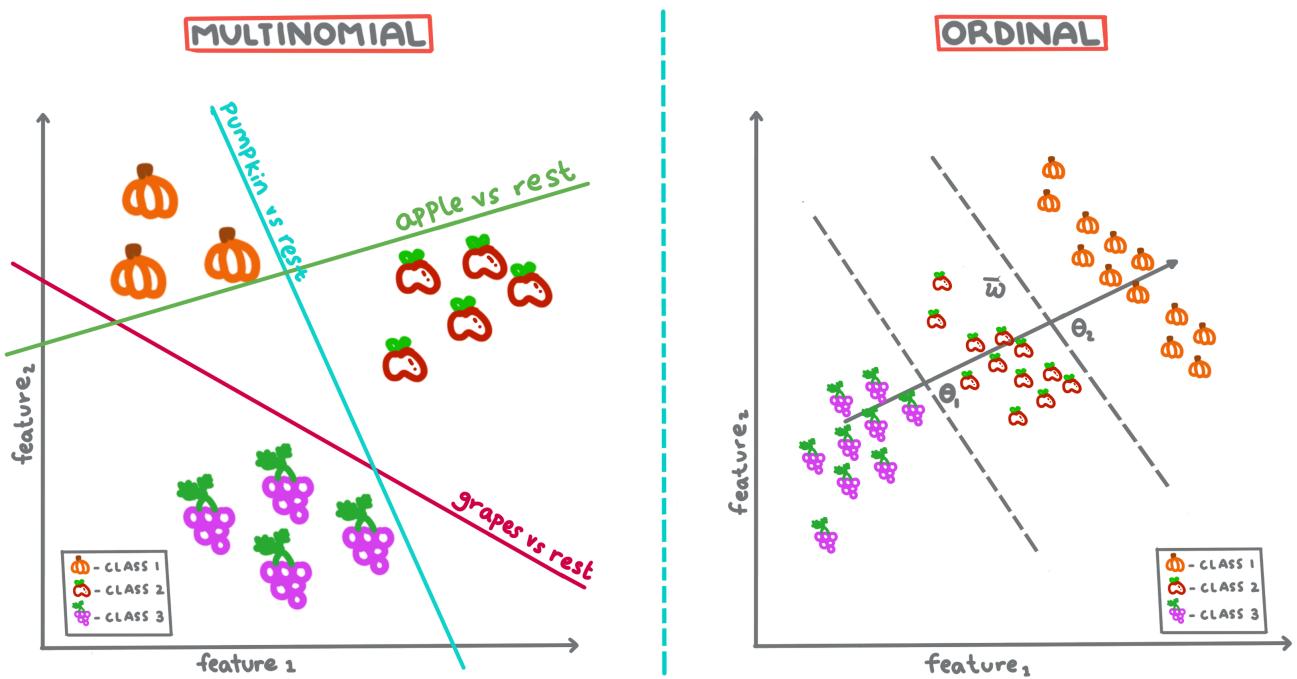


Infographic by [Dasani Madipalli](#)

## Other Classifications

There are other types of Logistic Regression, including Multinomial and Ordinal. Multinomial involves having more than one categories - "Orange, White, and Striped". Ordinal 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).

# MULTINOMIAL v.s ORDINAL LOGISTIC REGRESSION



@DASANI\_DECODED

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

Logistic Regression will give more accurate results if you use more data; our small dataset is not optimal for this task, so keep that in mind.

- ✓ Think about the types of data that would lend themselves well to Logistic Regression

## Tidy the Data

---

First, clean the data a bit, dropping null values and selecting only some of the columns:

python

```
from sklearn.preprocessing import LabelEncoder

new_columns = ['Color','Origin','Item Size','Variety','City Name','Package

new_pumpkins = pumpkins.drop([c for c in pumpkins.columns if c not in new_]

new_pumpkins.dropna(inplace=True)

new_pumpkins = new_pumpkins.apply(LabelEncoder().fit_transform)
```

You can always take a peek at your new dataframe:

python

```
new_pumpkins.info
```

## Visualization

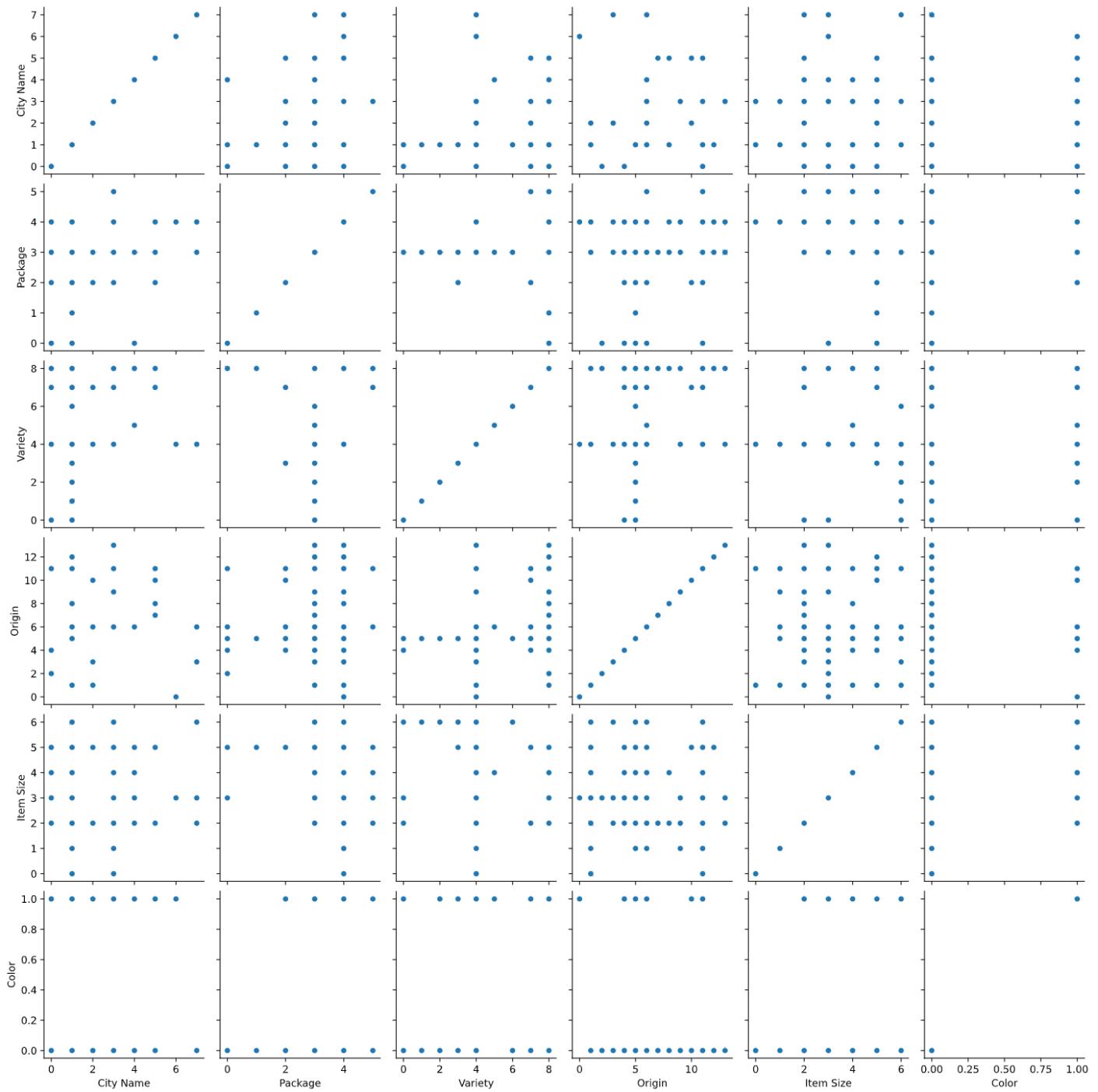
By now you have loaded up the [starter notebook](#) with pumpkin data once again and cleaned it so as to preserve a dataset containing a few variables, including Color. Let's visualize the dataframe in the notebook using a different library: [Seaborn](#), which is built on Matplotlib which we used earlier.

Seaborn offers some neat ways to visualize your data. For example, you can compare distributions of the data for each point in a side-by side grid.

python

```
import seaborn as sns

g = sns.PairGrid(new_pumpkins)
g.map(sns.scatterplot)
```



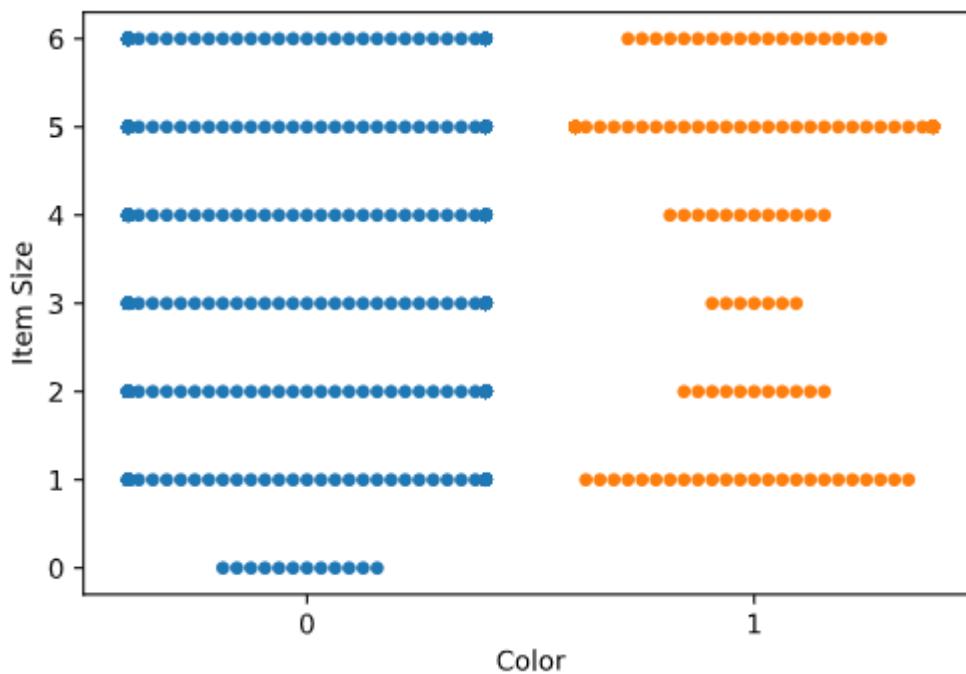
By observing data side-by-side, you can see how the Color data relates to the other columns.

Given this scatterplot grid, what are some interesting explorations you can envision?

Since Color is a binary category (Orange or Not), it's called 'categorical data' and needs 'a more specialized approach to visualization'. There are other ways to visualize the relationship of this category with other variables. You can visualize variables side-by-side with Seaborn plots. Try a 'swarm' plot to show the distribution of values:

python

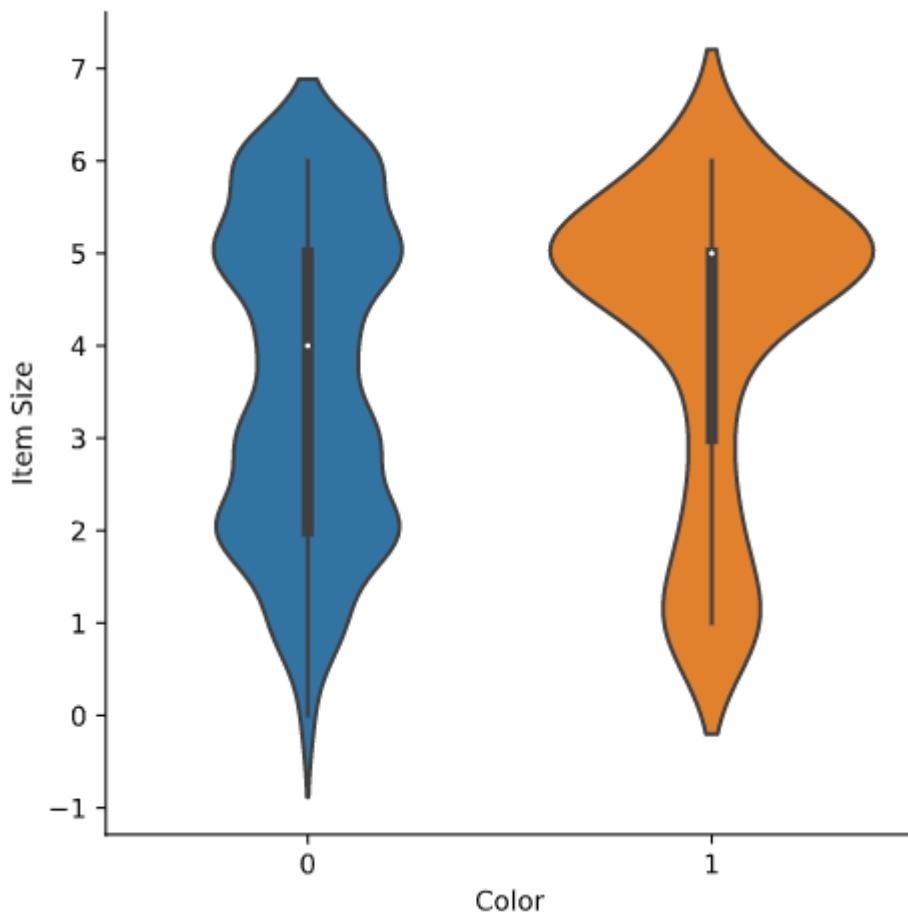
```
sns.swarmplot(x="Color", y="Item Size", data=new_pumpkins)
```



A 'violin' type plot is useful as you can easily visualize the way that data in the two categories is distributed. Violin plots don't work so well with smaller datasets as the distribution is displayed more 'smoothly'.

python

```
sns.catplot(x="Color", y="Item Size",
             kind="violin", data=new_pumpkins)
```



Try creating this plot, and other Seaborn plots, using other variables.

Now that we have an idea of the relationship between the binary categories of color and the larger group of sizes, let's explore Logistic Regression to determine a given pumpkin's likely color.

### Show Me The Math

Remember how Linear Regression often used ordinary least squares to arrive at a value?

Logistic Regression relies on the concept of 'maximum likelihood' using sigmoid functions. A 'Sigmoid Function' on a plot looks like an 'S' shape. It takes a value and maps it to somewhere between 0 and 1. Its curve is also called a 'logistic curve'. Its formula looks like thus:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

where the sigmoid's midpoint finds itself at x's 0 point, L is the curve's maximum value, and k is the curve's steepness. If the outcome of the function is more than 0.5, the label in question will be given the class '1' of the binary choice. If not, it will be classified as '0'.

## Build your model

Building a model to find these binary classification is surprisingly straightforward in Scikit-Learn.

Select the variables you want to use in your classification model and split the training and test sets:

python

```
from sklearn.model_selection import train_test_split

Selected_features = ['Origin', 'Item Size', 'Variety', 'City Name', 'Package']

X = new_pumpkins[Selected_features]
y = new_pumpkins['Color']

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

Now you can train your model and print out its result:

python

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)

print(classification_report(y_test, predictions))
print('Predicted labels: ', predictions)
print('Accuracy: ', accuracy_score(y_test, predictions))
```

Take a look at your model's scoreboard. It's not too bad, considering you have only about 1000 rows of data:

## Better comprehension via a confusion matrix

While you can get a scoreboard report terms by printing out the items above, you might be able to understand your model more easily by using a confusion matrix to help us understand how the model is performing.

🎓 A 'confusion matrix' (or 'error matrix') is a table that expresses your model's true vs. false positives and negatives, thus gauging the accuracy of predictions.

python

```
from sklearn.metrics import confusion_matrix  
confusion_matrix(y_test, predictions)
```

Take a look at your model's confusion matrix:

```
array([[162,    4],  
       [ 33,    0]])
```

What's going on here? Let's say our model is asked to classify items between two binary categories, category 'pumpkin' and category 'not-a-pumpkin'.

- If your model predicts something as a pumpkin and it belongs to category 'pumpkin' in reality we call it a true positive, shown by the top left number.
- If your model predicts something as not a pumpkin and it belongs to category 'pumpkin' in reality we call it a false positive, shown by the top right number.
- If your model predicts something as a pumpkin and it belongs to category 'not-a-pumpkin' in reality we call it a false negative, shown by the bottom left number.
- If your model predicts something as not a pumpkin and it belongs to category 'not-a-pumpkin' in reality we call it a true negative, shown by the bottom right number.

# Actual values

		Pumpkin	Not-a-Pumpkin
Predicted values	Pumpkin		
	Not-a-Pumpkin		
		TP	FP
		FN	TN

Infographic by [Jen Looper](#)

As you might have guessed it's preferable to have a larger number of true positives and true negatives and a lower number of false positives and false negatives, which implies that the model performs better.

- Q: According to the confusion matrix, how did the model do? A: Not too bad; there are a good number of true positives but also several false negatives.

Let's revisit the terms we saw earlier with the help of the confusion matrix's mapping of TP/TN and FP/FN:

- 👉 Precision:  $TP/(TP + FN)$  The fraction of relevant instances among the retrieved instances (e.g. which labels were well-labeled)
- 👉 Recall:  $TP/(TP + FP)$  The fraction of relevant instances that were retrieved, whether well-labeled or not
- 👉 f1-score:  $(2 * \text{precision} * \text{recall})/(\text{precision} + \text{recall})$  A weighted average of the precision and recall, with best being 1 and worst being 0
- 👉 Support: The number of occurrences of each label retrieved
- 👉 Accuracy:  $(TP + TN)/(TP + TN + FP + FN)$  The percentage of labels predicted accurately for a sample.

🎓 Macro Avg: The calculation of the unweighted mean metrics for each label, not taking label imbalance into account.

🎓 Weighted Avg: The calculation of the mean metrics for each label, taking label imbalance into account by weighting them by their support (the number of true instances for each label).

✓ Can you think which metric you should watch if you want your model to reduce the number of false negatives?

## Visualize the ROC Curve of this Model

---

This is not a bad model; its accuracy is in the 80% range so ideally you could use it to predict the color of a pumpkin given a set of variables.

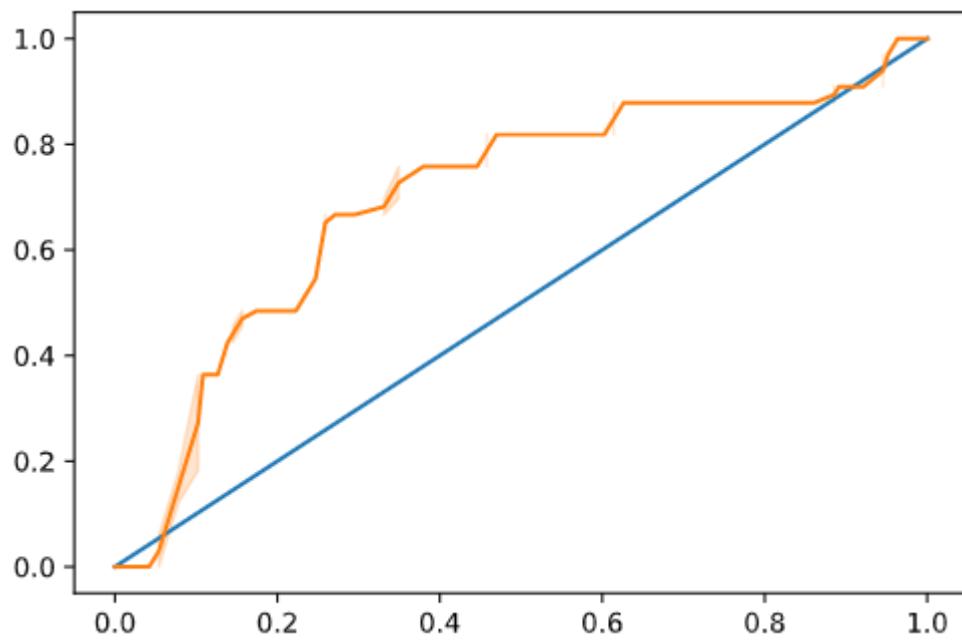
Let's do one more visualization to see the so-called 'ROC' score:

python

```
from sklearn.metrics import roc_curve, roc_auc_score

y_scores = model.predict_proba(X_test)
# calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])
sns.lineplot([0, 1], [0, 1])
sns.lineplot(fpr, tpr)
```

Using Seaborn again, plot the model's Receiving Operating Characteristic or ROC. ROC curves are often used to get a view of the output of a classifier in terms of its true vs. false positives. "ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis." Thus, the steepness of the curve and the space between the midpoint line and the curve matter: you want a curve that quickly heads up and over the line. In our case, there are false positives to start with, and then the line heads up and over properly:



Finally, use Scikit-Learn's [roc\\_auc\\_score API](#) to compute the actual 'Area Under the Curve' (AUC):

```
python
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

The result is `0.6976998904709748`. Given that the AUC ranges from 0 to 1, you want a big score, since a model that is 100% correct in its predictions will have an AUC of 1; in this case, the model is *pretty good*.

In future lessons on classifications, you will learn how to iterate to improve your model's scores. But for now, congratulations! You've completed these regression lessons!

---

## 🚀 Challenge

---

There's a lot more to unpack regarding Logistic Regression! But the best way to learn is to experiment. Find a dataset that lends itself to this type of analysis and build a model with it. What do you learn? tip: try [Kaggle](#) for interesting datasets.

---

## Post-lecture quiz

---

# Review & Self Study

---

Read the first few pages of [this paper from Stanford](#) on some practical uses for Logistic Regression. Think about tasks that are better suited for one or the other type of Regression tasks that we have studied up to this point. What would work best?

## Assignment

---

[Retrying this Regression](#)

## Build a Web App to use a ML Model

In this lesson, you will train a Linear Regression model and a Classification model on a dataset that's out of this world: UFO Sightings over the past century, sourced from [NUFORC's database](#). We will continue our use of notebooks to clean data and train our model, but you can take the process one step further by exploring using a model 'in the wild', so to speak: in a web app. To do this, you need to build a web app using Flask.

## Pre-lecture quiz

---

There are several ways to build web apps to consume machine learning models. Your web architecture may influence the way your model is trained. Imagine that you are working in a business where the data science group has trained a model that they want you to use in an app. There are many questions you need to ask: Is it a web app, or a mobile app? Where will the model reside, in the cloud or locally? Does the app have to work offline? And what technology was used to train the model, because that may influence the tooling you need to use?

If you are training a model using TensorFlow, for example, that ecosystem provides the ability to convert a TensorFlow model for use in a web app by using [TensorFlow.js](#). If you are building a mobile app or need to use the model in an IoT context, you could use [TensorFlow Lite](#) and use the model in an Android or iOS app.

If you are building a model using [PyTorch](#), you have the option to export it in [ONNX](#) (Open Neural Network Exchange) format for use in JavaScript web apps that can use [onnx.js](#). This option will be explored in a future lesson.

If you are using an ML SaaS (Software as a Service) system such as [Lobe.ai](#) or [Azure Custom Vision](#) to train a model, this type of software provides ways to export the model for many platforms, including building a bespoke API to be queried in the cloud by your online application.

You also have the opportunity to build an entire Flask web app that would be able to train the model itself in a web browser. This can also be done using TensorFlow.js in a JavaScript context. For our purposes, since we have been working with notebooks, let's explore the steps you need to take to export a trained model to a format readable by a Python-built web app.

## Tools

---

For this task, you need two tools: Flask and Pickle, both of which run on Python.

- ✓ What's [Flask](#)? Defined as a 'micro-framework' by its creators, Flask provides the basic features of web frameworks using Python and a templating engine to build web pages. Take a look at [this Learn module](#) to practice building with Flask.
  
- ✓ What's [Pickle](#)? Pickle  is a Python module that serializes and de-serializes a Python object structure. When you 'pickle' a model, you serialize or flatten its structure for use on the web. Be careful: pickle is not intrinsically secure, so be careful if prompted to 'un-pickle' a file. A pickled file has the suffix `.pkl`.

## Clean your data

---

In this lesson you'll use data from 80,000 UFO sightings, gathered by [NUFORC](#) (The National UFO Reporting Center). This data has some interesting descriptions of UFO sightings, for example "A man emerges from a beam of light that shines on a grassy field at night and he runs towards the Texas Instruments parking lot" or simply "the lights chased us". The [ufos.csv](#) spreadsheet includes columns about the city, state and country where the sighting occurred, the object's shape and its latitude and longitude.

In the blank [notebook](#) included in this lesson, import pandas, matplotlib, and numpy as you did in previous lessons and import the ufos spreadsheet. You can take a look at a sample data set:

python

```
import pandas as pd
import numpy as np

ufos = pd.read_csv('..../data/ufos.csv')
ufos.head()
```

Convert the ufos data to a small dataframe with fresh titles. Check the unique values in the Country field.

python

```
ufos = pd.DataFrame({'Seconds': ufos['duration (seconds)'], 'Country': ufos['Country']})  
ufos.Country.unique()
```

Now, you can reduce the amount of data we need to deal with by dropping any null values and only importing sightings between 1-60 seconds:

python

```
ufos.dropna(inplace=True)  
  
ufos = ufos[(ufos['Seconds'] >= 1) & (ufos['Seconds'] <= 60)]  
  
ufos.info()
```

Next, import Scikit-Learn's LabelEncoder library to convert the text values for countries to a number.

- LabelEncoder encodes data alphabetically

python

```
from sklearn.preprocessing import LabelEncoder  
  
ufos['Country'] = LabelEncoder().fit_transform(ufos['Country'])  
  
ufos.head()
```

Your data should look like this:

	Seconds	Country	Latitude	Longitude
2	20.0	3	53.200000	-2.916667
3	20.0	4	28.978333	-96.645833
14	30.0	4	35.823889	-80.253611
23	60.0	4	45.582778	-122.352222
24	3.0	3	51.783333	-0.783333

## Build your model

Now you can get ready to train a model by diving the data into the training and testing group. Select the three features you want to train on as your X vector, and the y vector will be the Country. You want to be able to input seconds, latitude and longitude and get a country id to return.

python

```
from sklearn.model_selection import train_test_split

Selected_features = ['Seconds','Latitude','Longitude']

X = ufos[Selected_features]
y = ufos['Country']

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

Finally, train your model using Logistic Regression:

python

```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)

print(classification_report(y_test, predictions))
print('Predicted labels: ', predictions)
print('Accuracy: ', accuracy_score(y_test, predictions))
```

The accuracy isn't bad (around 95%), unsurprisingly, as country and latitude/longitude have a good correlation. The model you created isn't very revolutionary, but it's a good exercise to try to train from raw data that you cleaned, export, and then use this model in a web app.

## Pickle your model

---

Now, it's time to pickle your model! You can do that in just a few lines of code. Load your pickled model and test it against a sample data array containing values for seconds, latitude and longitude,

python

```
import pickle
model_filename = 'ufo-model.pkl'
pickle.dump(model, open(model_filename, 'wb'))
```

```
model = pickle.load(open('ufo-model.pkl','rb'))
print(model.predict([[50,44,-12]]))
```

The model returns '3', which is the country code for the UK. Wild! 🎉

## Build a Flask app

---

Now you can build a Flask app to call your model and return similar results, but in a more visually pleasing way.

Start by creating a folder called `web-app` next to the `notebook.ipynb` file where your `ufo-model.pkl` file resides. In that folder create three more folders: `static`, with a folder `css` inside it, and `templates`.

Refer to the solution folder for a view of the finished app

The first file to create in `web-app` is a `requirements.txt` file. Like `package.json` in a JavaScript app, this file lists dependencies required by the app. In `requirements.txt` add the lines:

```
text
scikit-learn
pandas
numpy
flask
```

Now, run this file by navigating to `web-app` (`cd web-app`) in your terminal and typing `pip install -r requirements.txt`.

Now, you're ready to create three more files to finish the app:

1. Create `app.py` in the root
2. Create `index.html` in `templates`
3. Create `styles.css` in `static/css`

Build out the `styles.css` file with a few styles:

```

body {
    width: 100%;
    height: 100%;
    font-family: 'Helvetica';
    background: black;
    color: #fff;
    text-align: center;
    letter-spacing: 1.4px;
    font-size: 30px;
}

input {
    min-width: 150px;
}

.grid {
    width: 300px;
    border: 1px solid #2d2d2d;
    display: grid;
    justify-content: center;
    margin: 20px auto;
}

.box {
    color: #fff;
    background: #2d2d2d;
    padding: 12px;
    display: inline-block;
}

```

Next, build out the `index.html` file:

```

html

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>🛸 UFO Appearance Prediction! 🛸 </title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}>
</head>

<body>
    <div class="grid">

```

```

<div class="box">

<p>According to the number of seconds, latitude and longitude, which cou

<form action="{{ url_for('predict') }}" method="post">
    <input type="number" name="seconds" placeholder="Seconds" required="required">
    <input type="text" name="latitude" placeholder="Latitude" required="required">
    <input type="text" name="longitude" placeholder="Longitude" required="required">
    <button type="submit" class="btn">Predict country where the UFO is seen</button>
</form>

<p>{{ prediction_text }}</p>

</div>
</div>

</body>
</html>

```

Take a look at the templating in this file. Notice the 'mustache' syntax around variables that will be provided by the app, like the prediction text: {{}} . There's also a form that posts a prediction to the /predict route.

Finally, you're ready to build the python file that drives the consumption of the model and the display of predictions:

In app.py add:

```

python

import numpy as np
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

model = pickle.load(open("../ufo-model.pkl", "rb"))

@app.route("/")
def home():
    return render_template("index.html")

```

```

@app.route("/predict", methods=["POST"])
def predict():

    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    output = prediction[0]

    countries = ["Australia", "Canada", "Germany", "UK", "US"]

    return render_template(
        "index.html", prediction_text="Likely country: {}".format(countries)
    )

if __name__ == "__main__":
    app.run(debug=True)

```

 Tip: when you add `debug=True` while running the web app using Flask, any changes you make to your application will be reflected immediately without the need to restart the server. Beware! Don't enable this mode in a production app.

If you run `python app.py` or `python3 app.py` - your web server starts up, locally, and you can fill out a short form to get an answer to your burning question about where UFOs have been sighted!

Before doing that, take a look at the parts of `app.py`.

First, dependencies are loaded and the app starts. Then, the model is imported. Then, `index.html` is rendered on the home route. On the `/predict` route, several things happen when the form is posted:

1. The form variables are gathered and converted to a numpy array. They are then sent to the model and a prediction is returned.
2. The Countries that we want displayed are re-rendered as readable text from their predicted country code, and that value is sent back to `index.html` to be rendered in the template.

Using a model this way, with Flask and a pickled model, is relatively straightforward. The hardest thing is to understand what shape the data is that must be sent to the model to get a prediction. That

all depends on how the model was trained. This one has three data points to be input in order to get a prediction. In a professional setting, you can see how good communication is necessary between the folks who train the model and those who consume it in a web or mobile app. In our case, it's only one person, you!

## Challenge:

---

Instead of working in a notebook and importing the model to the Flask app, you could train the model right within the Flask app! Try converting your Python code in the notebook, perhaps after your data is cleaned, to train the model from within the app on a route called `train`. What are the pros and cons of pursuing this method?

## Post-lecture quiz

---

## Review & Self Study

---

There are many ways to build a web app to consume ML models. Make a list of the ways you could use JavaScript or Python to build a web app to leverage machine learning. Consider architecture: should the model stay in the app or live in the cloud? If the latter, how would you access it? Draw out an architectural model for an applied ML web solution.

## Assignment

---

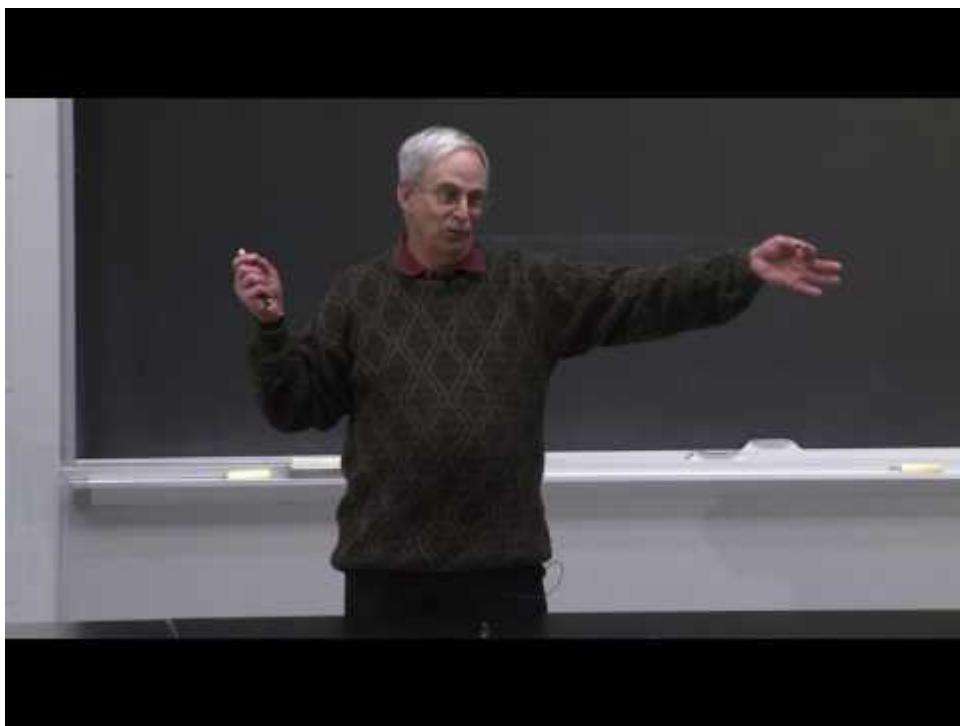
[Try a different model](#)

# Introduction to Classification

In these four lessons, you will discover the 'meat and potatoes' of classic machine learning - Classification. No pun intended - we will walk through using various classification algorithms with a dataset all about the brilliant cuisines of Asia and India. Hope you're hungry!

Classification is a form of supervised learning that bears a lot in common with Regression techniques. If machine learning is all about assigning names to things via datasets, then classification generally

falls into two groups: binary classification and multiclass classification.



🎥 Click the image above for a video: MIT's John Guttag introduces Classification

Remember, Linear Regression helped you predict relationships between variables and make accurate predictions on where a new datapoint would fall in relationship to that line. So, you could predict what price a pumpkin would be in September vs. December, for example. Logistic Regression helped you discover binary categories: at this price point, is this pumpkin orange or not-orange?

Classification uses various algorithms to determine other ways of determining a data point's label or class. Let's work with this cuisine data to see whether, by observing a group of ingredients, we can determine its cuisine of origin.

## Pre-lecture quiz

### Introduction

Classification is one of the fundamental activities of the machine learning researcher and data scientist. From basic classification of a binary value ("is this email spam or not?") to complex image classification and segmentation using computer vision, it's always useful to be able to sort data into classes and ask questions of it. Or, to state the process in a more scientific way, your classification method creates a predictive model that enables you to map the relationship between input variables to output variables.

Before starting the process of cleaning our data, visualizing it, and prepping it for our ML tasks, let's learn a bit about the various ways machine learning can be leveraged to classify data.

Derived from [statistics](#), classification using classic machine learning uses features, such as 'smoker', 'weight', and 'age' to determine 'likelihood of developing X disease'. As a supervised learning technique similar to the Regression exercises you performed earlier, your data is labeled and the ML algorithms use those labels to classify and predict classes (or 'features') of a dataset and assign them to a group or outcome.

✓ Take a moment to imagine a dataset about cuisines. What would a multiclass model be able to answer? What would a binary model be able to answer? What if you wanted to determine whether a given cuisine was likely to use fenugreek? What if you wanted to see if, given a present of a grocery bag full of star anise, artichokes, cauliflower, and horseradish, you could create a typical Indian dish?

## Hello 'classifier'

---

The question we want to ask of this cuisine dataset is actually a **multiclass question**, as we have several potential national cuisines to work with. Given a batch of ingredients, which of these many classes will the data fit?

Scikit-Learn offers several different algorithms to use to classify data, depending on the kind of problem you want to solve. In the next two lessons, you'll learn about several of these algorithms.

## Clean and Balance Your Data

---

The first task at hand before starting this project is to clean and **balance** your data to get better results. Start with the blank `notebook.ipynb` file in the root of this folder.

The first think to install is [imblearn](#). This is a Scikit-Learn package that will allow you to better balance the data (you will learn more about this task in a minute).

python

```
pip install imblearn
```

Then, import the packages you need to import your data and visualize it. Import SMOTE from imblearn.

python

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import matplotlib as mpl
import numpy as np
from imblearn.over_sampling import SMOTE
```

The next task will be to import the data:

```
df = pd.read_csv('../data/cuisines.csv')  
python
```

Check the data's shape:

```
df.head()  
python
```

The first five rows look like this:

	Unnamed: 0	cuisine	almond	angelica	anise	anise_seed	apple	apple_brandy	apric
0	65	indian	0	0	0	0	0	0	0
1	66	indian	1	0	0	0	0	0	0
2	67	indian	0	0	0	0	0	0	0
3	68	indian	0	0	0	0	0	0	0
4	69	indian	0	0	0	0	0	0	0

Get info about this data:

```
df.info()  
python
```

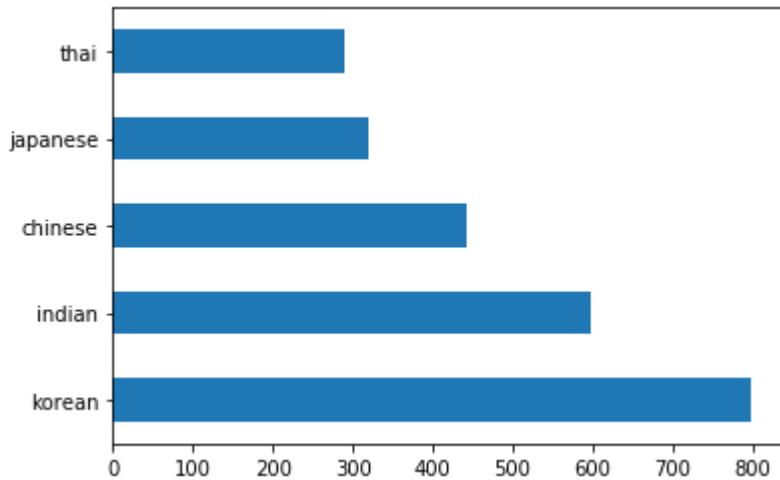
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2448 entries, 0 to 2447
Columns: 385 entries, Unnamed: 0 to zucchini
dtypes: int64(384), object(1)
memory usage: 7.2+ MB
```

# Learning about cuisines

Now the work starts to become more interesting. Let's discover the distribution of data, per cuisine:

python

```
df.cuisine.value_counts().plot.barch()
```



There are a finite number of cuisines, but the distribution of data is uneven. You can fix that! Before doing so, explore a little more. How much data exactly is available per cuisine?

python

```
thai_df = df[(df.cuisine == "thai")]
japanese_df = df[(df.cuisine == "japanese")]
chinese_df = df[(df.cuisine == "chinese")]
indian_df = df[(df.cuisine == "indian")]
korean_df = df[(df.cuisine == "korean")]

print(f'thai df: {thai_df.shape}')
print(f'japanese df: {japanese_df.shape}')
print(f'chinese df: {chinese_df.shape}')
print(f'indian df: {indian_df.shape}')
print(f'korean df: {korean_df.shape}'')
```

thai df: (289, 385) japanese df: (320, 385) chinese df: (442, 385) indian df: (598, 385) korean df: (799, 385)

## Discovering ingredients

Now you can dig deeper into the data and learn what are the typical ingredients per cuisine. You should clean out recurrent data that creates confusion between cuisines, so let's learn about this problem.

Create a function in Python to create an ingredient dataframe. This function will start by dropping an unhelpful column and sort through ingredients by their count:

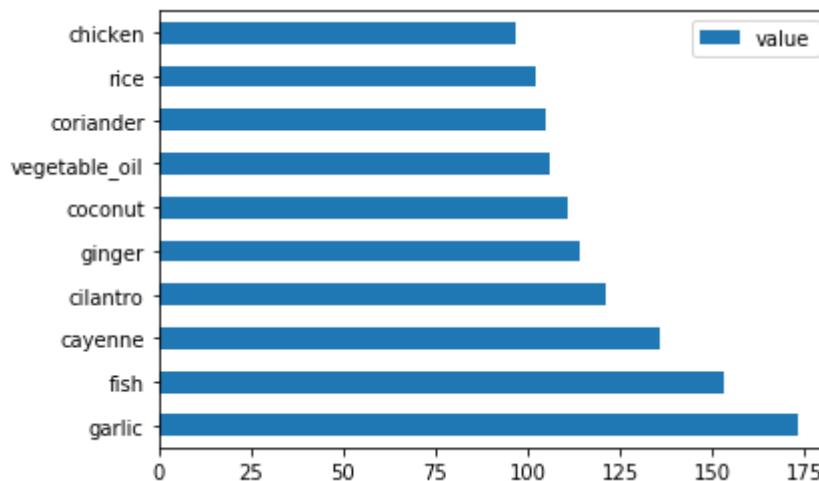
python

```
def create_ingredient_df(df):
    ingredient_df = df.T.drop(['cuisine', 'Unnamed: 0']).sum(axis=1).to_frame()
    ingredient_df = ingredient_df[(ingredient_df.T != 0).any()]
    ingredient_df = ingredient_df.sort_values(by='value', ascending=False,
                                                inplace=False)
    return ingredient_df
```

Now you can use that function to get an idea of top ten most popular ingredients by cuisine:

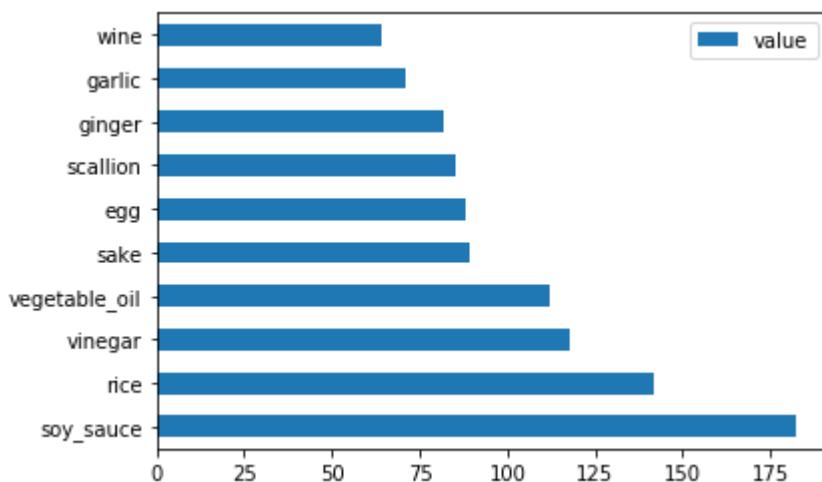
python

```
thai_ingredient_df = create_ingredient_df(thai_df)
thai_ingredient_df.head(10).plot.barh()
```



python

```
japanese_ingredient_df = create_ingredient_df(japanese_df)
japanese_ingredient_df.head(10).plot.barh()
```



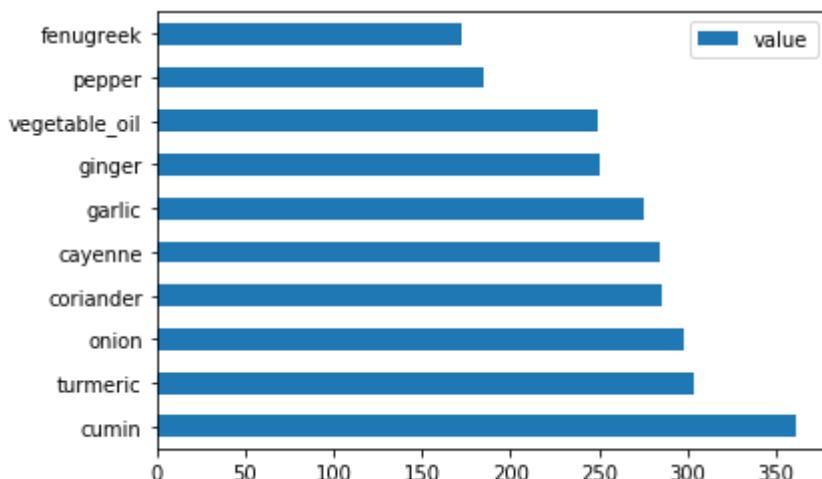
python

```
chinese_ingredient_df = create_ingredient_df(chinese_df)
chinese_ingredient_df.head(10).plot.barh()
```



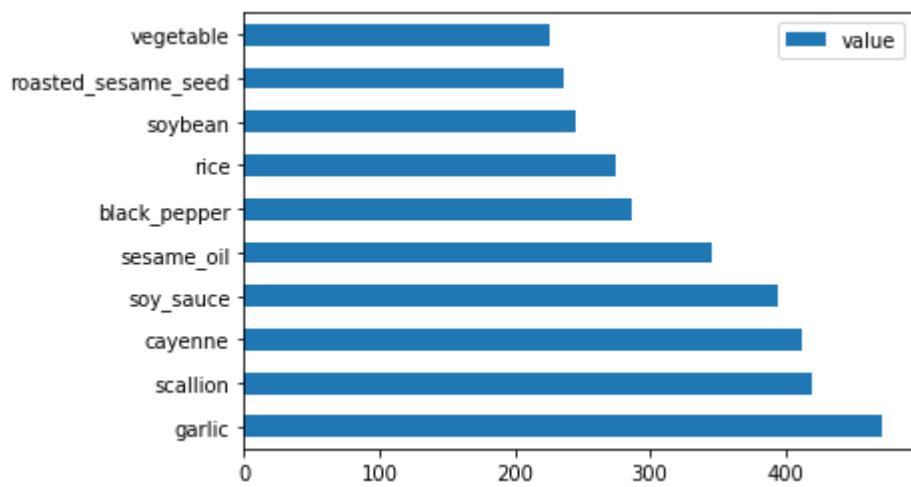
python

```
indian_ingredient_df = create_ingredient_df(indian_df)
indian_ingredient_df.head(10).plot.barh()
```



python

```
korean_ingredient_df = create_ingredient_df(korean_df)
korean_ingredient_df.head(10).plot.barh()
```



Now, drop the most common ingredients that create confusion between distinct cuisines. Everyone loves rice, garlic and ginger!

python

```
feature_df= df.drop(['cuisine','Unnamed: 0','rice','garlic','ginger'], axis=1)
labels_df = df.cuisine#.unique()
feature_df.head()
```

## Balance the dataset

---

Now that you have cleaned the data, use SMOTE - "Synthetic Minority Over-sampling Technique" - to balance it. This strategy generates new samples by interpolation.

python

```
oversample = SMOTE()
transformed_feature_df, transformed_label_df = oversample.fit_resample(fea-
```

By balancing your data, you'll have better results when classifying it. Think about a binary classification. If most of your data is one class, a ML model is going to predict that class more frequently, just because there is more data for it. Balancing the data takes any skewed data and helps remove this imbalance.

Now you can check the numbers of labels per ingredient:

python

```
print(f'new label count: {transformed_label_df.value_counts()}' )
print(f'old label count: {df.cuisine.value_counts()}' )
```

```
new label count: korean      799
chinese      799
indian       799
japanese     799
thai         799
Name: cuisine, dtype: int64
old label count: korean      799
indian       598
chinese      442
japanese     320
thai         289
Name: cuisine, dtype: int64
```

The data is nice and clean, balanced, and very delicious! You can take one more look at the data using `transformed_df.head()` and `transformed_df.info()`. Save a copy of this data for use in future lessons:

python

```
transformed_df.to_csv("../data/cleaned_cuisine.csv")
```

This fresh CSV can now be found in the root data folder.

## Challenge

---

This curriculum contains several interesting datasets. Dig through the `data` folders and see if any contain datasets that would be appropriate for binary or multi-class classification? What questions would you ask of this dataset?

## Post-lecture quiz

---

## Review & Self Study

---

Explore SMOTE's API. What use cases is it best used for? What problems does it solve?

# Assignment

---

[Explore classification methods](#)

## Cuisine Classifiers 1

In this lesson, you will use the dataset you saved from the last lesson full of balanced, clean data all about cuisines. You will use this dataset with a variety of classifiers to predict a given national cuisine based on a group of ingredients. While doing so, you'll learn more about some of the ways that algorithms can be leveraged for classification tasks.

### Pre-lecture quiz

---

## Preparation

Assuming you completed Lesson 1, make sure that a `cleaned_cuisines.csv` file exists in the root `/data` folder for these four lessons.

Working in this lesson's `notebook.ipynb` folder, import that file along with the Pandas library:

python

```
import pandas as pd  
cuisines_df = pd.read_csv("../data/cleaned_cuisine.csv")  
cuisines_df.head()
```

The data looks like this:

Unnamed: 0	cuisine	almond	angelica	anise	anise_seed	apple	apple_braney	apric
0 0	indian	0	0	0	0	0	0	0
1 1	indian	1	0	0	0	0	0	0
2 2	indian	0	0	0	0	0	0	0

Unnamed: 0	cuisine	almond	angelica	anise	anise_seed	apple	apple_brandy	apric
3 3	indian	0	0	0	0	0	0	0
4 4	indian	0	0	0	0	0	0	0

Now, import several more libraries:

python

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score,precision_score,confusion_matrix
from sklearn.svm import SVC
import numpy as np
```

Divide the X and y coordinates into two dataframes for training. `cuisine` can be the labels dataframe:

python

```
cuisines_label_df = cuisines_df['cuisine']
cuisines_label_df.head()
```

It will look like this:

```
0    indian
1    indian
2    indian
3    indian
4    indian
Name: cuisine, dtype: object
```

Drop that `Unnamed: 0` column and the `cuisine` column and save the rest of the data as trainable features:

python

```
cuisines_feature_df = cuisines_df.drop(['Unnamed: 0', 'cuisine'], axis=1)
cuisines_feature_df.head()
```

Your features look like this:

almond	angelica	anise	anise_seed	apple	apple_brandy	apricot	armagnac	artemisia	blackberry
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0

Now you are ready to train your model!

## Choosing your classifier

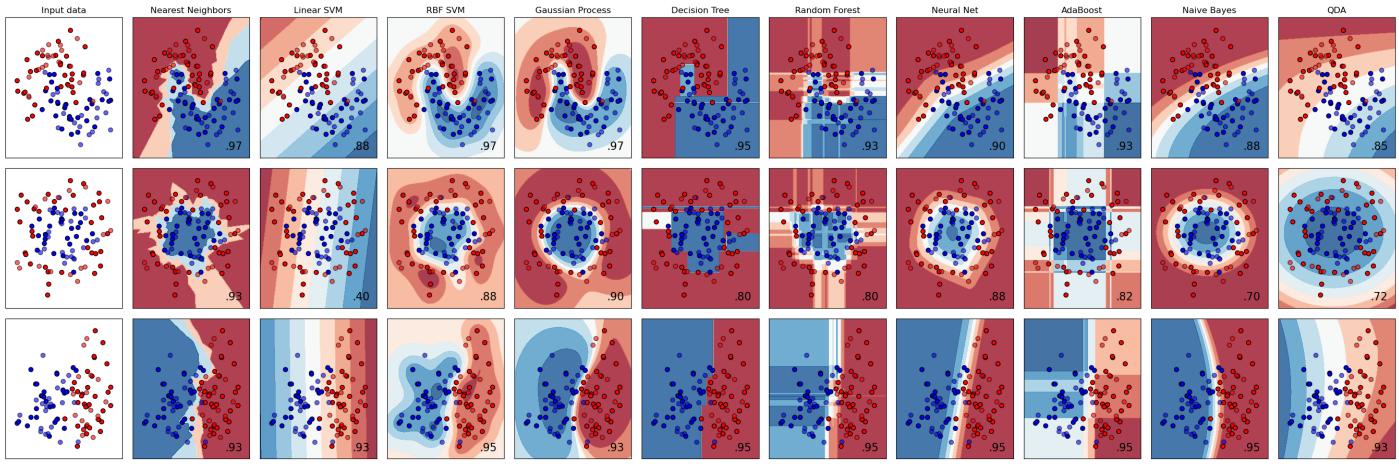
Now that your data is clean and ready for training, you have to decide which algorithm to use for the job.

Scikit-Learn groups Classification under Supervised Learning, and in that category you will find many ways to classify. The variety is quite bewildering at first sight. The following methods all include classification techniques:

- Linear Models
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Decision Trees
- Ensemble methods (voting Classifier)
- Multiclass and multioutput algorithms (multiclass and multilabel classification, multiclass-multioutput classification)

You can also use neural networks to classify, but that is outside the scope of this lesson.

So, which classifier should you choose? Often, running through several and looking for a good result is a way to test. Scikit-Learn offers a side-by-side comparison on a created dataset, comparing KNeighbors, SVC two ways, GaussianProcessClassifier, DecisionTreeClassifier, RandomForestClassifier, MLPClassifier, AdaBoostClassifier, GaussianNB and QuadraticDiscrinatAnalysis, showing the results visualized:



Plots generated on Scikit-Learn's documentation

AutoML solves this problem neatly by running these comparisons in the cloud, allowing you to choose the best algorithm for your data. Try it [here](#)

A better way than wildly guessing, however, is to follow the ideas on this downloadable [ML Cheat sheet](#). Here, we discover that, for our multiclass problem, we have some choices:

## Multiclass Classification

**Answers complex questions with multiple possible answers**

*Answers questions like: Is this A or B or C or D?*

### Multiclass Logistic Regression

Fast training times,  
linear model

### Multiclass Neural Network

Accuracy, long training times

### Multiclass Decision Forest

Accuracy, fast training times

### One-vs-All Multiclass

Depends on the  
two-class classifier

### Multiclass Boosted Decision Tree

Non-parametric, fast  
training times and scalable

A section of Microsoft's Algorithm Cheat Sheet, detailing multiclass classification options

✓ Download this cheat sheet, print it out, and hang it on your wall!

Given our clean, but minimal dataset, and the fact that we are running training locally via notebooks, neural networks are too heavyweight for this task. We do not use a two-class classifier, so that rules out One-vs-All. A Decision Tree might work, or Logistic Regression for multiclass data. The Multiclass Boosted Decision Tree is most suitable for nonparametric tasks, e.g. tasks designed to build rankings, so it is not useful for us.

We can focus on Decision Trees and Logistic Regression.

Let's focus on Logistic Regression for our first training trial since you recently learned about it in a previous lesson.

# Train your model

---

Let's train that model. Split your data into training and testing groups:

python

```
X_train, X_test, y_train, y_test = train_test_split(cuisines_feature_df, ci
```

There are many ways to use the LogisticRegression library in Scikit-Learn. Take a look at the [parameters to pass](#).

According to the docs, "In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi\_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi\_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)"

Since you are using the multiclass case, you need to choose what scheme to use and what 'solver' to set.

Use LogisticRegression with a multiclass setting and the liblinear solver to train.

 The 'scheme' here can either be 'ovr' (one-vs-rest) or 'multinomial'. Since Logistic Regression is really designed to support binary classification, these schemes allow it to better handle multiclass classification tasks. [source](#)

 The 'solver' is defined as "the algorithm to use in the optimization problem". [source](#).

Scikit-Learn offers this table to explain how solvers handle different challenges presented by different kinds of data structures:

	Solvers				
Penalties	'liblinear'	'lbfgs'	'newton-cg'	'sag'	'saga'
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

python

```
lr = LogisticRegression(multi_class='ovr', solver='liblinear')
model = lr.fit(X_train, np.ravel(y_train))

accuracy = model.score(X_test, y_test)
print ("Accuracy is {}".format(accuracy))
```

 Try a different solver like `lbfgs`, which is often set as default

Note, use Pandas `ravel` function to flatten your data when needed.

The accuracy is good at over 80%!

You can see this model in action by testing one row of data (#50):

python

```
print(f'ingredients: {X_test.iloc[50][X_test.iloc[50]!=0].keys()}')
print(f'cuisine: {y_test.iloc[50]}' )
```

The result is printed:

```
ingredients: Index(['cilantro', 'onion', 'pea', 'potato', 'tomato', 'vegetable'],
cuisine: indian
```

 Try a different row number and check the results

Digging deeper, you can check for the accuracy of this prediction:

python

```
test= X_test.iloc[50].values.reshape(-1, 1).T  
proba = model.predict_proba(test)  
classes = model.classes_  
resultdf = pd.DataFrame(data=proba, columns=classes)  
  
topPrediction = resultdf.T.sort_values(by=[0], ascending = [False])  
topPrediction.head()
```

The result is printed - Indian cuisine is its best guess, with good probability:

0

---

indian 0.715851

---

chinese 0.229475

---

japanese 0.029763

---

korean 0.017277

---

thai 0.007634

✓ Can you explain why the model is pretty sure this is an Indian cuisine?

Get more detail by printing a classification report, as you did in the Regression lessons:

python

```
y_pred = model.predict(X_test)  
print(classification_report(y_test,y_pred))
```

---

precision	recall	f1-score	support
-----------	--------	----------	---------

---

chinese	0.73	0.71	0.72	229
---------	------	------	------	-----

---

indian	0.91	0.93	0.92	254
--------	------	------	------	-----

---

japanese	0.70	0.75	0.72	220
----------	------	------	------	-----

---

korean	0.86	0.76	0.81	242
--------	------	------	------	-----

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

---

thai	0.79	0.85	0.82	254
------	------	------	------	-----

---

accuracy	0.80	1199		
----------	------	------	--	--

---

macro avg	0.80	0.80	0.80	1199
-----------	------	------	------	------

---

weighted avg	0.80	0.80	0.80	1199
--------------	------	------	------	------

---

## Challenge

---

In this lesson, you used your cleaned data to build a machine learning model that can predict a national cuisine based on a series of ingredients. Take some time to read through the many options Scikit-Learn provides to classify data. Dig deeper into the concept of 'solver' to understand what goes on behind the scenes.

## Post-lecture quiz

---

## Review & Self Study

---

Dig a little more into the math behind Logistic Regression in [this lesson](#)

## Assignment

---

[Study the solvers](#)

# Cuisine Classifiers 2

In this second Classification lesson, you will explore more ways to classify numeric data, and the ramifications for choosing one over the other.

# Pre-lecture quiz

## Prerequisite

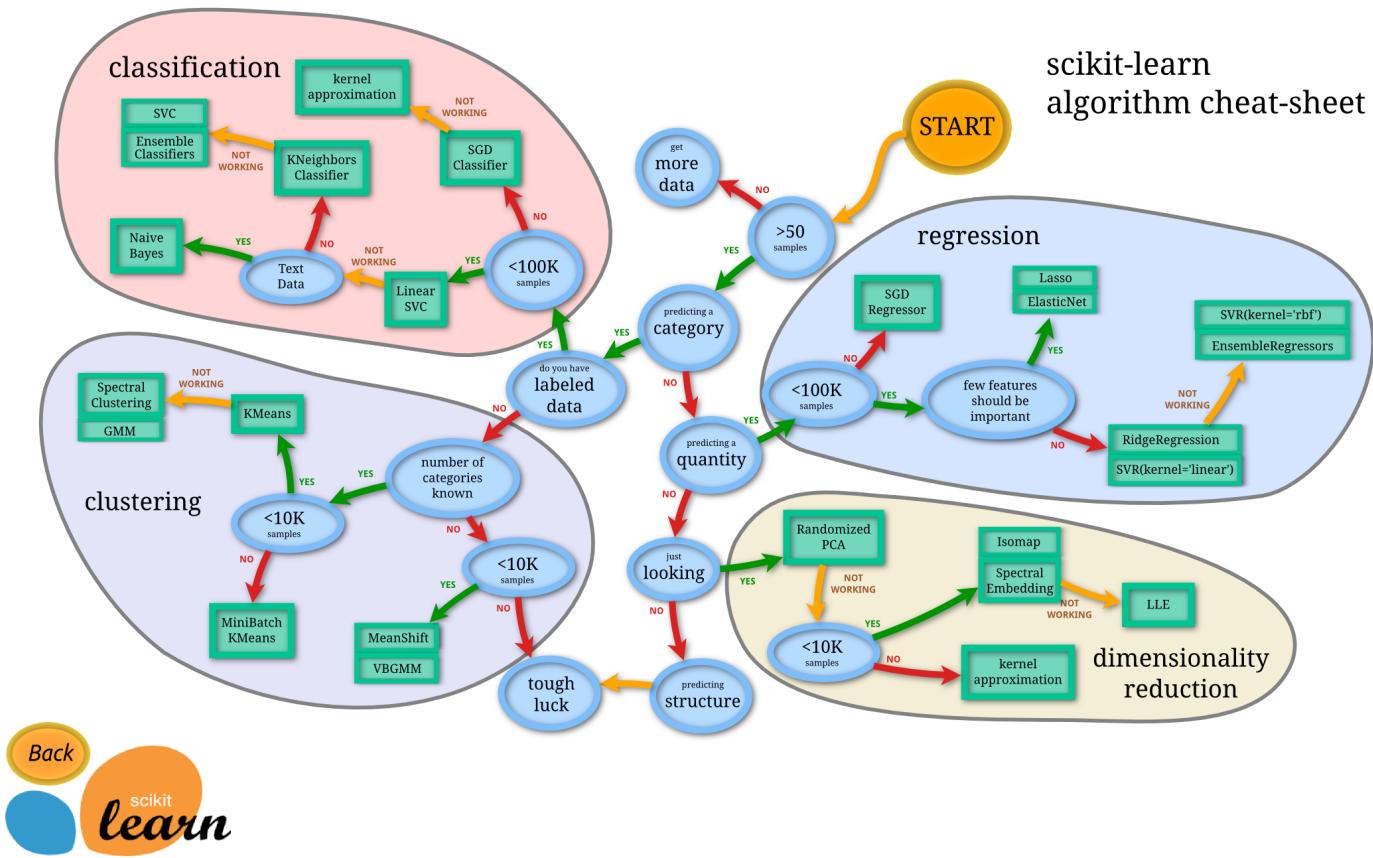
We assume that you have completed the previous lessons and have a cleaned dataset in your data folder called `cleaned_cuisine.csv` in the root of this 4-lesson folder.

## Preparation

We have loaded your `notebook.ipynb` file with the cleaned dataset and have divided it into X and y dataframes, ready for the model building process.

## A Classification Map

Previously, you learned about the various options you have when classifying data using Microsoft's cheat sheet. Scikit-Learn offers a similar, but more granular cheat sheet that can further help narrow down your estimators (another term for classifiers):



Tip: [visit this map online](#) and click along the path to read documentation.

This map is very helpful once you have a clear grasp of your data, as you can 'walk' along its paths to a decision:

- We have >50 samples
- We want to predict a category
- We have labeled data
- We have fewer than 100K samples
- ✨ We can choose a Linear SVC
- If that doesn't work, since we have numeric data
  - We can try a ✨ KNeighbors Classifier
    - If that doesn't work, try ✨ SVC and ✨ Ensemble Classifiers

This is a terrific trail to try. Following this path, we should start by importing some libraries to use:

python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix
import numpy as np
```

Split your training and test data:

python

```
X_train, X_test, y_train, y_test = train_test_split(cuisines_feature_df, cui
```

## Linear SVC Classifier

---

Start by creating an array of classifiers. You will add progressively to this array as we test. Start with a Linear SVC:

python

```
C = 10
# Create different classifiers.
classifiers = {
    'Linear SVC': SVC(kernel='linear', C=C, probability=True, random_state=42)}
```

Train your model using the Linear SVC and print out a report:

python

```
n_classifiers = len(classifiers)

for index, (name, classifier) in enumerate(classifiers.items()):
    classifier.fit(X_train, np.ravel(y_train))

    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy (train) for %s: %0.1f%%" % (name, accuracy * 100))
    print(classification_report(y_test,y_pred))
```

The result is pretty good:

Accuracy (train) for Linear SVC: 78.6%

	precision	recall	f1-score	support
chinese	0.71	0.67	0.69	242
indian	0.88	0.86	0.87	234
japanese	0.79	0.74	0.76	254
korean	0.85	0.81	0.83	242
thai	0.71	0.86	0.78	227
accuracy			0.79	1199
macro avg	0.79	0.79	0.79	1199
weighted avg	0.79	0.79	0.79	1199

## Learn about Linear SVC

Support-Vector Clustering (SVC) is a child of the Support-Vector machines family of ML techniques (learn more about these below). In this method, you can choose a 'kernel' to decide how to cluster the labels. The 'C' parameter refers to 'regularization' which regulates the influence of parameters. The kernel can be one of several; here we set it to 'linear' to ensure that we leverage Linear SVC. Probability defaults to 'false'; here we set it to 'true' to gather probability estimates. We set the random state to '0' to shuffle the data to get probabilities.

## K-Neighbors Classifier

---

The previous classifier was good, and worked well with the data, but maybe we can get better accuracy. Try a K-Neighbors Classifier. Add a line to your classifier array (add a comma after the Linear SVC item):

python

```
'KNN classifier': KNeighborsClassifier(C),
```

The result is a little worse:

```
Accuracy (train) for KNN classifier: 73.8%
precision    recall   f1-score   support
chinese      0.64      0.67      0.66      242
indian       0.86      0.78      0.82      234
japanese     0.66      0.83      0.74      254
korean        0.94      0.58      0.72      242
thai          0.71      0.82      0.76      227
accuracy           0.74      0.74      0.74      1199
macro avg      0.76      0.74      0.74      1199
weighted avg    0.76      0.74      0.74      1199
```

 Learn about [K-Neighbors](#)

K-Neighbors is part of the "neighbors" family of ML methods, which can be used for both supervised and unsupervised learning. In this method, a predefined number of points is created and data are gathered around these points such that generalized labels can be predicted for the data.

## Support Vector Classifier

---

Let's try for a little better accuracy with a Support Vector Classifier. Add a comma after the K-Neighbors item, and then add this line:

python

```
'SVC': SVC(),
```

The result is quite good!

Accuracy (train) for SVC: 83.2%

	precision	recall	f1-score	support
chinese	0.79	0.74	0.76	242
indian	0.88	0.90	0.89	234
japanese	0.87	0.81	0.84	254
korean	0.91	0.82	0.86	242
thai	0.74	0.90	0.81	227
accuracy			0.83	1199
macro avg	0.84	0.83	0.83	1199
weighted avg	0.84	0.83	0.83	1199

Learn about [Support-Vectors](#)

Support-Vector Classifiers are part of the [Support-Vector Machine](#) family of ML methods that are used for classification and regression tasks. SVMs "map training examples to points in space" to maximize the distance between two categories. Subsequent data is mapped into this space so their category can be predicted.

## Ensemble Classifiers

---

Let's follow the path to the very end, even though the previous test was quite good. Let's try some Ensemble Classifiers, specifically Random Forest and AdaBoost:

```
'RFST': RandomForestClassifier(n_estimators=100),  
'ADA': AdaBoostClassifier(n_estimators=100)
```

The result is very good, especially for Random Forest:

Accuracy (train) for RFST: 84.5%

	precision	recall	f1-score	support
chinese	0.80	0.77	0.78	242
indian	0.89	0.92	0.90	234
japanese	0.86	0.84	0.85	254
korean	0.88	0.83	0.85	242
thai	0.80	0.87	0.83	227

accuracy		<b>0.84</b>	1199
macro avg	<b>0.85</b>	<b>0.85</b>	<b>0.84</b>
weighted avg	<b>0.85</b>	<b>0.84</b>	<b>0.84</b>

Accuracy (train) for ADA: 72.4%

	precision	recall	f1-score	support
chinese	<b>0.64</b>	<b>0.49</b>	<b>0.56</b>	242
indian	<b>0.91</b>	<b>0.83</b>	<b>0.87</b>	234
japanese	<b>0.68</b>	<b>0.69</b>	<b>0.69</b>	254
korean	<b>0.73</b>	<b>0.79</b>	<b>0.76</b>	242
thai	<b>0.67</b>	<b>0.83</b>	<b>0.74</b>	227
accuracy			<b>0.72</b>	1199
macro avg	<b>0.73</b>	<b>0.73</b>	<b>0.72</b>	1199
weighted avg	<b>0.73</b>	<b>0.72</b>	<b>0.72</b>	1199

### Learn about [Ensemble Classifiers](#)

This method of Machine Learning "combines the predictions of several base estimators" to improve the model's quality. In our example, we used Random Trees and AdaBoost.

- [Random Forest](#), an averaging method, builds a 'forest' of 'decision trees' infused with randomness to avoid overfitting. The n\_estimators parameter is set to the number of trees.
- [AdaBoost](#) fits a classifier to a dataset and then fits copies of that classifier to the same dataset. It focuses on the weights of incorrectly classified items and adjusts the fit for the next classifier to correct.

## Challenge

---

Each of these techniques has a large number of parameters that you can tweak. Research each one's default parameters and think about what tweaking these parameters would mean for the model's quality.

## Post-lecture quiz

---

# Review & Self Study

There's a lot of jargon in these lessons, so take a minute to review [this list](#) of useful terminology!

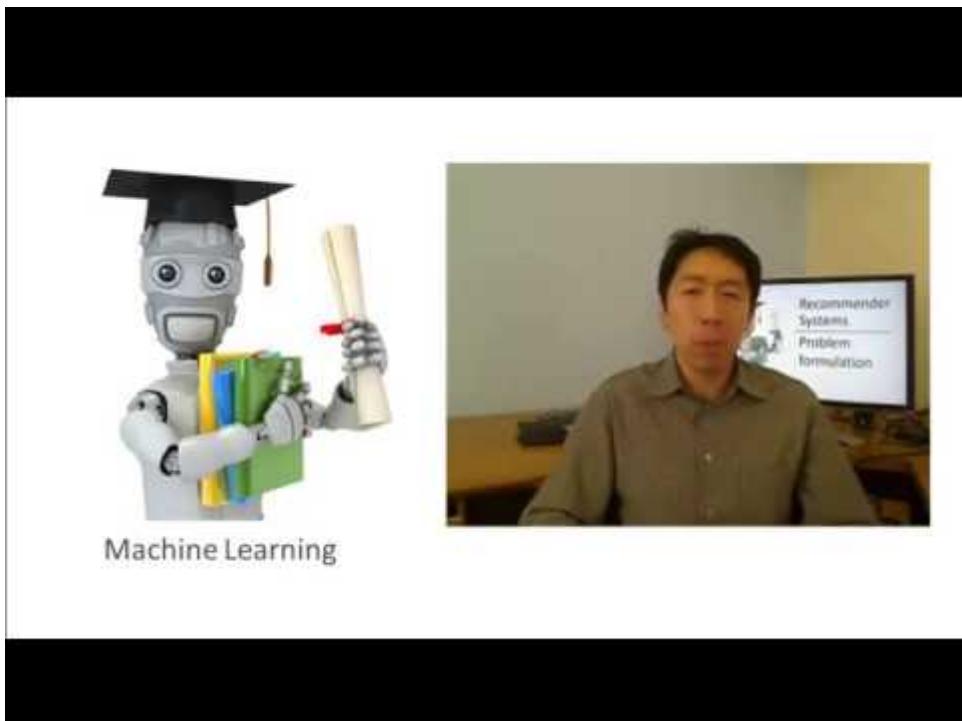
## Assignment

[Parameter Play.](#)

## Build a Cuisine Recommender Web App

In this lesson, you will build a classification model using some of the techniques you have learned in previous lessons and with the delicious cuisine dataset used throughout this series. In addition, you will build a small web app to use a saved model, leveraging Onnx's web runtime.

One of the most useful practical uses of machine learning is building recommendation systems, and you can take the first step in that direction today!



 Click the image above for a video: Andrew Ng introduces recommendation system design

## [Pre-lecture quiz](#)

In this lesson you will learn:

- How to build a model and save it as an Onnx model
- How to use Netron to inspect the model
- How to use your model in a web app for inference

## Build your model

---

Building applied ML systems is an important part of leveraging these technologies for your business systems. You can use models within your web applications (and thus use them in an offline context if needed) by using Onnx. In a [previous lesson](#), you built a Regression model about UFO sightings, "pickled" it, and used it in a Flask app. While this architecture is very useful to know it is a full-stack Python app, and your requirements may include the use of a JavaScript application. In this lesson, you can build a basic JavaScript-based system for inference. First, however, you need to train a model and convert it for use with Onnx.

First, train a classification model using the cleaned cuisines dataset we used. Start by importing useful libraries:

python

```
pip install skl2onnx  
import pandas as pd
```

You need '[skl2onnx](#)' to help convert your Scikit-Learn model to Onnx format.

Then, work with your data in the same way you did in previous lessons:

python

```
data = pd.read_csv('../data/cleaned_cuisine.csv')  
data.head()
```

Remove the first two unnecessary columns and save the remaining data as 'X':

python

```
X = data.iloc[:,2:]  
X.head()
```

Save the labels as 'y':

python

```
y = data[['cuisine']]  
y.head()
```

Commence the training routine. We will use the 'SVC' library which has good accuracy. Import the appropriate libraries from Scikit-Learn:

python

```
from sklearn.model_selection import train_test_split  
from sklearn.svm import SVC  
from sklearn.model_selection import cross_val_score  
from sklearn.metrics import accuracy_score,precision_score,confusion_matrix
```

Separate training and test sets:

python

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

Build an SVC Classification model as you did in the previous lesson:

python

```
model = SVC(kernel='linear', C=10, probability=True,random_state=0)  
model.fit(X_train,y_train.values.ravel())
```

Now, test your model:

python

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

Print out a classification report to check the model's quality:

python

```
print(classification_report(y_test,y_pred))
```

As we saw before, the accuracy is good:

	precision	recall	f1-score	support
chinese	0.72	0.69	0.70	257
indian	0.91	0.87	0.89	243
japanese	0.79	0.77	0.78	239

korean	0.83	0.79	0.81	236
thai	0.72	0.84	0.78	224
accuracy			0.79	1199
macro avg	0.79	0.79	0.79	1199
weighted avg	0.79	0.79	0.79	1199

Now, convert your model to Onnx. Make sure to do the conversion with the proper Tensor number. This dataset has 380 ingredients listed, so you need to note that number in `FloatTensorType` :

python

```
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

initial_type = [('float_input', FloatTensorType([None, 380]))]
options = {id(model): {'nocl': True, 'zipmap': False}}
onx = convert_sklearn(model, initial_types=initial_type, options=options)
with open("./model.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```

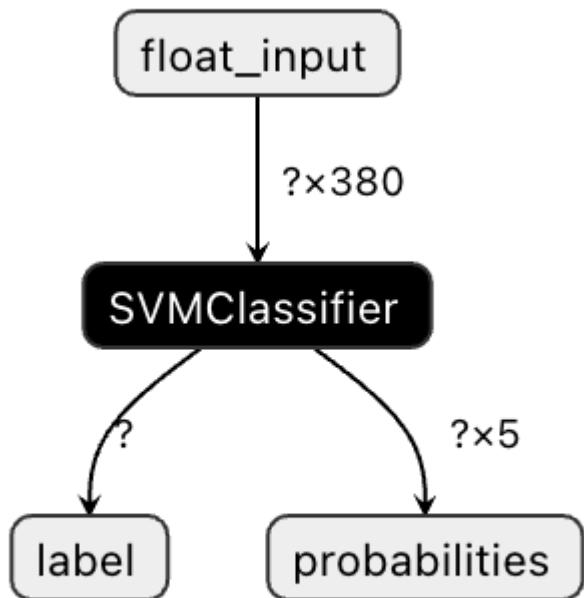
Note, you can pass in options in your conversion script. In this case, we passed in 'nocl' to be True and 'zipmap' to be False. Since this is a classification model, you have the option to remove ZipMap which produces a list of dictionaries (not necessary). `nocl` refers to class information being included in the model. Reduce your model's size by setting `nocl` to 'True'.

Running the entire notebook will now build an Onnx model and save it to this folder.

## View your model

---

Onnx models are not very visible in Visual Studio code, but there's a very good free software that many researchers use to visualize the model to ensure that it is properly built. Download [Netron](#) and open your `model.onnx` file. You can see your simple model visualized, with its 380 inputs and classifier listed:



Netron is a helpful tool to view your models.

Now you are ready to use this neat model in a web app. Let's build an app that will come in handy when you look in your refrigerator and try to figure out which combination of your leftover ingredients you can use to cook a given cuisine, as determined by your model.

## Build a recommender web application

---

You can use your model directly in a web app. This architecture also allows you to run it locally and even offline if needed. Start by creating an `index.html` file in the same folder where you stored your `model.onnx` file.

In this file, add the following markup:

```

html
<!DOCTYPE html>
<html>
  <header>
    <title>Cuisine Matcher</title>
  </header>
  <body>
    ...
  </body>
</html>

```

Now, working within the `body` tags, add a little markup to show a list of checkboxes reflecting some ingredients:

html

```
<h1>Check your refrigerator. What can you create?</h1>
<div id="wrapper">
  <div class="boxCont">
    <input type="checkbox" value="4" class="checkbox">
    <label>apple</label>
  </div>

  <div class="boxCont">
    <input type="checkbox" value="247" class="checkbox">
    <label>pear</label>
  </div>

  <div class="boxCont">
    <input type="checkbox" value="77" class="checkbox">
    <label>cherry</label>
  </div>

  <div class="boxCont">
    <input type="checkbox" value="126" class="checkbox">
    <label>fenugreek</label>
  </div>

  <div class="boxCont">
    <input type="checkbox" value="302" class="checkbox">
    <label>sake</label>
  </div>

  <div class="boxCont">
    <input type="checkbox" value="327" class="checkbox">
    <label>soy sauce</label>
  </div>

  <div class="boxCont">
    <input type="checkbox" value="112" class="checkbox">
    <label>cumin</label>
  </div>
</div>
<div style="padding-top:10px">
  <button onClick="startInference()">What kind of cuisine can you
</div>
```

Notice that each checkbox is given a value. This reflects the index where the ingredient is found according to the dataset. Apple, for example, in this alphabetic list, occupies the fifth column, so its value is '4' since we start counting at 0. You can consult the [ingredients spreadsheet](#) to discover a given ingredient's index.

Continuing your work in the index.html file, add a script block where the model is called after the final closing `</div>`. First, import the [Onnx Runtime](#):

html

```
<script src="https://cdn.jsdelivr.net/npm/onnxruntime-web@1.8.0-dev.20210611">
```

Onnx Runtime is used to enable running your Onnx models across a wide range of hardware platforms, including optimizations and an API to use.

Once the Runtime is in place, you can call it:

## javascript

```
        console.log(ingredients)
    }
}

function testCheckboxes() {
    for (var i = 0; i < checks.length; i++)
        if (checks[i].type == "checkbox")
            if (checks[i].checked)
                return true;
    return false;
}

async function startInference() {

    let checked = testCheckboxes()

    if (checked) {

        try {
            // create a new session and load the model.

            const session = await ort.InferenceSession.create('./model.onnx');

            const input = new ort.Tensor(new Float32Array(ingredients));
            const feeds = { float_input: input };

            // feed inputs and run

            const results = await session.run(feeds);

            // read from results
            alert('You can enjoy ' + results.label.data[0] + ' cuisine!');

        } catch (e) {
            console.log(`failed to inference ONNX model: ${e}`);
        }
    }
    else alert("Please check an ingredient")

}
init();
```

```
</script>
```

In this code, there are several things happening:

1. You created an array of 380 possible values (1 or 0) to be set and sent to the model for inference, depending on whether an ingredient checkbox is checked.
2. You created an array of checkboxes and a way to determine whether they were checked in an `init` function that is called when the application starts. When a checkbox is checked, the `ingredients` array is altered to reflect the chosen ingredient.
3. You created a `testCheckboxes` function that checks whether any checkbox was checked.
4. You use that function when the button is pressed and, if any checkbox is checked, you start inference.
5. The inference routine includes:
  1. Setting up an asynchronous load of the model
  2. Creating a Tensor structure to send to the model
  3. Creating 'feeds' that reflects the `float_input` input that you created when training your model (you can use Netron to verify that name)
  4. Sending these 'feeds' to the model and waiting for a response

## Test your application

Open a terminal session in Visual Studio Code in the folder where your index.html file resides. Ensure that you have `[http-server]` (<https://www.npmjs.com/package/http-server>) installed globally, and type `http-server` at the prompt. A localhost should open and you can view your web app. Check what cuisine is recommended based on various ingredients:

The screenshot shows a web browser window with the URL `127.0.0.1:8081` in the address bar. A modal dialog box is open, displaying the message "127.0.0.1:8081 says" and "You can enjoy indian cuisine today!" with an "OK" button. The main content area of the page features a heading "Check your refrigerator." and a list of ingredients with checkboxes. The checked items are "fenugreek" and "cumin". At the bottom, there is a text input field with the placeholder "What kind of cuisine can you make?"

127.0.0.1:8081 says  
You can enjoy indian cuisine today!

OK

Check your refrigerator.

apple  
 pear  
 cherry  
 fenugreek  
 sake  
 soy sauce  
 cumin

What kind of cuisine can you make?

Congratulations, you have created a simple web app recommendation with a few fields. Take some time to build out this system!

## Challenge

---

Your web app is very minimal, so continue to build it out using ingredients and their indexes from the [ingredient\\_indexes](#) data. What flavor combinations work to create a given national dish?

## Post-lecture quiz

---

## Review & Self Study

---

While this lesson just touched on the utility of creating a recommendation system for food ingredients, this area of ML applications is very rich in examples. Read some more about how these systems are built:

- <https://www.sciencedirect.com/topics/computer-science/recommendation-engine>
- <https://www.technologyreview.com/2014/08/25/171547/the-ultimate-challenge-for-recommendation-engines/>
- <https://www.technologyreview.com/2015/03/23/168831/everything-is-a-recommendation/>

## Assignment

---

[Build a new recommender](#)

## Introduction to Clustering

Clustering is a type of [Unsupervised Learning](#) that presumes that a dataset is unlabelled. It uses various algorithms to sort through unlabeled data and provide groupings according to patterns it discerns in the data.



🎥 Click the image above for a video. While you're studying Machine Learning with Clustering, enjoy some Nigerian Dance Hall tracks - this is a highly rated song from 2014 by PSquare.

## Pre-lecture quiz

---

### Introduction

Clustering is very useful for data exploration. Let's see if it can help discover trends and patterns in the way Nigerian audiences consume music.

✓ Take a minute to think about the uses of clustering. In real life, clustering happens whenever you have a pile of laundry and need to sort out your family members' clothes 🧺 👕 🎖️ 🍃 . In data science, clustering happens when trying to analyze a user's preferences, or determine the characteristics of any unlabeled dataset. Clustering, in a way, helps make sense of chaos.



🎥 Click the image above for a video: MIT's John Guttag introduces Clustering

In a professional setting, clustering can be used to determine things like market segmentation, determining what age groups buy what items, for example. Another use would be anomaly detection, perhaps to detect fraud from a dataset of credit card transactions. Or you might use clustering to determine tumors in a batch of medical scans.

✓ Think a minute about how you might have encountered clustering 'in the wild', in a banking, e-commerce, or business setting.

🎓 Interestingly, Cluster Analysis originated in the fields of Anthropology and Psychology in the 1930s. Can you imagine how it might have been used?

Alternately, you could use it for grouping search results - by shopping links, images, or reviews, for example. Clustering is useful when you have a large dataset that you want to reduce and on which you want to perform more granular analysis, so the technique can be used to learn about data before other models are constructed.

✓ Once your data is organized in clusters, you assign it a cluster Id, and this technique can be useful when preserving a dataset's privacy; you can instead refer to a data point by its cluster id, rather than by more revealing identifiable data. Can you think of other reasons why you'd refer to a cluster Id rather than other elements of the cluster to identify it?

# Getting started with clustering

Scikit-Learn offers a large array of methods to perform clustering. The type you choose will depend on your use case. According to the documentation, each method has various benefits. Here is a simplified table of the methods supported by Scikit-Learn and their appropriate use cases:

Method name	Use case
K-Means	general purpose, inductive
Affinity propagation	many, uneven clusters, inductive
Mean-shift	many, uneven clusters, inductive
Spectral clustering	few, even clusters, transductive
Ward hierarchical clustering	many, constrained clusters, transductive
Agglomerative clustering	many, constrained, non Euclidean distances, transductive
DBSCAN	non-flat geometry, uneven clusters, transductive
OPTICS	non-flat geometry, uneven clusters with variable density, transductive
Gaussian mixtures	flat geometry, inductive
BIRCH	large dataset with outliers, inductive

🎓 How we create clusters has a lot to do with how we gather up the data points into groups. Let's unpack some vocabulary:

🎓 'Transductive' vs. 'inductive'

Transductive inference is derived from observed training cases that map to specific test cases. Inductive inference is derived from training cases that map to general rules which are

only then applied to test cases.

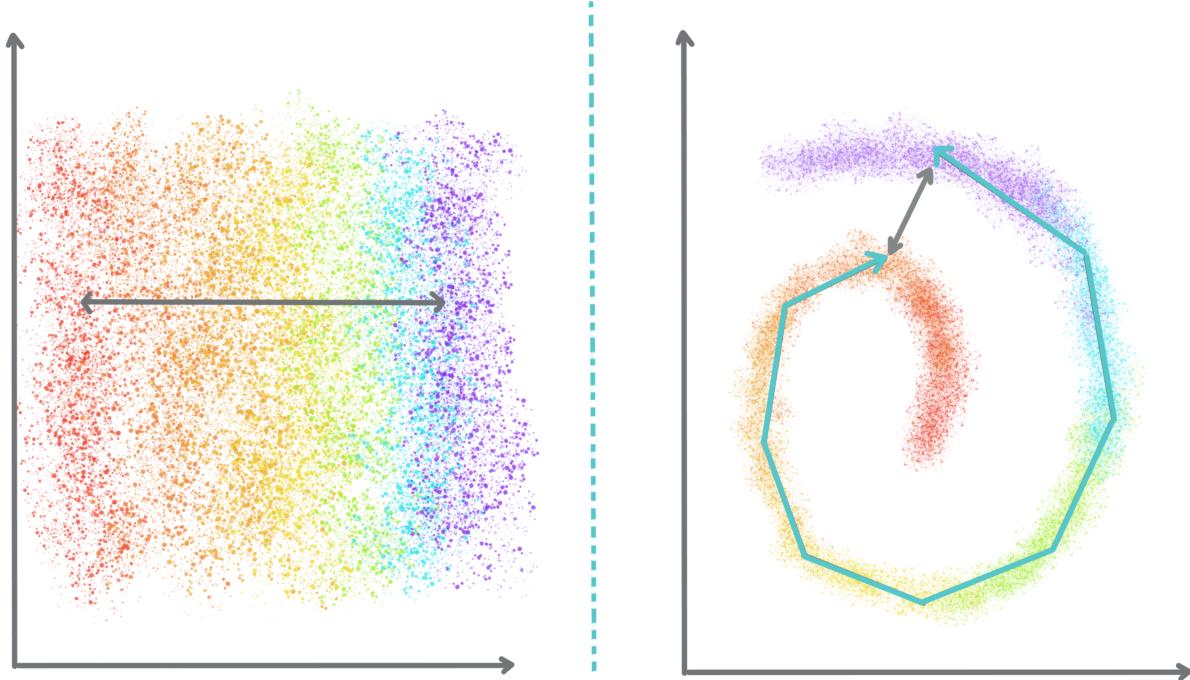
An example: Imagine you have a dataset that is only partially labelled. Some things are 'records', some 'cds', and some are blank. Your job is to provide labels for the blanks. If you choose an inductive approach, you'd train a model looking for 'records' and 'cds', and apply those labels to your unlabeled data. This approach will have trouble classifying things that are actually 'cassettes'. A transductive approach, on the other hand, handles this unknown data more effectively as it works to group similar items together and then applies a label to a group. In this case, clusters might reflect 'round musical things' and 'square musical things'.

### 'Non-flat' vs. 'flat' geometry

Derived from mathematical terminology, non-flat vs. flat geometry refers to the measure of distances between points by either 'flat' ([Euclidean](#)) or 'non-flat' (non-Euclidean) geometrical methods.

'Flat' in this context refers to Euclidean geometry (parts of which are taught as 'plane' geometry), and non-flat refers to non-Euclidean geometry. What does geometry have to do with machine learning? Well, as two fields that are rooted in mathematics, there must be a common way to measure distances between points in clusters, and that can be done in a 'flat' or 'non-flat' way, depending on the nature of the data. [Euclidean distances](#) are measured as the length of a line segment between two points. [Non-Euclidean distances](#) are measured along a curve. If your data, visualized, seems to not exist on a plane, you might need to use a specialized algorithm to handle it.

# FLAT v.s NONFLAT GEOMETRY



→ distance is measured by flat/euclidian geometry

→ distance is measured by non-flat geometry

@DASANI\_DECODED

Infographic by [Dasani Madipalli](#)

## 🎓 'Distances'

Clusters are defined by their distance matrix, e.g. the distances between points. This distance can be measured a few ways. Euclidean clusters are defined by the average of the point values, and contain a 'centroid' or center point. Distances are thus measured by the distance to that centroid. Non-Euclidean distances refer to 'clustroids', the point closest to other points. Clustroids in turn can be defined in various ways.

## 🎓 'Constrained'

Constrained Clustering introduces 'semi-supervised' learning into this unsupervised method. The relationships between points are flagged as 'cannot link' or 'must-link' so some rules are forced on the dataset.

An example: If an algorithm is set free on a batch of unlabelled or semi-labelled data, the clusters it produces may be of poor quality. In the example above, the clusters might group 'round music things' and 'square music things' and 'triangular things' and 'cookies'. If given some constraints, or rules to follow ("the item must be made of plastic", "the item needs to be able to produce music") this can help 'constrain' the algorithm to make better choices.

## 🎓 'Density'

Data that is 'noisy' is considered to be 'dense'. The distances between points in each of its clusters may prove, on examination, to be more or less dense, or 'crowded' and thus this data needs to be analyzed with the appropriate clustering method. [This article](#) demonstrates the difference between using K-Means clustering vs. HDBSCAN algorithms to explore a noisy dataset with uneven cluster density.

## Clustering Algorithms

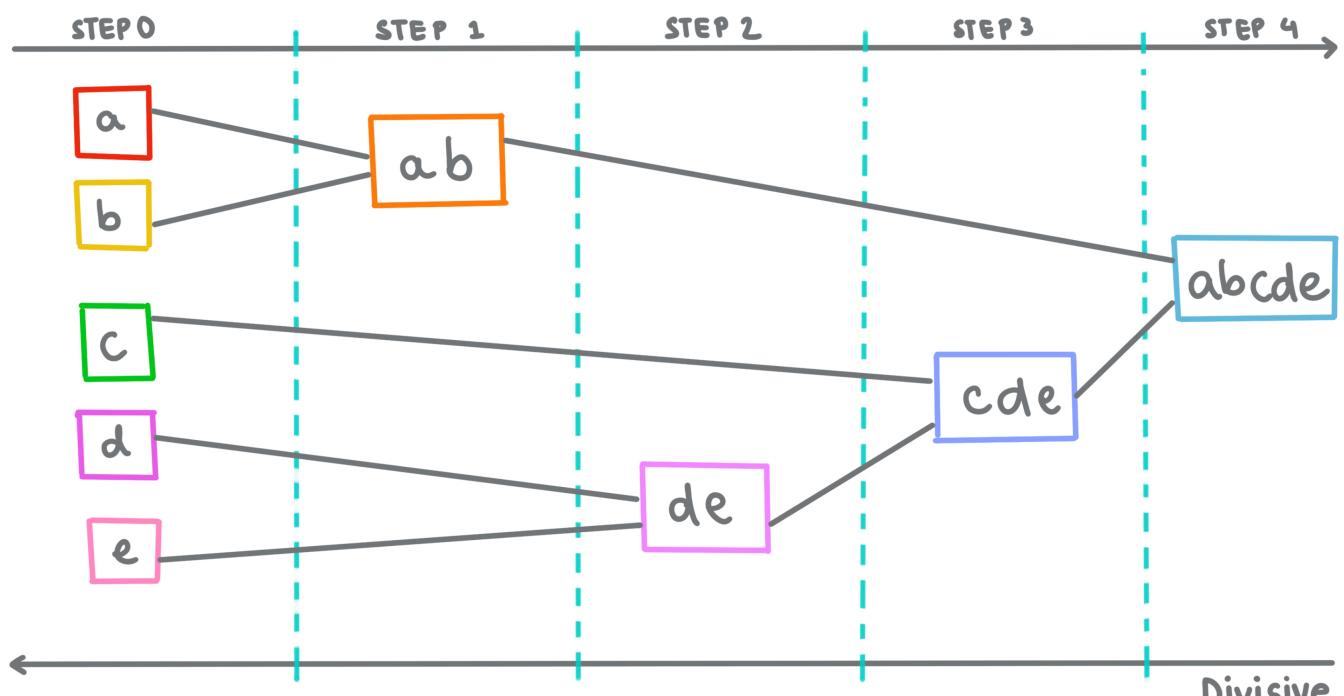
There are over 100 clustering algorithms, and their use depends on the nature of the data at hand. Let's discuss some of the major ones:

### Hierarchical clustering

If an object is classified by its proximity to a nearby object, rather than to one farther away, clusters are formed based on their members' distance to and from other objects. Scikit-Learn's Agglomerative clustering is hierarchical.

## HIERARCHICAL CLUSTERING

### Agglomerative

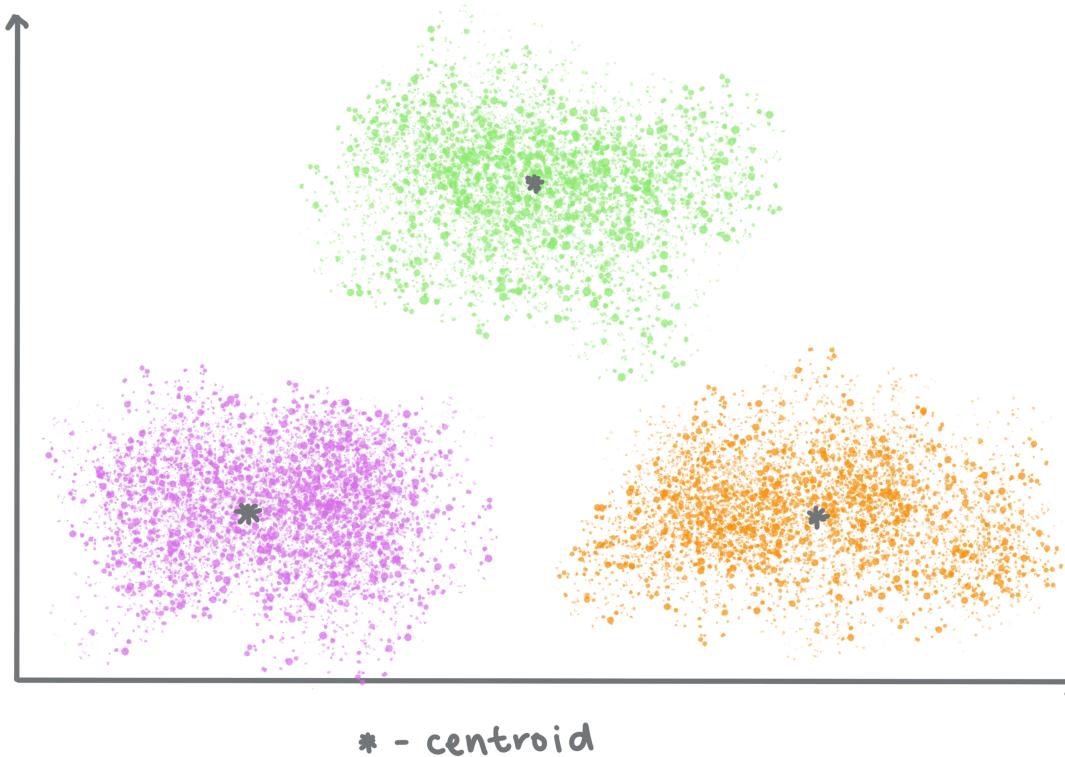


@DASANI\_DECODED

## Centroid clustering

This popular algorithm requires the choice of 'k', or the number of clusters to form, after which the algorithm determines the center point of a cluster and gathers data around that point. K-means clustering is a popular version of centroid clustering. The center is determined by the nearest mean, thus the name. The squared distance from the cluster is minimized.

## CENTROID CLUSTERING



Infographic by [Dasani Madipalli](#)

## Distribution-based clustering

Based in statistical modeling, distribution-based clustering centers on determining the probability that a data point belongs to a cluster, and assigning it accordingly. Gaussian Mixture methods belong to this type.

## Density-based clustering

Data points are assigned to clusters based on their density, or their grouping around each other. Data points far from the group are considered outliers or noise. DBSCAN, Mean-shift and OPTICS belong to this type of clustering.

## Grid-based clustering

For multi-dimensional datasets, a grid is created and the data is divided amongst the grid's cells, thereby creating clusters.

## Preparing the data

Clustering as a technique is greatly aided by proper visualization, so let's get started by visualizing our music data. This exercise will help us decide which of the methods of clustering we should most effectively use for the nature of this data.

Open the notebook.ipynb file in this folder. Import the Seaborn package for good data visualization.

python

```
pip install seaborn
```

Append the song data .csv file. Load up a dataframe with some data about the songs. Get ready to explore this data by importing the libraries and dumping out the data:

python

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("../data/nigerian-songs.csv")
df.head()
```

Check the first few lines of data:

	name	album	artist	artist_top_genre	release_date	length	popularity
0	Sparky	Mandy & The Jungle	Cruel Santino	alternative r&b	2019	144000	48
1	shuga rush	EVERYTHING YOU HEARD IS TRUE	Odunsi (The Engine)	afropop	2020	89488	30
2	LITT!	LITT!	AYLØ	indie r&b	2018	207758	40
3	Confident / Feeling Cool	Enjoy Your Life	Lady Donli	nigerian pop	2019	175135	14

	name	album	artist	artist_top_genre	release_date	length	popularity
4	wanted you	rare.	Odunsi (The Engine)	afropop	2018	152049	25

Get some information about the dataframe:

python

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 530 entries, 0 to 529
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   name            530 non-null    object 
 1   album           530 non-null    object 
 2   artist          530 non-null    object 
 3   artist_top_genre 530 non-null    object 
 4   release_date    530 non-null    int64  
 5   length          530 non-null    int64  
 6   popularity       530 non-null    int64  
 7   danceability     530 non-null    float64
 8   acousticness     530 non-null    float64
 9   energy           530 non-null    float64
 10  instrumentalness 530 non-null    float64
 11  liveness         530 non-null    float64
 12  loudness         530 non-null    float64
 13  speechiness      530 non-null    float64
 14  tempo            530 non-null    float64
 15  time_signature   530 non-null    int64  
dtypes: float64(8), int64(4), object(4)
memory usage: 66.4+ KB
```

Double-check for null values:

python

```
df.isnull().sum()
```

Looking good:

```
name          0  
album         0  
artist        0  
artist_top_genre 0  
release_date   0  
length         0  
popularity     0  
danceability    0  
acousticness    0  
energy          0  
instrumentalness 0  
liveness        0  
loudness         0  
speechiness      0  
tempo            0  
time_signature   0  
dtype: int64
```

Describe the data:

python

```
df.describe()
```

	release_date	length	popularity	danceability	acousticness	energy	inst1
count	530	530	530	530	530	530	530
mean	2015.390566	222298.1698	17.507547	0.741619	0.265412	0.760623	0.01
std	3.131688	39696.82226	18.992212	0.117522	0.208342	0.148533	0.01
min	1998	89488	0	0.255	0.000665	0.111	0
25%	2014	199305	0	0.681	0.089525	0.669	0
50%	2016	218509	13	0.761	0.2205	0.7845	0.01
75%	2017	242098.5	31	0.8295	0.403	0.87575	0.01
max	2020	511738	73	0.966	0.954	0.995	0.91

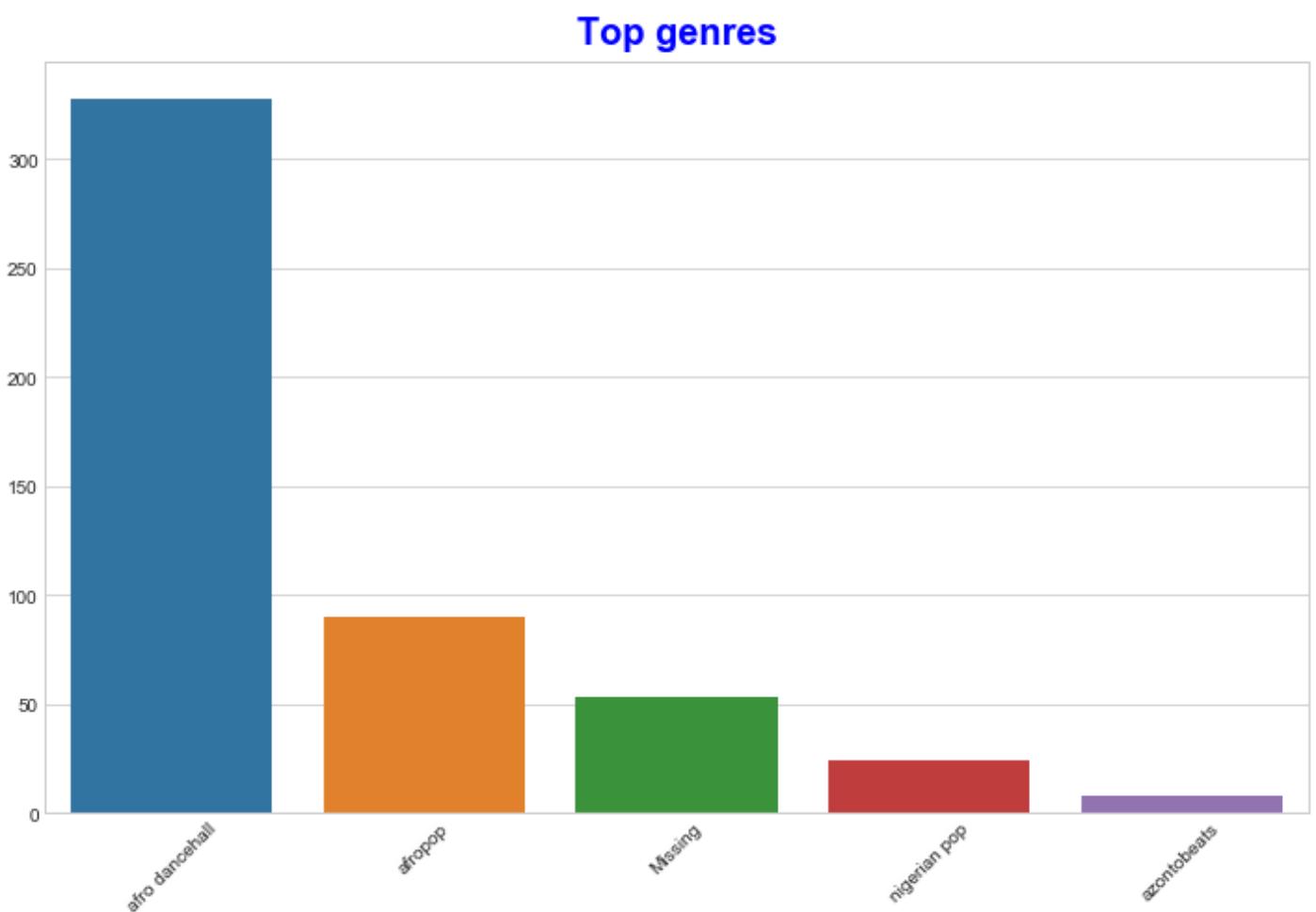
Look at the general values of the data. Note that popularity can be '0', which show songs that have no ranking. Let's remove those shortly.

Use a barplot to find out the most popular genres:

python

```
import seaborn as sns

top = df['artist_top_genre'].value_counts()
plt.figure(figsize=(10,7))
sns.barplot(x=top[:5].index,y=top[:5].values)
plt.xticks(rotation=45)
plt.title('Top genres',color = 'blue')
```



- If you'd like to see more top values, change the top `[:5]` to a bigger value, or remove it to see all.

Note, when the top genre is described as 'Missing', that means that Spotify did not classify it, so let's get rid of it:

python

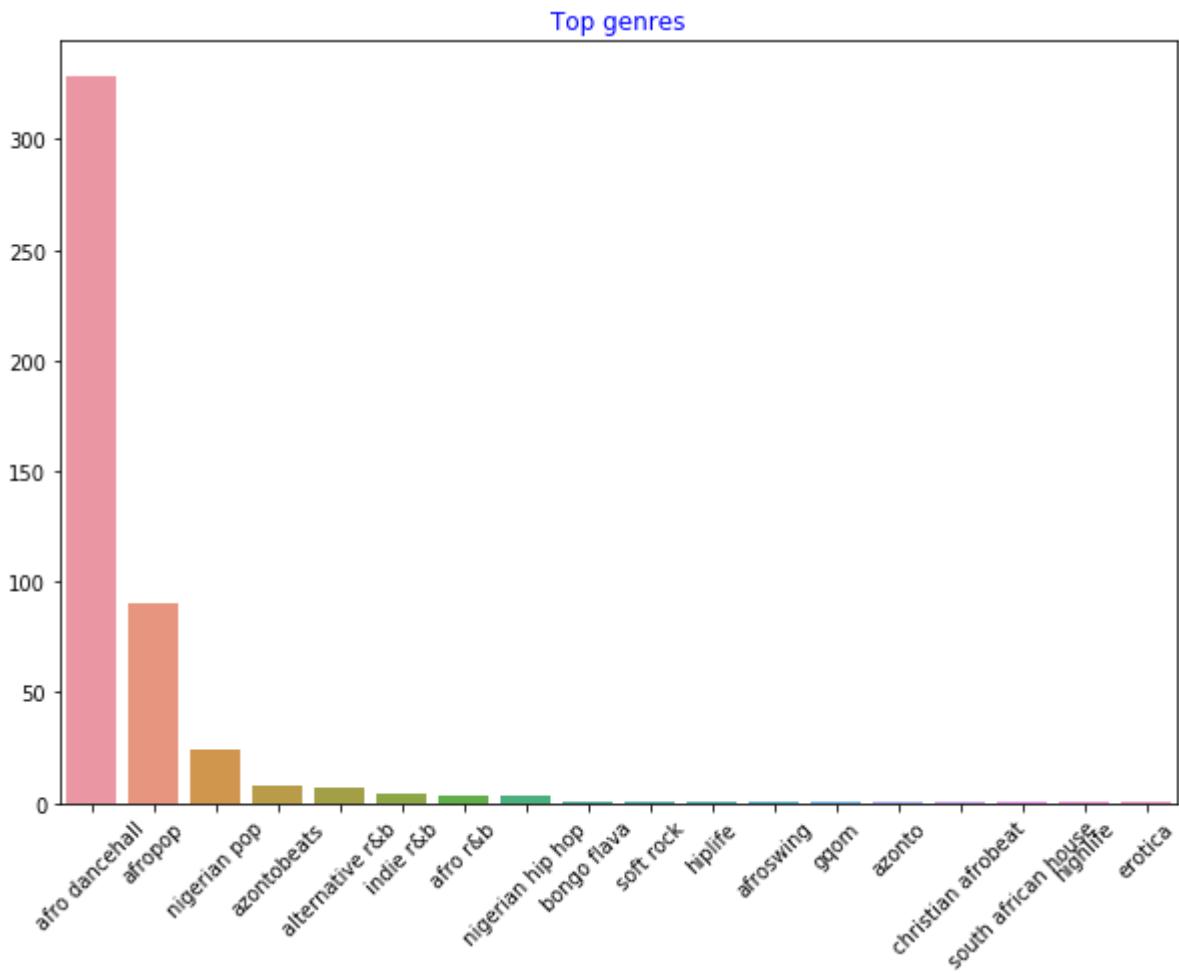
```
df = df[df['artist_top_genre'] != 'Missing']
top = df['artist_top_genre'].value_counts()
plt.figure(figsize=(10,7))
```

```

sns.barplot(x=top.index,y=top.values)
plt.xticks(rotation=45)
plt.title('Top genres',color = 'blue')

```

Now recheck the genres:



By far, the top three genres dominate this dataset, so let's concentrate on `afro dancehall`, `afropop`, and `nigerian pop`, also filtering the dataset to remove anything with a 0 popularity value (meaning it was not classified with a popularity in the dataset and can be considered noise for our purposes):

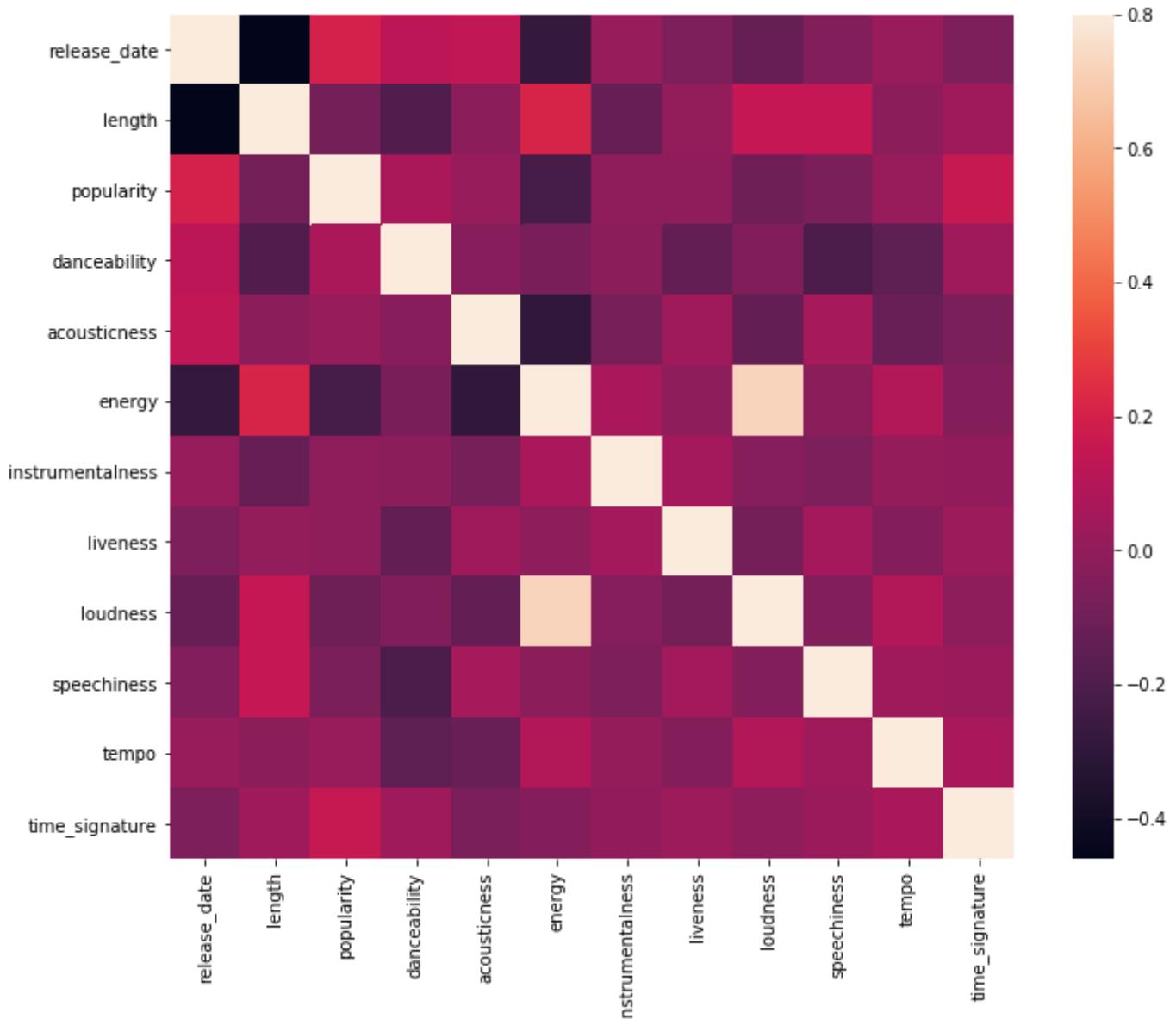
```

python
df = df[(df['artist_top_genre'] == 'afro dancehall') | (df['artist_top_gen
df = df[(df['popularity'] > 0)]
top = df['artist_top_genre'].value_counts()
plt.figure(figsize=(10,7))
sns.barplot(x=top.index,y=top.values)
plt.xticks(rotation=45)
plt.title('Top genres',color = 'blue')

```

Do a quick test to see if the data correlates in any particularly strong way:

```
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



The only strong correlation is between energy and loudness, which is not too surprising, given that loud music is usually pretty energetic. Otherwise, the correlations are relatively weak. It will be interesting to see what a clustering algorithm can make of this data.

 Note that correlation does not imply causation! We have proof of correlation but no proof of causation. An [amusing web site](#) has some visuals that emphasize this point.

Is there any convergence in this dataset around a song's perceived popularity and danceability? A FacetGrid shows that there are concentric circles that line up, regardless of genre. Could it be that Nigerian tastes converge at a certain level of danceability for this genre?

Try different datapoints (energy, loudness, speechiness) and more or different musical genres. What can you discover? Take a look at the `df.describe()` table to see the general spread of the data points.

## Data Distribution

Are these three genres significantly different in the perception of their danceability, based on their popularity? Examine our top three genres data distribution for popularity and danceability along a given x and y axis.

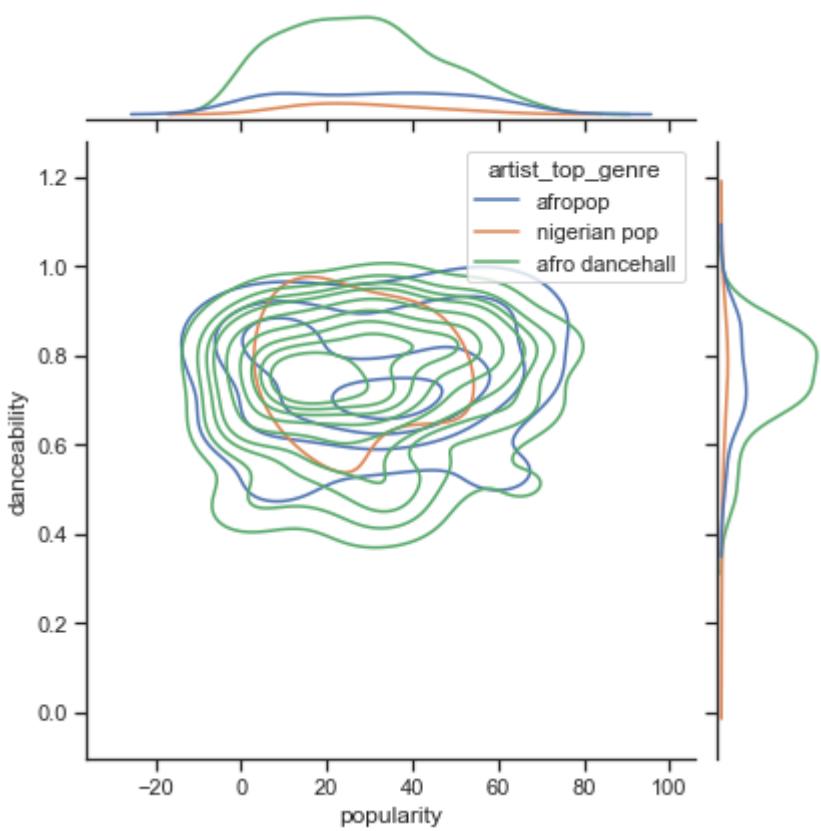
python

```
sns.set_theme(style="ticks")  
  
g = sns.jointplot(  
    data=df,  
    x="popularity", y="danceability", hue="artist_top_genre",  
    kind="kde",  
)
```

You can discover concentric circles around a general point of convergence, showing the distribution of points.

 Note that this example uses a KDE (Kernel Density Estimate) graph that represents the data using a continuous probability density curve. This allows us to interpret data when working with multiple distributions.

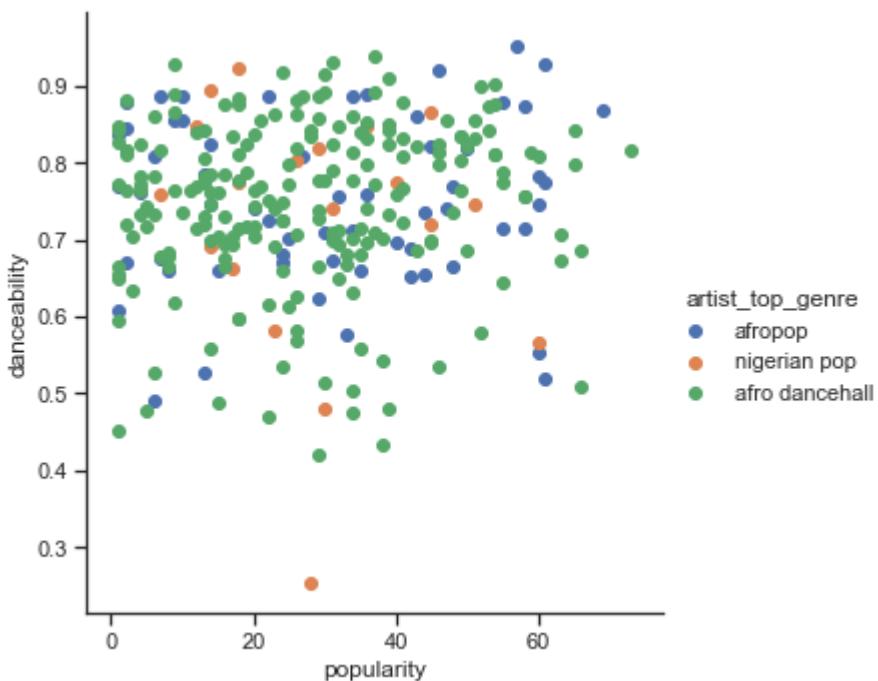
In general, the three genres align loosely in terms of their popularity and danceability. Determining clusters in this loosely-aligned data will be interesting:



A scatterplot of the same axes shows a similar pattern of convergence:

python

```
sns.FacetGrid(df, hue="artist_top_genre", size=5) \
    .map(plt.scatter, "popularity", "danceability") \
    .add_legend()
```



In general, for clustering, you can use scatterplots to show clusters of data, so mastering this type of visualization is very useful. In the next lesson, we will take this filtered data and use k-means clustering to discover groups in this data that seems to overlap in interesting ways.

---

In preparation for the next lesson, make a chart about the various clustering algorithms you might discover and use in a production environment. What kinds of problems is the clustering trying to address?

## Post-lecture quiz

---

## Review & Self Study

---

Before you apply clustering algorithms, as we have learned, it's a good idea to understand the nature of your dataset. Read more on this topic [here](#)

[This helpful article](#) walks you through the different ways that various clustering algorithms behave, given different data shapes.

## Assignment

---

[Research other visualizations for clustering](#)

## K-Means Clustering



Click the image above for a video: Andrew Ng explains Clustering

## Pre-lecture quiz

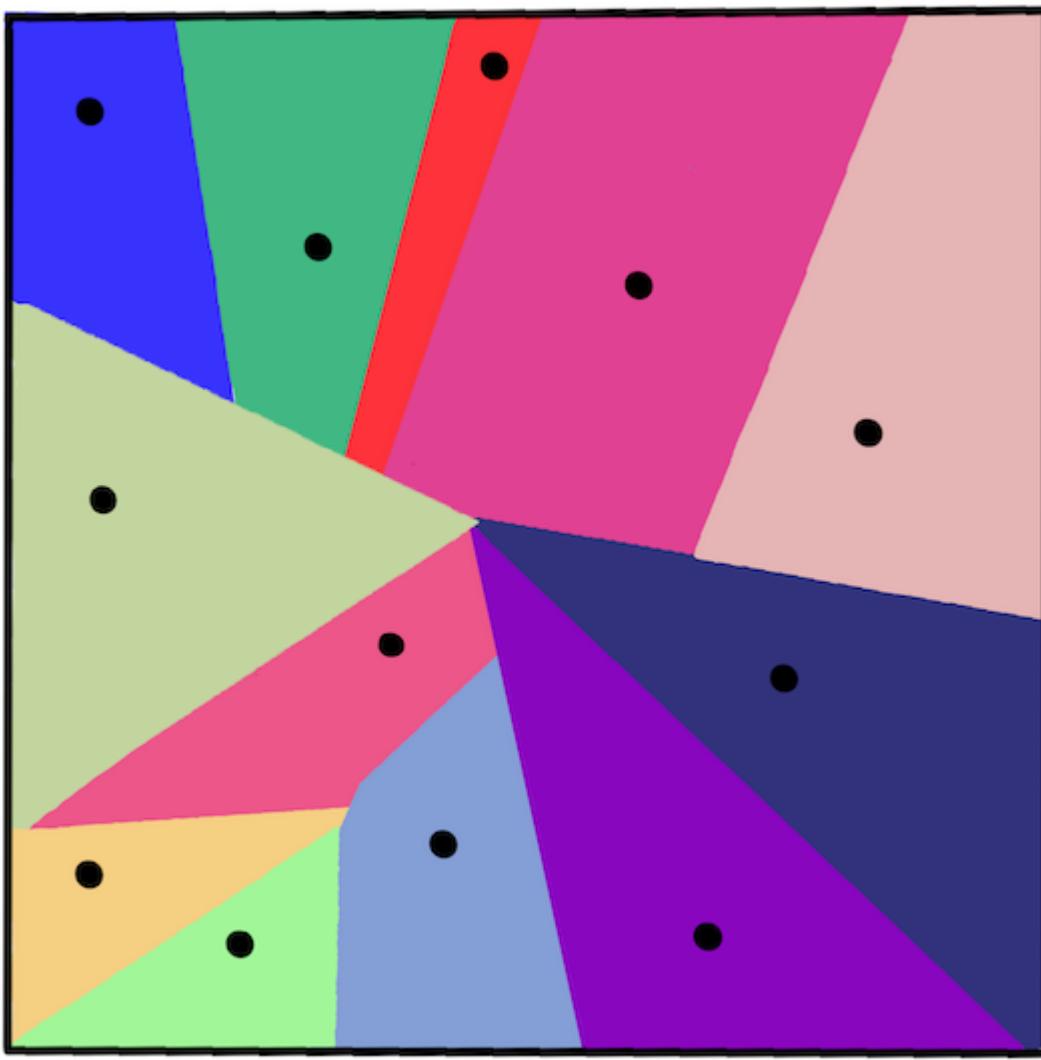
In this lesson, you will learn how to create clusters using Scikit-Learn and the Nigerian music dataset you imported earlier. We will cover the basics of K-Means for Clustering. Keep in mind that, as you learned in the earlier lesson, there are many ways to work with clusters and the method you use depends on your data. We will try K-Means as it's the most common Clustering technique. Let's get started!

Terms you will learn about:

- Silhouette Scoring
- Elbow Method
- Inertia
- Variance

### **Introduction**

K-Means Clustering is a method derived from the domain of signal processing. It is used to divide and partition groups of data into 'k' clusters using a series of observations. Each observation works to group a given datapoint closest to its nearest 'mean', or the center point of a cluster. The clusters can be visualized as Voronoi diagrams, which include a point (or 'seed') and its corresponding region.



infographic by [Jen Looper](#)

The K-Means Clustering process executes in a three-step process:

1. The algorithm selects k-number of center points by sampling from the dataset. After this, it loops:
  1. It assigns each sample to the nearest centroid
  2. It creates new centroids by taking the mean value of all of the samples assigned to the previous centroids.
  3. Then, it calculates the difference between the new and old centroids and repeats until the centroids are stabilized.

One drawback of using K-Means includes the fact that you will need to establish 'k', that is the number of centroids. Fortunately the 'elbow method' helps to estimate a good starting value for 'k'. You'll try it in a minute.

## Prerequisite

You will work in this lesson's `notebook.ipynb` file that includes the data import and preliminary cleaning you did in the last lesson.

## Preparation

Start by taking another look at the songs data. This data is a little noisy: by observing each column as a boxplot, you can see outliers:

python

```
plt.figure(figsize=(20,20), dpi=200)

plt.subplot(4,3,1)
sns.boxplot(x = 'popularity', data = df)

plt.subplot(4,3,2)
sns.boxplot(x = 'acousticness', data = df)

plt.subplot(4,3,3)
sns.boxplot(x = 'energy', data = df)

plt.subplot(4,3,4)
sns.boxplot(x = 'instrumentalness', data = df)

plt.subplot(4,3,5)
sns.boxplot(x = 'liveness', data = df)

plt.subplot(4,3,6)
sns.boxplot(x = 'loudness', data = df)

plt.subplot(4,3,7)
sns.boxplot(x = 'speechiness', data = df)

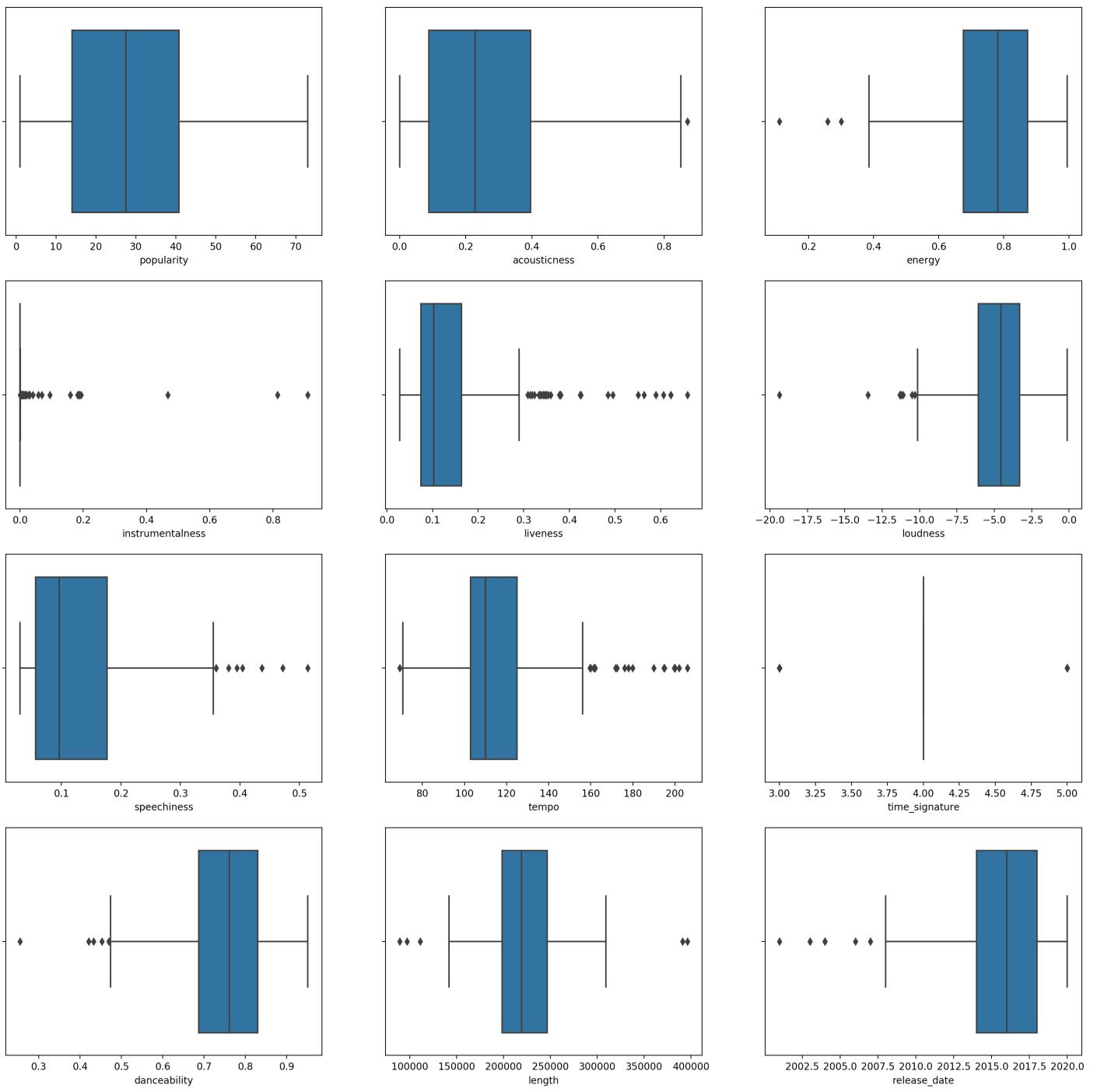
plt.subplot(4,3,8)
sns.boxplot(x = 'tempo', data = df)

plt.subplot(4,3,9)
sns.boxplot(x = 'time_signature', data = df)

plt.subplot(4,3,10)
sns.boxplot(x = 'danceability', data = df)

plt.subplot(4,3,11)
sns.boxplot(x = 'length', data = df)
```

```
plt.subplot(4,3,12)
sns.boxplot(x = 'release_date', data = df)
```



You could go through the dataset and remove these outliers, but that would make the data pretty minimal. For now, choose which columns you will use for your clustering exercise. Pick ones with similar ranges and encode the `artist_top_genre` column as numeric data:

python

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

X = df.loc[:, ('artist_top_genre','popularity','danceability','acousticness',
               'energy','liveness','loudness','speechiness','tempo','time_signature',
               'length','release_date')]

y = df['artist_top_genre']
```

```
X['artist_top_genre'] = le.fit_transform(X['artist_top_genre'])  
  
y = le.transform(y)
```

Now you need to pick how many clusters to target. You know there are 3 song genres that we carved out of the dataset, so let's try 3:

python

```
from sklearn.cluster import KMeans  
  
nclusters = 3  
seed = 0  
  
km = KMeans(n_clusters=nclusters, random_state=seed)  
km.fit(X)  
  
# Predict the cluster for each data point  
  
y_cluster_kmeans = km.predict(X)  
y_cluster_kmeans
```

You see an array printed out with predicted clusters (0, 1, or 2) for each row of the dataframe.

Use this array to calculate a 'silhouette score':

python

```
from sklearn import metrics  
score = metrics.silhouette_score(X, y_cluster_kmeans)  
score
```

## Silhouette score

Look for a silhouette score closer to 1. This score varies from -1 to 1, and if the score is 1, the cluster is dense and well-separated from other clusters. A value near 0 represents overlapping clusters with samples very close to the decision boundary of the neighboring clusters.[source](#).

Our score is .53, so right in the middle. This indicates that our data is not particularly well-suited to this type of clustering, but let's continue.

## Build a model

Now you can import KMeans and start the clustering process. There are a few parts here that warrant explaining:

python

```
from sklearn.cluster import KMeans  
wcss = []  
  
for i in range(1, 11):  
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)  
    kmeans.fit(X)  
    wcss.append(kmeans.inertia_)
```

👉 range: These are the iterations of the clustering process

👉 random\_state: "Determines random number generation for centroid initialization."[source](#)

👉 WCSS: "within-cluster sums of squares" measures the squared average distance of all the points within a cluster to the cluster centroid.[source](#).

👉 Inertia: K-Means algorithms attempt to choose centroids to minimize 'inertia', "a measure of how internally coherent clusters are."[source](#). The value is appended to the wcss variable on each iteration.

👉 k-means++: In [Scikit-Learn](#) you can use the 'k-means++' optimization, which "initializes the centroids to be (generally) distant from each other, leading to probably better results than random initialization."

## Elbow method

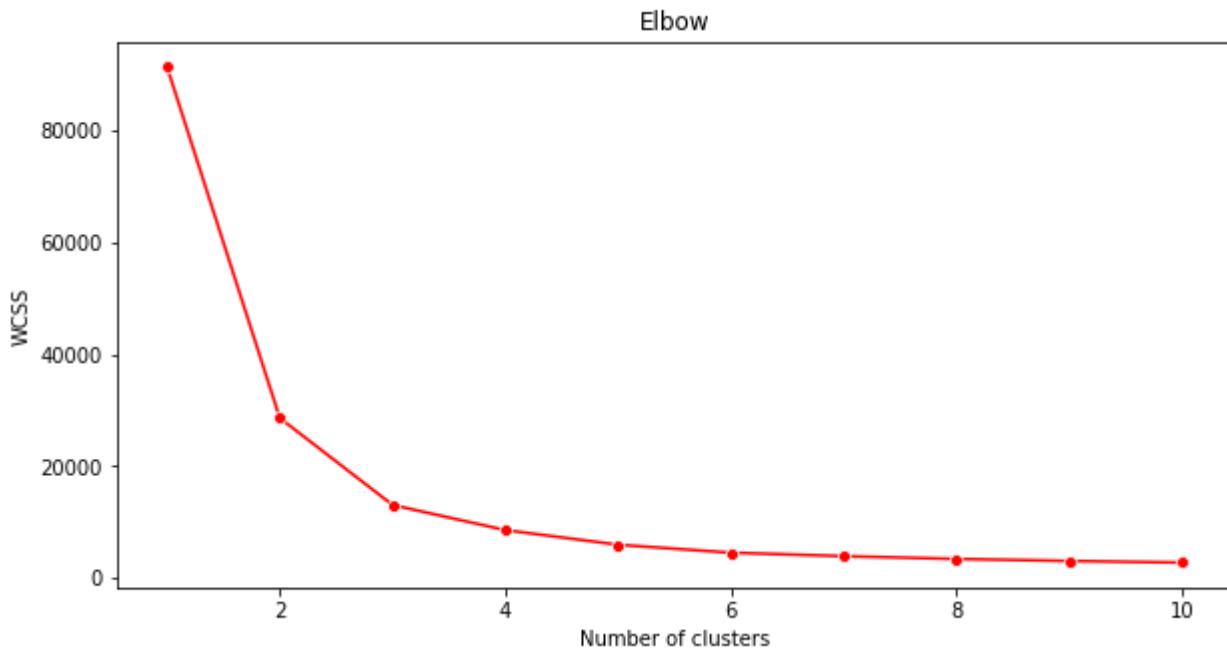
Previously, you surmised that, because you have targeted 3 song genres, you should choose 3 clusters. But is that the case? Use the 'elbow method' to make sure.

```

plt.figure(figsize=(10,5))
sns.lineplot(range(1, 11), wcss,marker='o',color='red')
plt.title('Elbow')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

```

Use the `wcss` variable that you built in the previous step to create a chart showing where the 'bend' in the elbow is, which indicates the optimum number of clusters. Maybe it **is** 3!



## Display the Clusters

Try the process again, this time setting three clusters, and display the clusters as a scatterplot:

```

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3)
kmeans.fit(X)
labels = kmeans.predict(X)
plt.scatter(df['popularity'],df['danceability'],c = labels)
plt.xlabel('popularity')
plt.ylabel('danceability')
plt.show()

```

Check the model's accuracy:

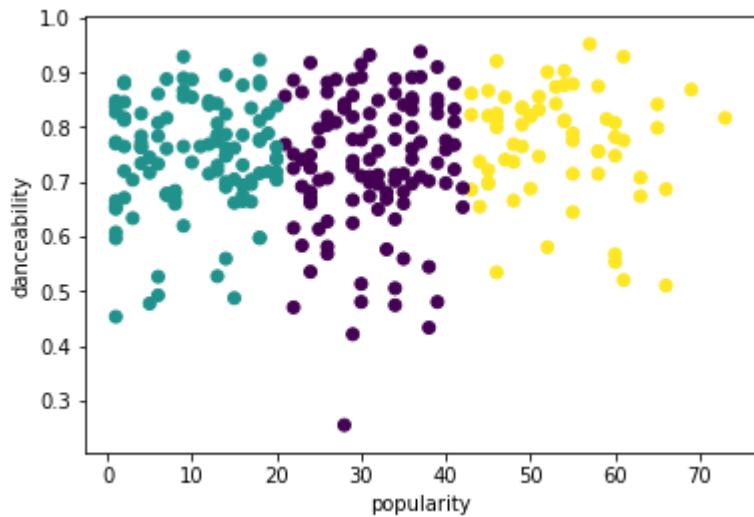
```
labels = kmeans.labels_

correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))

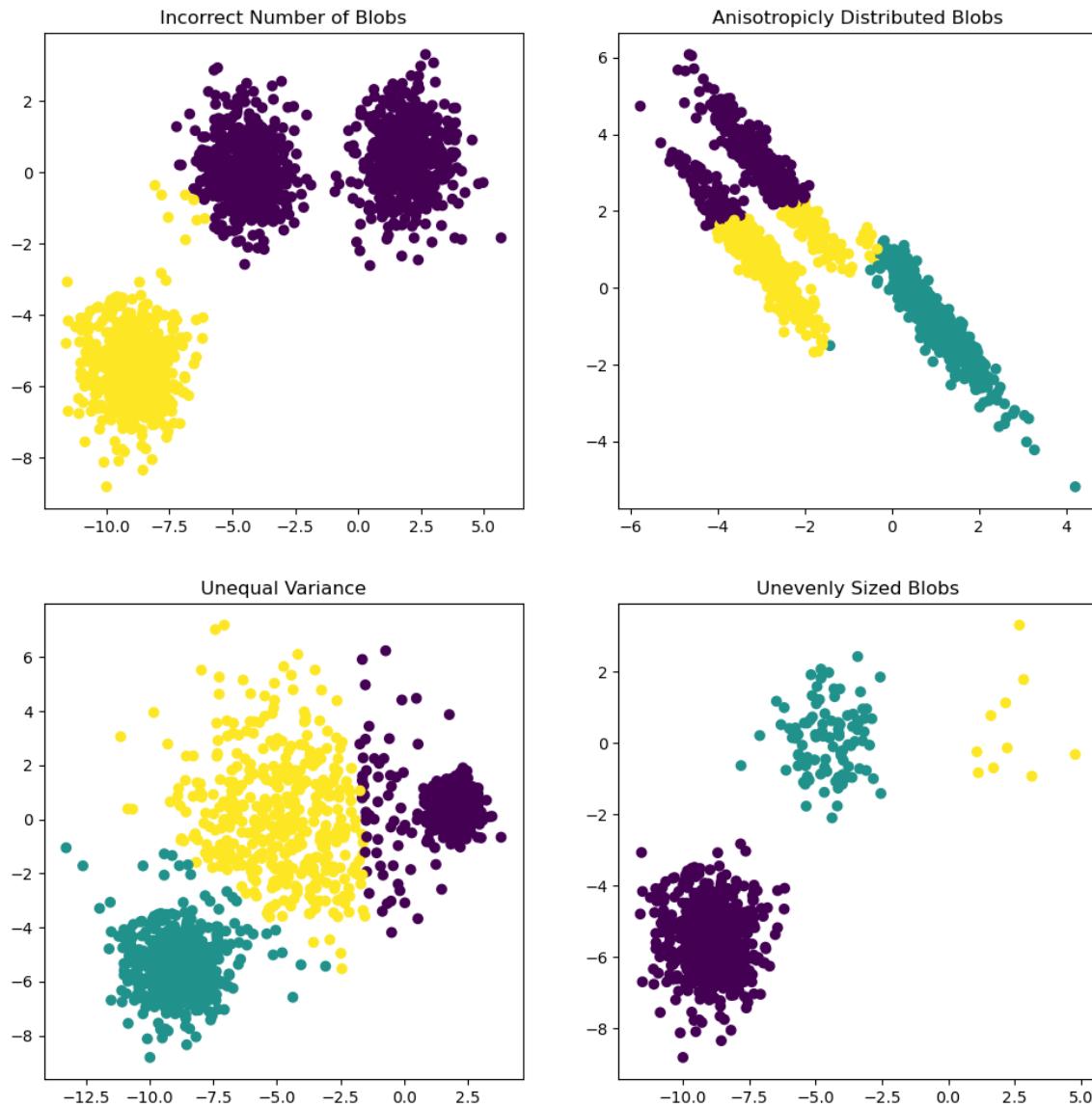
print('Accuracy score: {:.2f}'.format(correct_labels/float(y.size)))
```

This model's accuracy is not very good, and the shape of the clusters gives you a hint why.



This data is too imbalanced, too little correlated and there is too much variance between the column values, to cluster well. In fact, the clusters that form are probably heavily influenced or skewed by the three genre categories we defined above. That was a learning process!

In Scikit-Learn's documentation, you can see that a model like this one, with clusters not very well demarcated, has a 'variance' problem:



Infographic from Scikit-Learn

## Variance

Variance is defined as "the average of the squared differences from the Mean."[source](#) In the context of this clustering problem, it refers to data that the numbers of our dataset tend to diverge a bit too much from the mean.

✓ This is a great moment to think about all the ways you could correct this issue. Tweak the data a bit more? Use different columns? Use a different algorithm? Hint: Try [scaling your data](#) to normalize it

and test other columns.

Try this '[variance calculator](#)' to understand the concept a bit more.

## Challenge

---

Spend some time with this notebook, tweaking parameters. Can you improve the accuracy of the model by cleaning the data more (removing outliers, for example)? You can use weights to give more weight to given data samples. What else can you do to create better clusters?

Hint: Try to scale your data. There's commented code in the notebook that adds Standard Scaling to make the data columns resemble each other more closely in terms of range. You'll find that while the silhouette score goes down, the 'kink' in the elbow graph smooths out. This is because leaving the data unscaled allows data with less variance to carry more weight. Read a bit more on this problem [here](#).

## Post-lecture quiz

---

## Review & Self Study

---

Take a look at Stanford's K-Means Simulator [here](#). You can use this tool to visualize sample data points and determine its centroids. With fresh data, click 'update' to see how long it takes to find convergence. You can edit the data's randomness, numbers of clusters and numbers of centroids. Does this help you get an idea of how the data can be grouped?

Also, take a look at [this handout on k-means](#) from Stanford

## Assignment

---

[Try different clustering methods](#)

# Introduction to Natural Language Processing

This lesson covers a brief history and important concepts of *Computational Linguistics* focusing on *Natural Language Processing*.



## Pre-lecture quiz

---

## Introduction

---

NLP, as it is commonly known, is one of the best-known areas where machine learning has been applied and used in production software.

- Can you think of software that you use every day that probably has some NLP embedded? What about your word processing programs or mobile apps that you use regularly?

You will learn about how the ideas about languages developed and what the major areas of study have been. You will also learn definitions and concepts about how computers process text, including parsing, grammar, and identifying nouns and verbs. There are some coding tasks in this lesson, and several important concepts are introduced that you will learn to code later on in the next lessons.

Computational linguistics is an area of research and development over many decades that studies how computers can work with, and even understand, translate, and communicate with languages. Natural Language Processing (NLP) is a related field focused on how computers can process 'natural', or human, languages. If you have ever dictated to your phone instead of typing or asked a virtual assistant a question, your speech was converted into a text form and then processed or *parsed* from the language you spoke. The detected keywords were then processed into a format that the phone or assistant could understand and act on.

This is possible because someone wrote a computer program to do this. A few decades ago, some science fiction writers predicted that people would mostly speak to their computers, and the computers would always understand exactly what they meant. Sadly, it turned out to be a harder problem than many imagined, and while it is a much better understood problem today, there are significant challenges in achieving 'perfect' natural language processing when it comes to understanding the meaning of a sentence. This is a particularly hard problem when it comes to understanding humour or detecting emotions such as sarcasm in a sentence.

At this point, you may be remembering school classes where the teacher covered the parts of grammar in a sentence. In some countries, students are taught grammar and linguistics as a dedicated subject, but in many, these topics are included as part of learning a language: either your first language in primary school (learning to read and write) and perhaps a second language in post-primary, or high school. Don't worry if you are not an expert at differentiating nouns from verbs or adverbs from adjectives!

If you struggle with the difference between the *simple present* and *present progressive*, you are not alone. This is a challenging thing for many people, even native speakers of a language. The good news is that computers are really good at applying formal rules, and you will learn to write code that can *parse* a sentence as well as a human. The greater challenge you will examine later is understanding the *meaning*, and *sentiment*, of a sentence.

## Prerequisites

---

For this lesson, the main prerequisite is being able to read and understand the language of this lesson. There are no math problems or equations to solve. While the original author wrote this lesson in English, it is also translated into other languages, so you could be reading a translation. There are examples where a number of different languages are used (to compare the different grammar rules of different languages). These are *not* translated, but the explanatory text is, so the meaning should be clear.

For the coding tasks, you will use Python and the examples are using Python 3.8.

In this section, you will need:

- Python 3 programming language comprehension
  - this lesson uses input, loops, file reading, arrays
- Visual Studio Code with its Python extension
  - (*or the Python IDE of your choice*)
- TextBlob a simplified text processing library for Python
  - Follow the instructions on the TextBlob site to install it on your system (install the corpora as well, as shown below)

bash

```
pip install -U textblob
python -m textblob.download_corpora
```

 Tip: You can run Python directly in VS Code environments. Check the [docs](#) for more information.

## Conversing with Eliza

---

The history of trying to make computers understand human language goes back decades, and one of the earliest scientists to consider natural language processing was *Alan Turing*. When Turing was researching *Artificial Intelligence* in the 1950's, he considered if a conversational test could be given to a human and computer (via typed correspondence) where the human in the conversation was not sure if they were conversing with another human or a computer. If, after a certain length of conversation, the human could not determine that the answers were from a computer or not, then could the computer be said to be *thinking*?



The idea for this came from a party game called *The Imitation Game* where an interrogator is alone in a room and tasked with determining which of two people (in another room) are male and female respectively. The interrogator can send notes, and must try to think of questions where the written answers reveal the gender of the mystery person. Of course, the players in the other room are trying to trick the interrogator by answering questions in such a way as to mislead or confuse the interrogator, whilst also giving the appearance of answering honestly.

In the 1960's an MIT scientist called Joseph Weizenbaum developed *Eliza*, a computer 'therapist' that would ask the human questions and give the appearance of understanding their answers. However, while Eliza could parse a sentence and identify certain grammatical constructs and keywords so as to give a reasonable answer, it could not be said to *understand* the sentence. If Eliza was presented with a sentence following the format "I am sad" it might rearrange and substitute words in the sentence to form the response "How long have **you been sad**". This gave the impression that Eliza understood the statement and was asking a follow-on question, whereas in reality, it was changing the tense and adding some words. If Eliza could not identify a keyword that it had a response for, it would instead give a random response that could be applicable to many different statements. Eliza could be easily tricked, for instance if a user wrote "**You are a bicycle**" it might respond with "How long have **I been a bicycle?**", instead of a more reasoned response.

Note: You can read the original description of *Eliza* published in 1966 if you have an ACM account. Alternately, read about Eliza on [wikipedia](#)

## Task: Coding a basic conversational bot

A conversational bot, like Eliza, is a program that elicits user input and seems to understand and respond intelligently. Unlike Eliza, our bot will not have several rules giving it the appearance of having an intelligent conversation. Instead, our bot will have one ability only, to keep the conversation going with random responses that might work in almost any trivial conversation.

Your steps when building a conversational bot:

1. Print instructions advising the user how to interact with the bot
2. Start a loop
  1. Accept user input
  2. If user has asked to exit, then exit
  3. Process user input and determine response (in this case, the response is a random choice from a list of possible generic responses)
  4. Print response
3. loop back to step 2

Create this bot yourself in Python with the following random responses:

```
python
```

```
random_responses = ["That is quite interesting, please tell me more.",  
                    "I see. Do go on.",  
                    "Why do you say that?",  
                    "Funny weather we've been having, isn't it?",  
                    "Let's change the subject.",  
                    "Did you catch the game last night?"]
```

Here is some sample output to guide you (user input is on the lines starting with > ):

```
Hello, I am Marvin, the simple robot.  
You can end this conversation at any time by typing 'bye'  
After typing each answer, press 'enter'  
How are you today?  
> I am good thanks  
That is quite interesting, please tell me more.  
> today I went for a walk  
Did you catch the game last night?  
> I did, but my team lost  
Funny weather we've been having, isn't it?  
> yes but I hope next week is better  
Let's change the subject.  
> ok, lets talk about music  
Why do you say that?  
> because I like music!
```

Why do you say that?

> bye

It was nice talking to you, goodbye!

One possible solution to the task is [here](#)

 Stop and consider

1. Do you think the random responses would 'trick' someone into thinking that the bot actually understood them?
2. What features would the bot need to be more effective?
3. If a bot could really 'understand' the meaning of a sentence, would it need to 'remember' the meaning of previous sentences in a conversation too?

## Challenge

---

Choose one of the "stop and consider" elements above and either try to implement them in code or write a solution on paper using pseudocode.

In the next lesson, you'll learn about a number of other approaches to parsing natural language and machine learning.

## Post-lecture quiz

---

## Review & Self Study

---

Take a look at the references below as further reading opportunities.

## References

1. Schubert, Lenhart, "Computational Linguistics", *The Stanford Encyclopedia of Philosophy* (Spring 2020 Edition), Edward N. Zalta (ed.), URL =  
<https://plato.stanford.edu/archives/spr2020/entries/computational-linguistics/>.
2. Princeton University "About WordNet." [WordNet](#). Princeton University. 2010.

# Assignment

---

[Search for a Bot](#)

## Common Natural Language Processing Tasks and Techniques

For most *Natural Language Processing* tasks, the text to be processed must be broken down, examined, and the results stored or cross referenced with rules and data sets. This allows the programmer to derive the meaning or intent or only the frequency of terms and words in a text.

### Pre-lecture quiz

---

Let's discover common techniques used in processing text. Combined with machine learning, these techniques help you to analyse large amounts of text efficiently. Before applying ML to these tasks, however, let's understand the problems encountered by an NLP specialist.

### Tasks common to NLP

---

👉 **Tokenization** Probably the first thing most NLP algorithms have to do is split the text into tokens, or words. While this sounds simple, having to account for punctuation and different languages' word and sentence delimiters can make it tricky. Thought it might seem very straightforward to split a sentence into words, you might have to use some other methods to determine demarcations.

👉 **Embeddings**

Word embeddings are a way to convert your text data numerically. This is done in a way so that words with a similar meaning or words used together cluster together.

Try [this interesting tool](#) to experiment with word embeddings. Clicking on one word shows clusters of similar words: 'toy' clusters with 'disney', 'lego', 'playstation', and 'console'.

## Parsing & Part-of-speech Tagging

Every word that has been tokenized can be tagged as a part of speech - a noun, verb, or adjective etc. The sentence `the quick red fox jumped over the lazy brown dog` might be POS tagged as `fox = noun, jumped = verb` etc.

Parsing is recognizing what words are related to each other in a sentence - for instance `the quick red fox jumped` is an adjective-noun-verb sequence that is separate from `lazy brown dog` sequence.

## Word and Phrase Frequencies

A useful tool when analyzing a large body of text is to build a dictionary of every word or phrase of interest and how often it appears. The phrase `the quick red fox jumped over the lazy brown dog` has a word frequency of 2 for `the`.

### Example:

The Rudyard Kipling poem *The Winners* has a verse:

What the moral? Who rides may read.  
When the night is thick and the tracks are blind  
A friend at a pinch is a friend, indeed,  
But a fool to wait for the laggard behind.  
Down to Gehenna or up to the Throne,  
He travels the fastest who travels alone.

As phrase frequencies can be case insensitive or case sensitive as required, the phrase `a friend` has a frequency of 2 and `the` has a frequency of 6, and `travels` is 2.

## N-grams

A text can be split into sequences of words of a set length, a single word (unigram), two words (bigrams), three words (trigrams) or any number of words (n-grams).

## Example

For instance the quick red fox jumped over the lazy brown dog with a n-gram score of 2 produces the following n-grams:

1. the quick
2. quick red
3. red fox
4. fox jumped
5. jumped over
6. over the
7. the lazy
8. lazy brown
9. brown dog

It might be easier to visualise it as a sliding box over the sentence. Here it is for n-grams of 3 words, the n-gram is in bold in each sentence:

1. **the quick red** fox jumped over the lazy brown dog
2. the **quick red fox** jumped over the lazy brown dog
3. the quick **red fox jumped** over the lazy brown dog
4. the quick red **fox jumped over** the lazy brown dog
5. the quick red fox **jumped over the** lazy brown dog
6. the quick red fox jumped **over the lazy** brown dog
7. the quick red fox jumped over **the lazy brown** dog
8. the quick red fox jumped over the **lazy brown dog**

## 🎓 Noun phrase Extraction

In most sentences, there is a noun that is the subject, or object of the sentence. In English, it is often identifiable as having 'a' or 'an' or 'the' preceding it. Identifying the subject or object of a sentence by 'extracting the noun phrase' is a common task in NLP when attempting to understand the meaning of a sentence.

## Example

In the sentence the quick red fox jumped over the lazy brown dog there are 2 noun phrases: **quick red fox** and **lazy brown dog**.

### Sentiment analysis

A sentence or text can be analysed for sentiment, or how positive or negative it is. Sentiment is measured in polarity and objectivity/subjectivity. Polarity is measured from -1.0 to 1.0 (negative to positive) and 0.0 to 1.0 (most objective to most subjective).

- Later you'll learn that there are different ways to determine sentiment using machine learning, but one way is to have a list of words and phrases that are categorized as positive or negative by a human expert and apply that model to text to calculate a polarity score. Can you see how this would work in some circumstances and less well in others?

### Inflection

Inflection enables you to take a word and get the singular or plural of the word.

### Lemmatization

A lemma is the root or headword for a set of words, for instance flew, flies, flying have a lemma of the verb fly.

There are also useful databases available for the NLP researcher, notably:

### WordNet

WordNet is a database of words, synonyms, antonyms and many other details for every word in many different languages. It is incredibly useful when attempting to build translations, spell checkers, or language tools of any type.

## NLP Libraries

Luckily, you don't have to build all of these techniques yourself, as there are excellent Python libraries available that make it much more accessible to developers who aren't specialized in natural language processing or machine learning. The next lessons include more examples of these, but here you will learn some useful examples to help you with the next task.

Let's use a library called TextBlob as it contains helpful APIs for tackling these types of tasks.

TextBlob "stands on the giant shoulders of [NLTK](#) and [pattern](#), and plays nicely with both." It has a considerable amount of ML embedded in its API.

Note: A useful [Quick Start](#) guide is available for TextBlob that is recommended for experienced Python developers

When attempting to identify *noun phrases*, TextBlob offers several options of extractors to find noun phrases. Take a look at `ConllExtractor`.

python

```
from textblob import TextBlob
from textblob.np_extractors import ConllExtractor
# import and create a Conll extractor to use later
extractor = ConllExtractor()

# later when you need a noun phrase extractor:
user_input = input("> ")
user_input_blob = TextBlob(user_input, np_extractor=extractor) # note non-
np = user_input_blob.noun_phrases
```

What's going on here? [ConllExtractor](#) is "A noun phrase extractor that uses chunk parsing trained with the ConLL-2000 training corpus." ConLL-2000 refers to the Conference on Computational Natural Language Learning (CoNLL-2000). Each year the conference hosted a workshop to tackle a thorny NLP problem, and in 2000 it was noun chunking. A model was trained on the Wall Street Journal, with "sections 15-18 as training data (211727 tokens) and section 20 as test data (47377 tokens)". You can look at the procedures used [here](#) and the [results](#).

## Task: Improving your bot with a little NLP

In the previous lesson you built a very simple Q&A bot. Now, you'll make Marvin a bit more sympathetic by analyzing your input for sentiment and printing out a response to match the sentiment. You'll also need to identify a `noun_phrase` and ask about it.

Your steps when building a better conversational bot:

1. Print instructions advising the user how to interact with the bot
2. Start loop
  1. Accept user input
  2. If user has asked to exit, then exit
  3. Process user input and determine appropriate sentiment response
  4. If a noun phrase is detected in the sentiment, pluralize it and ask for more input on that topic
  5. Print response
3. loop back to step 2

Here is the code snippet to determine sentiment using TextBlob. Note there are only four *gradients* of sentiment response (you could have more if you like):

python

```
if user_input_blob.polarity <= -0.5:  
    response = "Oh dear, that sounds bad."  
elif user_input_blob.polarity <= 0:  
    response = "Hmm, that's not great."  
elif user_input_blob.polarity <= 0.5:  
    response = "Well, that sounds positive."  
elif user_input_blob.polarity <= 1:  
    response = "Wow, that sounds great."
```

Here is some sample output to guide you (user input is on the lines with starting with >):

```
Hello, I am Marvin, the friendly robot.  
You can end this conversation at any time by typing 'bye'  
After typing each answer, press 'enter'  
How are you today?  
> I am ok  
Well, that sounds positive. Can you tell me more?  
> I went for a walk and saw a lovely cat  
Well, that sounds positive. Can you tell me more about lovely cats?  
> cats are the best. But I also have a cool dog  
Wow, that sounds great. Can you tell me more about cool dogs?  
> I have an old hounddog but he is sick  
Hmm, that's not great. Can you tell me more about old hounddogs?
```

> bye

It was nice talking to you, goodbye!

One possible solution to the task is [here](#)

### Knowledge Check

1. Do you think the sympathetic responses would 'trick' someone into thinking that the bot actually understood them?
2. Does identifying the noun phrase make the bot more 'believable'?
3. Why would extracting a 'noun phrase' from a sentence a useful thing to do?

### Challenge

---

Take a task in the prior knowledge check and try to implement it. Test the bot on a friend. Can it trick them? Can you make your bot more 'believable'?

### Post-lecture quiz

---

### Review & Self Study

---

In the next few lessons you will learn more about sentiment analysis. Research this interesting technique in articles such as these on [KD Nuggets](#)

### Assignment

---

[Make a bot talk back](#)

## Translation and Sentiment Analysis with ML

In the previous lessons you learned how to build a basic bot using TextBlob, a library that embeds ML behind-the-scenes to perform basic NLP tasks such as noun phrase extraction. Another important challenge in computational linguistics is accurate *translation* of a sentence from one spoken or written language to another.

## Pre-lecture quiz

---

This is a very hard problem compounded by the fact that there are thousands of languages and each can have very different grammar rules. One approach is to convert the formal grammar rules for one language, such as English, into a non-language dependent structure, and then translate it by converting back to another language. This means that you would take the following steps:

1. Identify or tag the words in input language into nouns, verbs etc.
2. Produce a direct translation of each word in the target language format

✿ **Example:** In **English**, the simple sentence `I feel happy` is 3 words in the order **subject (I)**, **verb (feel)**, **adjective (happy)**. However, in the **Irish** language, the same sentence has a very different grammatical structure - emotions like "happy" or "sad" are expressed as being upon you. The English phrase `I feel happy` in Irish would be `Tá athas orm`. A literal translation would be `Happy is upon me`. Of course, an Irish speaker translating to English would say `I feel happy`, not `Happy is upon me`, because they understand the meaning of the sentence, even if the words and sentence structure are different. The formal order for the sentence in Irish are **verb (Tá or is)**, **adjective (athas, or happy)**, **subject (orm, or upon me)**.

## Translation

---

A naive translation program might translate words only, ignoring the sentence structure.

✓ If you've learned a second (or third or more) language as an adult, you might have started by thinking in your native language, translating a concept word by word in your head to the second language, and then speaking out your translation. This is similar to what naive translation computer programs are doing. It's important to get past this phase to attain fluency!

Naive translation leads to bad (and sometimes hilarious) mistranslations: `I feel happy` translates literally to `Mise bhraitheann athas` in Irish. That means (literally) `me feel happy`

and is not a valid Irish sentence. Even though English and Irish are languages spoken on two closely neighboring islands, they are very different languages with different grammar structures.

You can watch some videos about Irish linguistic traditions such as [this one](#)

## Machine learning approaches

So far, you've learned about the formal rules approach to natural language processing. Another approach is to ignore the meaning of the words, and *instead use machine learning to detect patterns*. This can work in translation if you have lots of text (*a corpus*) or texts (*corpora*) in both the origin and target languages. For instance, consider the case of *Pride and Prejudice*, a well-known English novel written by Jane Austen in 1813. If you consult the book in English and a human translation of the book in *French*, you could detect phrases in one that are idiomatically translated into the other. You'll do that in a minute.

For instance, when an English phrase such as

John looked at the cake with a wolfish grin is translated literally, to, say French, it might become John regarda le gâteau avec un sourire de loup . A reader of both languages would understand that the direct translation of wolfish grin is not the French translation wolf smile but a synonym - in this case for being very hungry or voracious. A better translation that a human might make would be John regarda le gâteau avec voracité , because it better conveys the meaning. If a ML model has enough human translations to build a model on, it can improve the accuracy of translations by identifying common patterns in texts that have been previously translated by expert human speakers of both languages.

## Task: Translation

You can use TextBlob to translate sentences. Try the famous first line of **Pride and Prejudice**:

```
python  
from textblob import TextBlob  
  
blob = TextBlob(  
    "It is a truth universally acknowledged, that a single man in possession  
)  
print(blob.translate(to="fr"))
```

TextBlob does a pretty good job at the translation: "C'est une vérité universellement reconnue, qu'un homme célibataire en possession d'une bonne fortune doit avoir besoin d'une femme!".

I would argue that TextBlob's translation is far more exact, in fact, than the 1932 French translation of the book by V. Leconte and Ch. Pressoir:

"C'est une vérité universelle qu'un celibataire pourvu d'une belle fortune doit avoir envie de se marier, et, si peu que l'on sache de son sentiment à cet égard, lorsqu'il arrive dans une nouvelle résidence, cette idée est si bien fixée dans l'esprit de ses voisins qu'ils le considèrent sur-le-champ comme la propriété légitime de l'une ou l'autre de leurs filles."

In this case, the translation informed by ML does a better job than the human translator who is unnecessarily putting words in the original author's mouth for 'clarity'.

What's going on here? and why is TextBlob so good at translation? Well, behind the scenes, it's using Google Translate, a sophisticated AI able to parse millions of phrases to predict the best strings for the task at hand. There's nothing manual going on here and you need an internet connection to use `blob.translate`.

 Try some more sentences. Which is better, ML or human translation? In which cases?

## Sentiment analysis

Another area where machine learning can work very well is sentiment analysis. A non-ML approach to sentiment is to identify words and phrases which are 'positive' and 'negative'. Then, given a new piece of text, calculate the total value of the positive, negative and neutral words to identify the overall sentiment.

This approach is easily tricked as you may have seen in the Marvin task - the sentence **Great, that was a wonderful waste of time, I'm glad we are lost on this dark road** is a sarcastic, negative sentiment sentence, but the simple algorithm detects 'great', 'wonderful', 'glad' as positive and 'waste', 'lost' and 'dark' as negative. The overall sentiment is swayed by these conflicting words.

 Stop a second and think about how we convey sarcasm as human speakers. Tone inflection plays a large role. Try to say the phrase "Well, that film was awesome" to discover how your voice conveys meaning.

## Machine learning approaches

The ML approach would be to hand gather negative and positive bodies of text - tweets, or movie reviews, or anything where the human has given a score *and* a written opinion. Then NLP techniques can be applied to opinions and scores, so that patterns emerge (e.g., positive movie reviews tend to have the phrase 'Oscar worthy' more than negative movie reviews, or positive restaurant reviews say 'gourmet' much more than 'disgusting').

 **Example:** If you worked in a politician's office and there was some new law being debated, constituents might write to the office with emails supporting or emails against the particular new law. Let's say you are tasked with reading the emails and sorting them in 2 piles, for and against. If there were a lot of emails, you might be overwhelmed attempting to read them all. Wouldn't it be nice if a bot could read them all for you, understand them and tell you in which pile each email belonged?

One way to achieve that is to use Machine Learning. You would train the model with a portion of the against emails and a portion of the for emails. The model would tend to associate phrases and words with the against side and the for side, but it would not understand any of the content, only that certain words and patterns were more likely to appear in an against or a for email. You could test it with some emails that you had not used to train the model, and see if it came to the same conclusion as you did. Then, once you were happy with the accuracy of the model, you could process future emails without having to read each one.

 Does this process sound like processes you have used in previous lessons?

## Task: Sentimental Sentences

Sentiment is measured in with a *polarity* of -1 to 1, meaning -1 is the most negative sentiment, and 1 is the most positive. Sentiment is also measured with an 0 - 1 score for objectivity (0) and subjectivity (1).

Take another look at Jane Austen's *Pride and Prejudice*. The text is available here at [Project Gutenberg](#). The sample below shows a short program which analyses the sentiment of first and last sentences from the book and display its sentiment polarity and subjectivity/objectivity score. You should us the TextBlob library (described above) to determine sentiment (you do not have to write your own sentiment calculator) in the following task.

python

```
from textblob import TextBlob  
  
quote1 = """It is a truth universally acknowledged, that a single man in po
```

```

quote2 = """Darcy, as well as Elizabeth, really loved them; and they were I

sentiment1 = TextBlob(quote1).sentiment
sentiment2 = TextBlob(quote2).sentiment

print(quote1 + " has a sentiment of " + str(sentiment1))
print(quote2 + " has a sentiment of " + str(sentiment2))
# outputs:
# It is a truth universally acknowledged, that a single man in possession o
# Darcy, as well as Elizabeth, really loved them; and they were
# both ever sensible of the warmest gratitude towards the persons
# who, by bringing her into Derbyshire, had been the means of
# uniting them. has a sentiment of Sentiment(polarity=0.7, subjectivi

```

Your task is to determine, using sentiment polarity, if *Pride and Prejudice* has more absolutely positive sentences than absolutely negative ones. For this task, you may assume that a polarity score of 1 or -1 is absolutely positive or negative respectively.

#### Steps:

1. Download a copy of Pride and Prejudice from Project Gutenberg as a .txt file. Remove the metadata at the start and end of the file, leaving only the original text
2. Open the file in Python and extract the contents as a string
3. Create a TextBlob using the book string
4. Analyse each sentence in the book in a loop
  1. If the polarity is 1 or -1 store the sentence in an array or list of positive or negative messages
5. At the end, print out all the positive sentences and negative sentences (separately) and the number of each.

Here is a sample solution.

#### Knowledge Check

1. The sentiment is based on words used in the sentence, but does it code *understand* the words?
2. Do you think the sentiment polarity is accurate, or in other words, do you *agree* with the scores?
  1. In particular, do you agree or disagree with the absolute **positive** polarity of the following sentences?
    - "What an excellent father you have, girls!" said she, when the door was shut.
    - "Your examination of Mr. Darcy is over, I presume," said Miss Bingley; "and pray what is the result?" "I am perfectly convinced by it that Mr. Darcy has no defect.
    - How wonderfully these sort of things occur!
    - I have the greatest dislike in the world to that sort of thing.
    - Charlotte is an excellent manager, I dare say.

- "This is delightful indeed!"
- I am so happy!
- Your idea of the ponies is delightful.

2. The next 3 sentences were scored with an absolute positive sentiment, but on close reading, they are not positive sentences. Why did the sentiment analysis think they were positive sentences?

- Happy shall I be, when his stay at Netherfield is over!" "I wish I could say anything to comfort you," replied Elizabeth; "but it is wholly out of my power.
- If I could but see you as happy!
- Our distress, my dear Lizzy, is very great.

3. Do you agree or disagree with the absolute **negative** polarity of the following sentences?

- Everybody is disgusted with his pride.
- "I should like to know how he behaves among strangers." "You shall hear then—but prepare yourself for something very dreadful.
- The pause was to Elizabeth's feelings dreadful.
- It would be dreadful!

 Any aficionado of Jane Austen will understand that she often uses her books to critique the more ridiculous aspects of English Regency society. Elizabeth Bennett, the main character in Pride and Prejudice, is a keen social observer (like the author) and her language is often heavily nuanced. Even Mr. Darcy (the love interest in the story) notes Elizabeth's playful and teasing use of language: "I have had the pleasure of your acquaintance long enough to know that you find great enjoyment in occasionally professing opinions which in fact are not your own."

## Challenge

---

Can you make Marvin even better by extracting other features from the user input?

## Post-lecture quiz

---

## Review & Self Study

---

There are many ways to extract sentiment from text. Think of the business applications that might make use of this technique. Think about how it can go awry. Read more about sophisticated enterprise-ready systems that analyze sentiment such as [Azure Text Analysis](#). Test some of the Pride and Prejudice sentences above and see if it can detect nuance.

# Assignment

Poetic License

## Introduction to Time Series Forecasting

**Machine Learning Time Series Forecasting**

Looking at data overtime to forecast or predict future values based on patterns or recurring trends.

**Data Characteristics:**

- Seasonality
- Outliers
- long-run cycle
- Constant variance
- abrupt changes

**examples**

- Weather forecast
- Holiday season sales
- Stock market
- online user traffic

**Using ARIMA**

Auto regressive  
integrated  
moving average  
stationarity

Previous values in your data  
long memory models

$y_t - y_{t-1}$   
time  
 $\Delta$  diff

LIB  
Statmodels  
Define model w/ SARIMAX()

@AzureAdvocates @girlie\_mac

Sketchnote by [Tomomi Imura](#)

In this lesson and the following one, you will learn a bit about Time Series Forecasting, an interesting and valuable part of a ML scientist's repertoire that is a bit lesser known than other topics. Time Series Forecasting is a sort of crystal ball: based on past performance of a variable such as price, you can predict its future potential value.

## Time Series Forecasting in Minutes



## Pre-lecture quiz

It's a useful and interesting field with real value to business, given its direct application to problems of pricing, inventory, and supply chain issues. While deep learning techniques have started to be used to gain more insights in the prediction of future performance, Time Series Forecasting remains a field greatly informed by classic ML techniques.

Penn State's useful Time Series curriculum can be found [here](#)

## Introduction

Supposing you maintain an array of smart parking meters that provide data about how often they are used and for how long over time. What if you could generate revenue to maintain your streets by slightly augmenting the prices of the meters when there is greater demand for them? What if you could predict, based on the meter's past performance, its future value according to the laws of supply and demand? This is a challenge that could be tackled by Time Series Forecasting. It wouldn't make those folks in search of a rare parking spot in busy times very happy to have to pay more for it, but it would be a sure way to generate revenue to clean the streets!

Let's explore some of the types of Time Series algorithms and start a notebook to clean and prepare some data. The data you will analyze is taken from the GEFCom2014 forecasting competition. It consists of 3 years of hourly electricity load and temperature values between 2012 and 2014. Given

the historical patterns of electricity load and temperature, you can predict future values of electricity load. In this example, you'll learn how to forecast one time step ahead, using historical load data only.

Before starting, however, it's useful to understand what's going on behind the scenes.

## Some Definitions

---

When encountering the term 'time series' you need to understand its use in several different contexts.

### Time Series

In mathematics, "a time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time." An example of a time series is the daily closing value of the [Dow Jones Industrial Average](#). The use of time series plots and statistical modeling is frequently encountered in signal processing, weather forecasting, earthquake prediction, and other fields where events occur and data points can be plotted over time.

### Time Series Analysis

Time Series Analysis is the analysis of the above mentioned time series data. Time series data can take distinct forms, including 'interrupted time series' which detects patterns in a time series' evolution before and after an interrupting event. The type of analysis needed for the time series depends on the nature of the data. Time series data itself can take the form of series of numbers or characters.

The analysis be performed using a variety of methods, including frequency-domain and time-domain, linear and nonlinear, and more. [Learn more](#) about the may ways to analyze this type of data.

### Time Series Forecasting

Time Series Forecasting is the use of a model to predict future values based on patterns displayed by previously gathered data as it occurred in the past. While it is possible to use regression models to explore time series data, with time indices as x variables on a plot, this type of data is best analyzed using special types of models.

Time series data is a list of ordered observations, unlike data that can be analyzed by linear regression. The most common one is ARIMA, an acronym that stands for "Autoregressive Integrated

## Moving Average".

ARIMA models "relate the present value of a series to past values and past prediction errors." They are most appropriate for analyzing time-domain data, where data is ordered over time.

There are several types of ARIMA models, which you can learn about [here](#) and which you will touch on in the next lesson.

In the next lesson, you will build an ARIMA model using Univariate Time Series, which focuses on one variable that changes its value over time. An example of this type of data is [this dataset](#) that records the monthly CO<sub>2</sub> concentration at the Mauna Loa Observatory:

CO2	YearMonth	Year	Month
330.62	1975.04	1975	1
331.40	1975.13	1975	2
331.87	1975.21	1975	3
333.18	1975.29	1975	4
333.92	1975.38	1975	5
333.43	1975.46	1975	6
331.85	1975.54	1975	7
330.01	1975.63	1975	8
328.51	1975.71	1975	9
328.41	1975.79	1975	10
329.25	1975.88	1975	11
330.97	1975.96	1975	12

- Identify the variable that changes over time in this dataset

# Time Series data characteristics to consider

When looking at time series data, you might notice that it has certain characteristics that you need to take into account and mitigate to better understand its patterns. If you consider time series data as potentially providing a 'signal' that you want to analyze, these characteristics can be thought of as 'noise'. You often will need to reduce this 'noise' by offsetting some of these characteristics using some statistical techniques.

## Trends

Measurable increases and decreases over time. [Read more](#) about how to use and, if necessary, remove trends from your time series.

## Seasonality

Periodic fluctuations, such as holiday rushes that might affect sales, for example. [Take a look](#) at how different types of plots display seasonality in data.

## Outliers

Outliers are far away from the standard data variance.

## Long-run cycle

Independent of seasonality, data might display a long-run cycle such as an economic down-turn that lasts longer than a year.

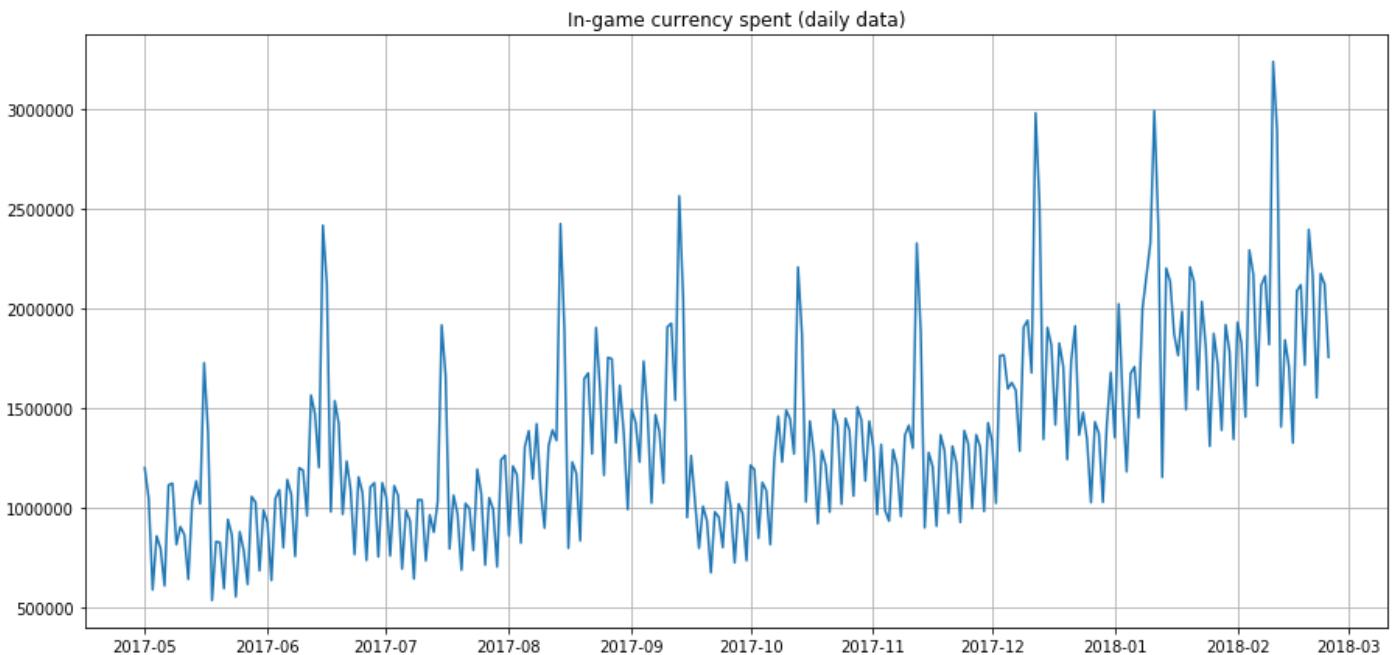
## Constant variance

Over time, some data display constant fluctuations, such as energy usage per day and night.

## Abrupt changes

The data might display an abrupt change that might need further analysis. The abrupt shuttering of businesses due to COVID, for example, caused changes in data.

Here is a [sample time series plot](#) showing daily in-game currency spent over a few years. Can you identify any of the characteristics listed above in this data?



## Getting started with power usage data

Let's get started creating a time series model to predict future power usage given past usage.

The data in this example is taken from the GEFCom2014 forecasting competition. It consists of 3 years of hourly electricity load and temperature values between 2012 and 2014.

Tao Hong, Pierre Pinson, Shu Fan, Hamidreza Zareipour, Alberto Troccoli and Rob J. Hyndman, "Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond", International Journal of Forecasting, vol.32, no.3, pp 896-913, July-September, 2016.

In the `working` folder of this lesson, open the `notebook.ipynb` file. Start by adding libraries that will help you load and visualize data

python

```
import os
import matplotlib.pyplot as plt
from common.utils import load_data
%matplotlib inline
```

Note, you are using the files from the included `common` folder which set up your environment and handle downloading the data.

Next, examine the data as a dataframe

python

```
data_dir = './data'  
energy = load_data(data_dir)[['load']]  
energy.head()
```

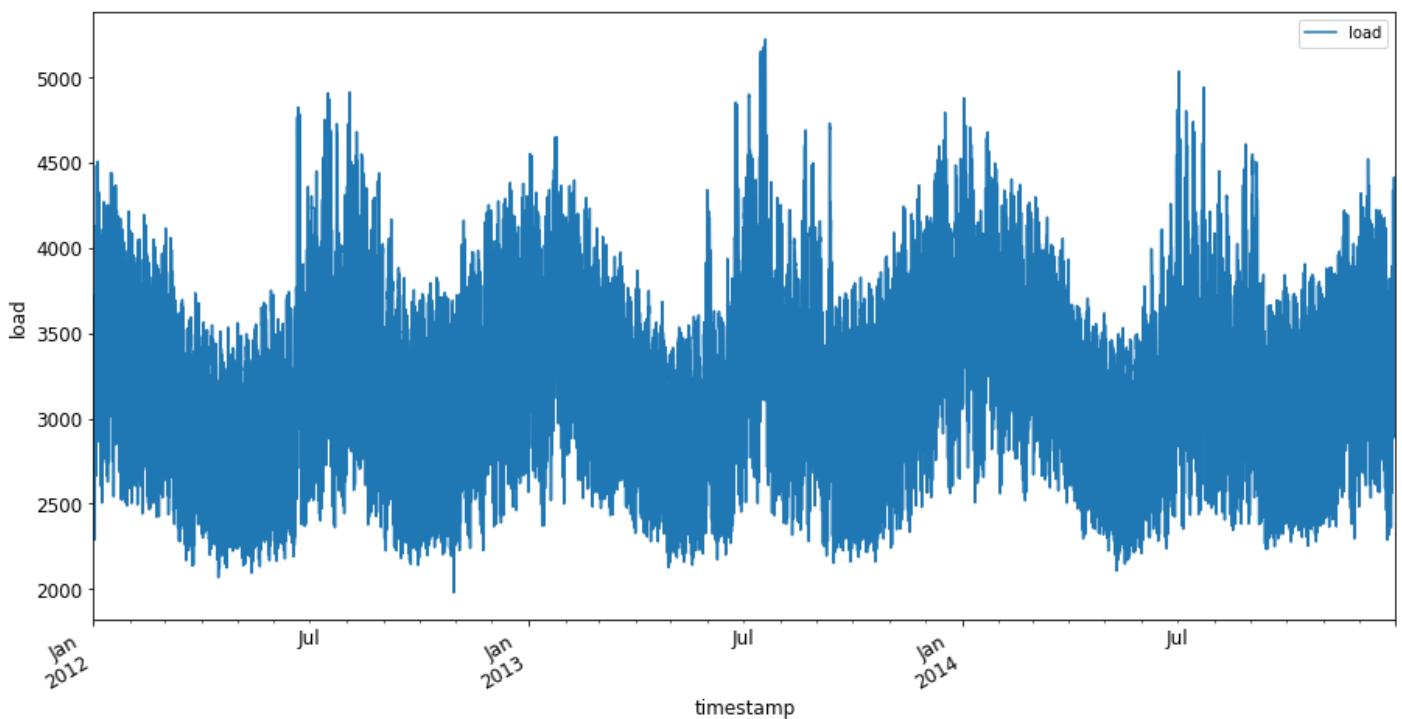
You can see that there are two columns representing date and load:

load	
2012-01-01 00	2698.0
2012-01-01 01	2558.0
2012-01-01 02	2444.0
2012-01-01 03	2402.0
2012-01-01 04	2403.0

Now, plot the data:

python

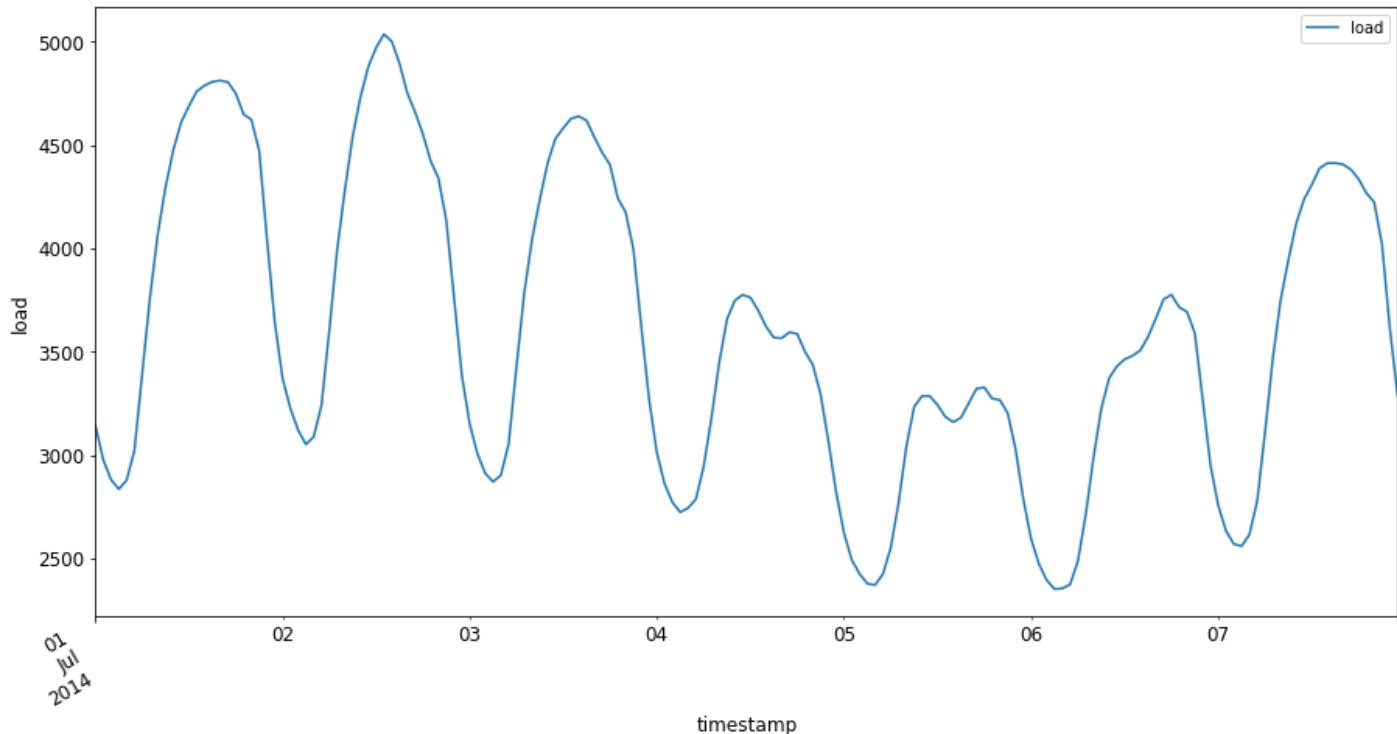
```
energy.plot(y='load', subplots=True, figsize=(15, 8), fontsize=12)  
plt.xlabel('timestamp', fontsize=12)  
plt.ylabel('load', fontsize=12)  
plt.show()
```



Now, plot the first week of July 2014

python

```
energy['2014-07-01':'2014-07-07'].plot(y='load', subplots=True, figsize=(12, 8))
plt.xlabel('timestamp', fontsize=12)
plt.ylabel('load', fontsize=12)
plt.show()
```



A beautiful plot! Take a look at these plots and see if you can determine any of the characteristics listed above. What can we surmise just by visualizing the data?

In the next lesson, you will create an ARIMA model to create some forecasts.

## 🚀 Challenge

Make a list of all the industries and areas of inquiry you can think of that would benefit from Time Series Forecasting. Can you think of an application of these techniques in the arts? In Econometrics? Ecology? Retail? Industry? Finance? Where else?

## Post-lecture quiz

## Review & Self Study

Although we won't cover them here, neural networks are sometimes used to enhance classic methods of Time Series Forecasting. Read more about them [in this article](#)

## Assignment

---

[Visualize some more Time Series](#)

# Time Series Forecasting with ARIMA

In the previous lesson, you learned a bit about Time Series Forecasting and loaded a dataset showing the fluctuations of electrical load over a time period.

The screenshot shows a DataCamp course interface. At the top, there's a black bar. Below it is a blue header bar with the DataCamp logo on the left and the text 'Forecasting Product Demand in R' on the right. The main content area has a white background. In the center, the title 'ARIMA Time Series 101' is displayed in bold black text. Below the title, there's a box containing the author's name, Aric LaBarr, Ph.D., and their title, Senior Data Scientist at Elder Research. The rest of the page is mostly blacked out.

A brief introduction to ARIMA models. The example is done in R, but the concepts are universal.

## Pre-lecture quiz

---

In this lesson, you will discover a specific way to build models with ARIMA: AutoRegressive /Integrated Moving Average. ARIMA models are particularly suited to fit data that shows non-stationarity.

🎓 Stationarity, from a statistical context, refers to data whose distribution does not change when shifted in time. Non-stationary data, then, shows fluctuations due to trends that must be transformed to be analyzed. Seasonality, for example, can introduce fluctuations in data and can be eliminated by a process of 'seasonal-differencing'.

🎓 Differencing data, again from a statistical context, refers to the process of transforming non-stationary data to make it stationary by removing its non-constant trend. "Differencing removes the changes in the level of a time series, eliminating trend and seasonality and consequently stabilizing the mean of the time series." [Paper by Shixiong et al](#)

Let's unpack the parts of ARIMA to better understand how it helps us model Time Series and help us make predictions against it.

## AR - for AutoRegressive

---

Autoregressive models, as the name implies, look 'back' in time to analyze previous values in your data and make assumptions about them. These previous values are called 'lags'. An example would be data that shows monthly sales of pencils. Each month's sales total would be considered an 'evolving variable' in the dataset. This model is built as the "evolving variable of interest is regressed on its own lagged (i.e., prior) values." [wikipedia](#)

## I - for Integrated

---

As opposed to the similar 'ARMA' models, the 'I' in ARIMA refers to its integrated aspect. The data is 'integrated' when differencing steps are applied so as to eliminate non-stationarity.

## MA - for Moving Average

---

The moving-average aspect of this model refers to the output variable that is determined by observing the current and past values of lags.

Bottom line: ARIMA is used to make a model fit the special form of time series data as closely as possible.

## Preparation

Open the `/working` folder in this lesson and find the `notebook.ipynb` file. Run the notebook to load the `statsmodels` Python library; you will need this for ARIMA models.

## Load necessary libraries

---

Now, load up several more libraries useful for plotting data:

python

```
import os
import warnings
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime as dt
import math

from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.preprocessing import MinMaxScaler
from common.utils import load_data, mape
from IPython.display import Image

%matplotlib inline
pd.options.display.float_format = '{:,.2f}'.format
np.set_printoptions(precision=2)
warnings.filterwarnings("ignore") # specify to ignore warning messages
```

## Load the data

---

Load the data from the `/data/energy.csv` file into a Pandas dataframe and take a look:

python

```
energy = load_data('./data')[['load']]
energy.head(10)
```

# Plot the data

---

Plot all the available energy data from January 2012 to December 2014. There should be no surprises as we saw this data in the last lesson:

python

```
energy.plot(y='load', subplots=True, figsize=(15, 8), fontsize=12)
plt.xlabel('timestamp', fontsize=12)
plt.ylabel('load', fontsize=12)
plt.show()
```

Now, let's build a model!

# Create training and testing datasets

---

Now your data is loaded, so you can separate it into train and test sets. You'll train your model on the train set. As usual, after the model has finished training, you'll evaluate its accuracy using the test set. You need to ensure that the test set covers a later period in time from the training set to ensure that the model does not gain information from future time periods.

Allocate a two-month period from September 1 to October 31, 2014 to the training set. The test set will include the two-month period of November 1 to December 31, 2014.

python

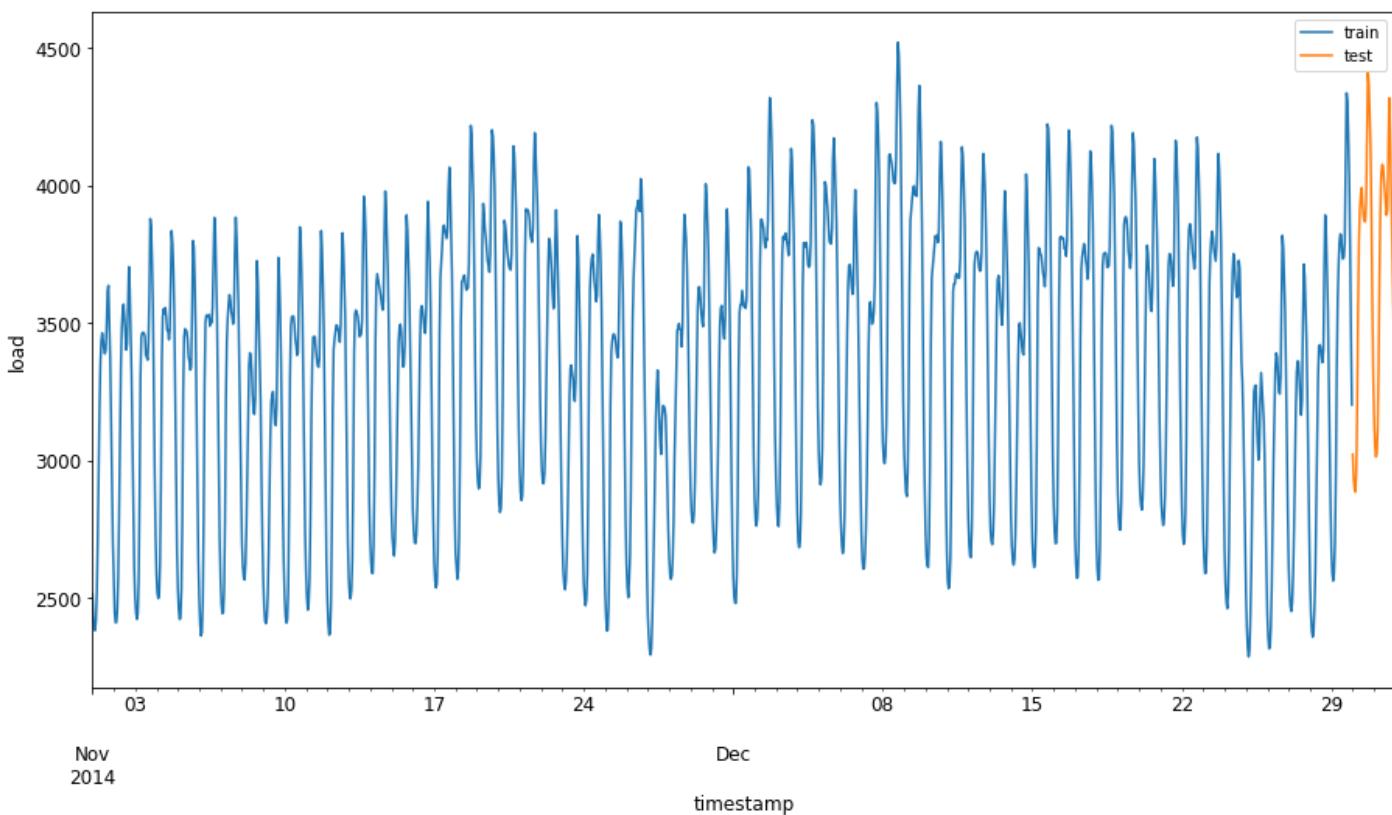
```
train_start_dt = '2014-11-01 00:00:00'
test_start_dt = '2014-12-30 00:00:00'
```

Since this data reflects the daily consumption of energy, there is a strong seasonal pattern, but the consumption is most similar to the consumption in more recent days. You can visualize the differences:

python

```
energy[(energy.index < test_start_dt) & (energy.index >= train_start_dt)][
    .join(energy[test_start_dt:][['load']].rename(columns={'load':'test'}))
    .plot(y=['train', 'test'], figsize=(15, 8), fontsize=12)
plt.xlabel('timestamp', fontsize=12)
```

```
plt.ylabel('load', fontsize=12)
plt.show()
```



Therefore, using a relatively small window of time for training the data should be sufficient.

Note: Since the function we use to fit the ARIMA model uses in-sample validation during fitting, we will omit validation data.

## Prepare the data for training

Now, you need to prepare the data for training by performing two tasks:

1. Filter the original dataset to include only the aforementioned time periods per set and only including the needed column 'load' plus the date:

```
python
train = energy.copy()[(energy.index >= train_start_dt) & (energy.index < test_start_dt)]
test = energy.copy()[energy.index >= test_start_dt][['load']]

print('Training data shape: ', train.shape)
print('Test data shape: ', test.shape)
```

You can see the shape of the data:

Training data shape: (1416, 1) Test data shape: (48, 1)

1. Scale the data to be in the range (0, 1).

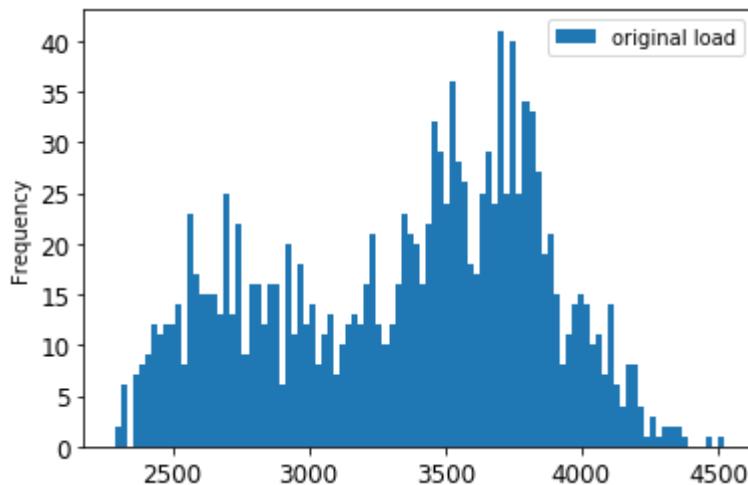
python

```
scaler = MinMaxScaler()  
train['load'] = scaler.fit_transform(train)  
train.head(10)
```

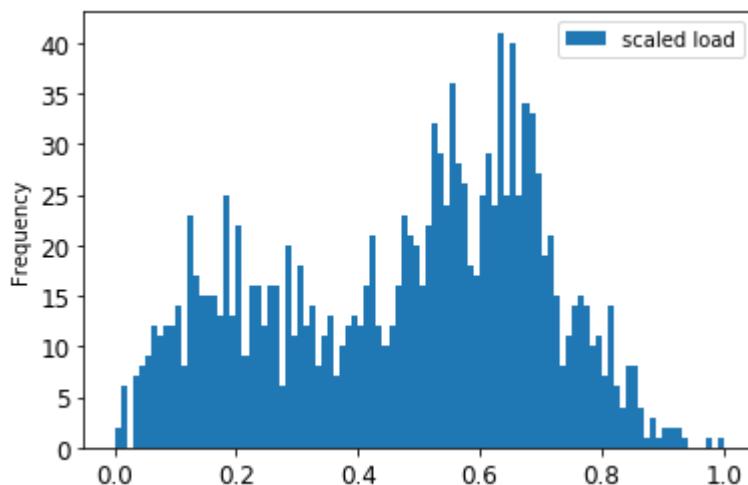
Now, visualize the original vs. scaled data:

python

```
energy[(energy.index >= train_start_dt) & (energy.index < test_start_dt)][  
    train.rename(columns={'load':'scaled load'}).plot.hist(bins=100, fontsize=  
    plt.show()
```



The original data



## The scaled data

Now that you have calibrated the scaled data, you can scale the test data:

python

```
test['load'] = scaler.transform(test)  
test.head()
```

## Implement ARIMA

It's time to implement ARIMA! You'll now use the `statsmodels` library that you installed earlier.

Now you need to follow several steps

1. Define the model by calling `SARIMAX()` and passing in the model parameters: `p`, `d`, and `q` parameters, and `P`, `D`, and `Q` parameters.
2. The model is prepared on the training data by calling the `fit()` function.
3. Predictions can be made by calling the `forecast()` function and specifying the number of steps (the `horizon`) to forecast

🎓 What are all these parameters for? In an ARIMA model there are 3 parameters that are used to help model the major aspects of a time series: seasonality, trend, and noise. These parameters are:

`p` : the parameter associated with the auto-regressive aspect of the model, which incorporates *past* values. `d` : the parameter associated with the integrated part of the model, which affects the amount of *differencing* (🎓 remember differencing 🤔 ?) to apply to a time series. `q` : the parameter associated with the moving-average part of the model.

Note: If your data has a seasonal aspect - which this one does - , we use a seasonal ARIMA model (SARIMA). In that case you need to use another set of parameters: `P` , `D` , and `Q` which describe the same associations as `p` , `d` , and `q` , but correspond to the seasonal components of the model.

Start by setting your preferred horizon value. Let's try 3 hours:

python

```
# Specify the number of steps to forecast ahead
HORIZON = 3
print('Forecasting horizon:', HORIZON, 'hours')
```

Selecting the best values for an ARIMA model's parameters can be challenging as it's somewhat subjective and time intensive. You might consider using an `auto_arima()` function from the [pyramid library](#), but for now try some manual selections to find a good model.

python

```
order = (4, 1, 0)
seasonal_order = (1, 1, 0, 24)

model = SARIMAX(endog=train, order=order, seasonal_order=seasonal_order)
results = model.fit()

print(results.summary())
```

A table of results is printed.

You've built your first model! Now we need to find a way to evaluate it.

## Evaluate your model

---

To evaluate your model, you can perform the so-called `walk forward` validation. In practice, time series models are re-trained each time a new data becomes available. This allows the model to make the best forecast at each time step.

Starting at the beginning of the time series using this technique, train the model on the train data set. Then make a prediction on the next time step. The prediction is evaluated against the known value. The training set is then expanded to include the known value and the process is repeated.

Note: You should keep the training set window fixed for more efficient training so that every time you add a new observation to the training set, you remove the observation from the beginning of the set.

This process provides a more robust estimation of how the model will perform in practice. However, it comes at the computation cost of creating so many models. This is acceptable if the data is small or if the model is simple, but could be an issue at scale.

Walk-forward validation is the gold standard of time series model evaluation and is recommended for your own projects.

First, create a test data point for each HORIZON step.

python

```
test_shifted = test.copy()

for t in range(1, HORIZON):
    test_shifted['load'+str(t)] = test_shifted['load'].shift(-t, freq='H')

test_shifted = test_shifted.dropna(how='any')
test_shifted.head(5)
```

		load	load+1	load+2
2014-12-30	00  0000	0.33	0.29	0.27
2014-12-30	01  0000	0.29	0.27	0.27
2014-12-30	02  0000	0.27	0.27	0.30
2014-12-30	03  0000	0.27	0.30	0.41
2014-12-30	04  0000	0.30	0.41	0.57

The data is shifted horizontally according to its horizon point.

Now, make predictions on your test data using this sliding window approach in a loop the size of the test data length:

python

```
%%time
training_window = 720 # dedicate 30 days (720 hours) for training

train_ts = train['load']
test_ts = test_shifted

history = [x for x in train_ts]
```

```

history = history[(-training_window):]

predictions = list()

order = (2, 1, 0)
seasonal_order = (1, 1, 0, 24)

for t in range(test_ts.shape[0]):
    model = SARIMAX(endog=history, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit()
    yhat = model_fit.forecast(steps = HORIZON)
    predictions.append(yhat)
    obs = list(test_ts.iloc[t])
    # move the training window
    history.append(obs[0])
    history.pop(0)
    print(test_ts.index[t])
    print(t+1, ': predicted =', yhat, 'expected =', obs)

```

You can watch the training occurring:

2014-12-30 000000 1 : predicted = [0.32 0.29 0.28] expected = [0.32945389435989236, 0.2900626678603402, 0.2739480752014323]

2014-12-30 010000 2 : predicted = [0.3 0.29 0.3 ] expected = [0.2900626678603402, 0.2739480752014323, 0.26812891674127126]

2014-12-30 020000 3 : predicted = [0.27 0.28 0.32] expected = [0.2739480752014323, 0.26812891674127126, 0.3025962399283795]

Now you can compare the predictions to the actual load:

```

python
eval_df = pd.DataFrame(predictions, columns=['t'+str(t) for t in range(1, len(predictions)+1)])
eval_df['timestamp'] = test.index[0:len(test.index)-HORIZON+1]
eval_df = pd.melt(eval_df, id_vars='timestamp', value_name='prediction', var_name='t')
eval_df['actual'] = np.array(np.transpose(test_ts)).ravel()
eval_df[['prediction', 'actual']] = scaler.inverse_transform(eval_df[['prediction', 'actual']])
eval_df.head()

```

	timestamp	h	prediction	actual	
0	2014-12-30 00	0000	t+1	3,008.74	3,023.00

	timestamp	h	prediction	actual
1	2014-12-30	01  0000	t+1	2,955.53
2	2014-12-30	02  0000	t+1	2,900.17
3	2014-12-30	03  0000	t+1	2,917.69
4	2014-12-30	04  0000	t+1	2,946.99

Observe the hourly data's prediction, compared to the actual load. How accurate is this?

Check the accuracy of your model by testing its mean absolute percentage error (MAPE) over all the predictions.

### Show me the math

$$MAPE = \frac{1}{n} \sum_{t=1}^n \left| \frac{actual_t - predicted_t}{actual_t} \right|$$

MAPE is used to show prediction accuracy as a ratio defined by the above formula. The difference between  $actual_t$  and  $predicted_t$  is divided by the  $actual_t$ . "The absolute value in this calculation is summed for every forecasted point in time and divided by the number of fitted points n." [wikipedia](#)

If this equation is expressed in code:

```
python
```

```
if(HORIZON > 1):
    eval_df['APE'] = (eval_df['prediction'] - eval_df['actual']).abs() / eval_df['actual']
    print(eval_df.groupby('h')['APE'].mean())
```

You can calculate one step's MAPE:

```
python
```

```
print('One step forecast MAPE: ', (mape(eval_df[eval_df['h'] == 't+1']['predic
```

One step forecast MAPE: 0.5570581332313952 %

And while you're at it, print the multi-step forecast MAPE:

python

```
print('Multi-step forecast MAPE: ', mape(eval_df['prediction'], eval_df['actual']))
```

Multi-step forecast MAPE: 1.1460048657704118 %

A nice low number is best: consider that a forecast that has a MAPE of 10 is off by 10%.

But as always, it's easier to see this kind of accuracy measurement visually, so let's plot it:

python

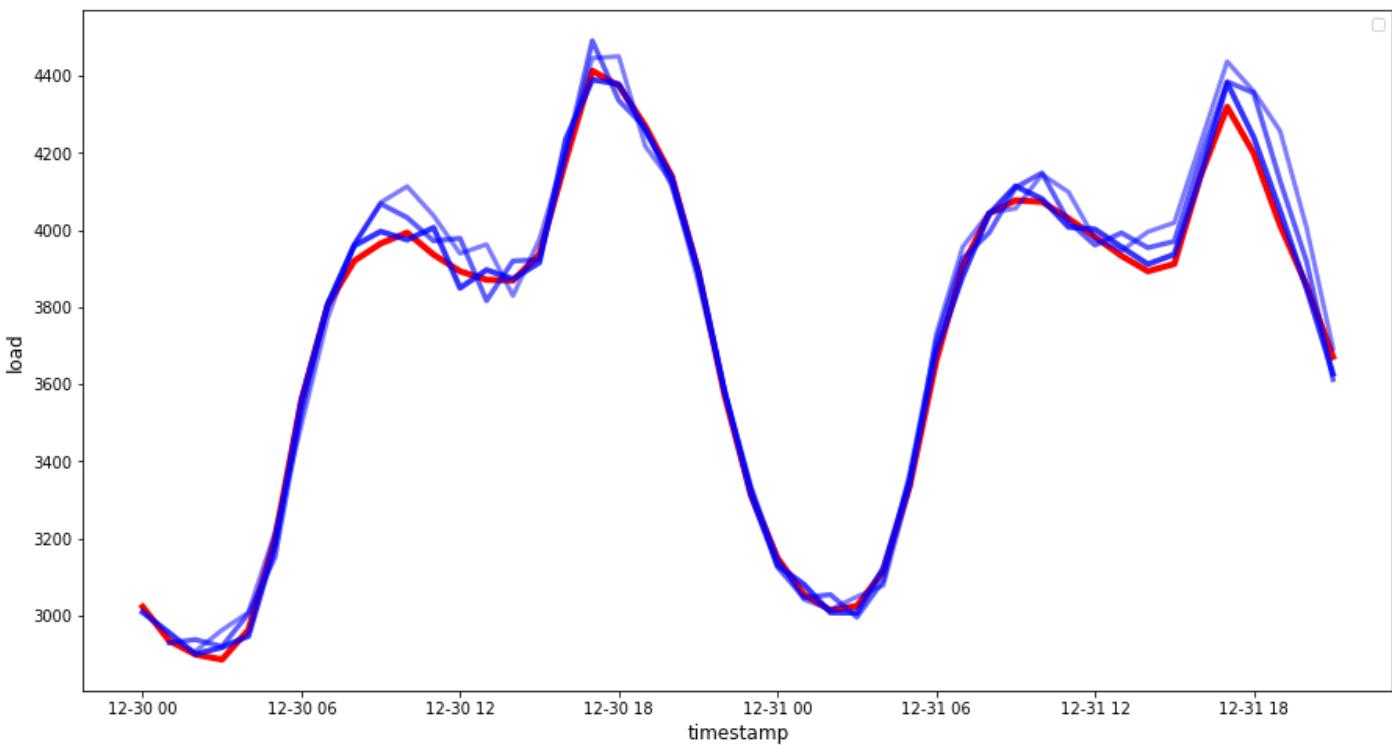
```
if(HORIZON == 1):
    ## Plotting single step forecast
    eval_df.plot(x='timestamp', y=['actual', 'prediction'], style=['r', 'b'])

else:
    ## Plotting multi step forecast
    plot_df = eval_df[(eval_df.h=='t+1')][['timestamp', 'actual']]
    for t in range(1, HORIZON+1):
        plot_df['t+'+str(t)] = eval_df[(eval_df.h=='t+'+str(t))]['prediction']

    fig = plt.figure(figsize=(15, 8))
    ax = plt.plot(plot_df['timestamp'], plot_df['actual'], color='red', linewidth=2)
    ax = fig.add_subplot(111)
    for t in range(1, HORIZON+1):
        x = plot_df['timestamp'][(t-1):]
        y = plot_df['t+'+str(t)][0:len(x)]
        ax.plot(x, y, color='blue', linewidth=4*math.pow(.9,t), alpha=math.exp(-t))

    ax.legend(loc='best')

plt.xlabel('timestamp', fontsize=12)
plt.ylabel('load', fontsize=12)
plt.show()
```



A very nice plot, showing a model with good accuracy. Well done!

## 🚀Challenge

---

Dig into the ways to test the accuracy of a Time Series Model. We touch on MAPE in this lesson, but are there other methods you could use? Research them and annotate them. A helpful document can be found [here](#)

## Post-lecture quiz

---

## Review & Self Study

---

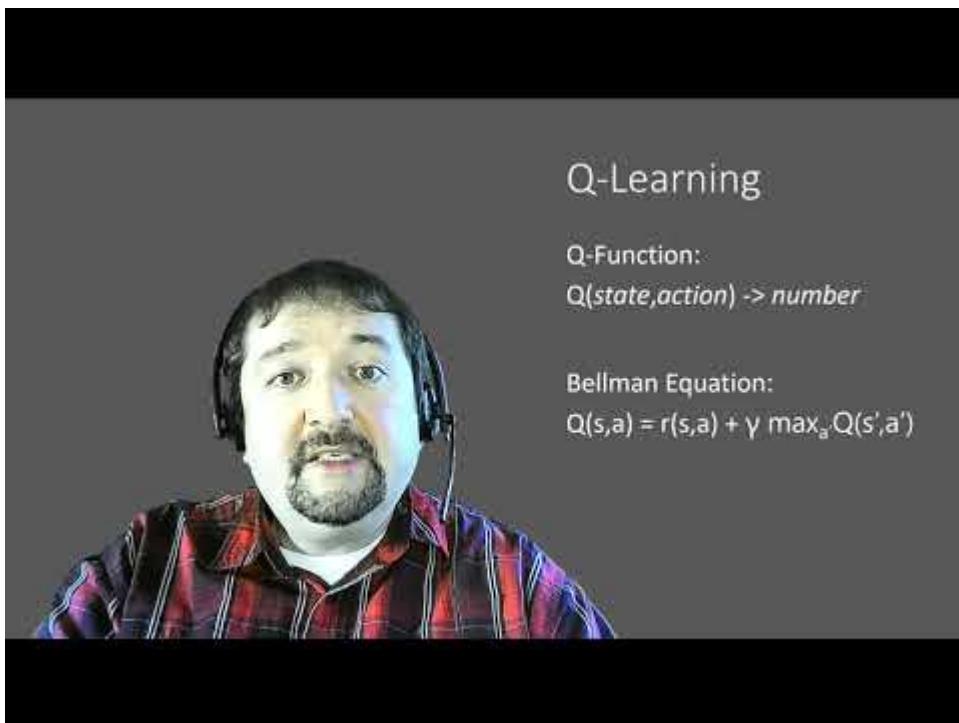
This lesson touches on only the basics of Time Series Forecasting with ARIMA. Take some time to deepen your knowledge by digging into [this repository](#) and its various model types to learn other ways to build Time Series models.

## Assignment

---

[A new ARIMA model](#)

# Introduction to Reinforcement Learning and Q-Learning



Click the image above to hear Dmitry discuss Reinforcement Learning

## Pre-lecture quiz

In this lesson, we will explore the world of **Peter and the Wolf**, inspired by a musical fairy tale by a Russian composer, Sergei Prokofiev. We will use **Reinforcement Learning** to let Peter explore his environment, collect tasty apples and avoid meeting the wolf.

## Prerequisites and Setup

In this lesson, we will be experimenting with some code in Python. So you are expected to be able to run the Jupyter Notebook code from this lesson, either on your computer, or somewhere in the cloud.

You can open the lesson notebook and continue reading the material there, or continue reading here, and run the code in your favorite Python environment.

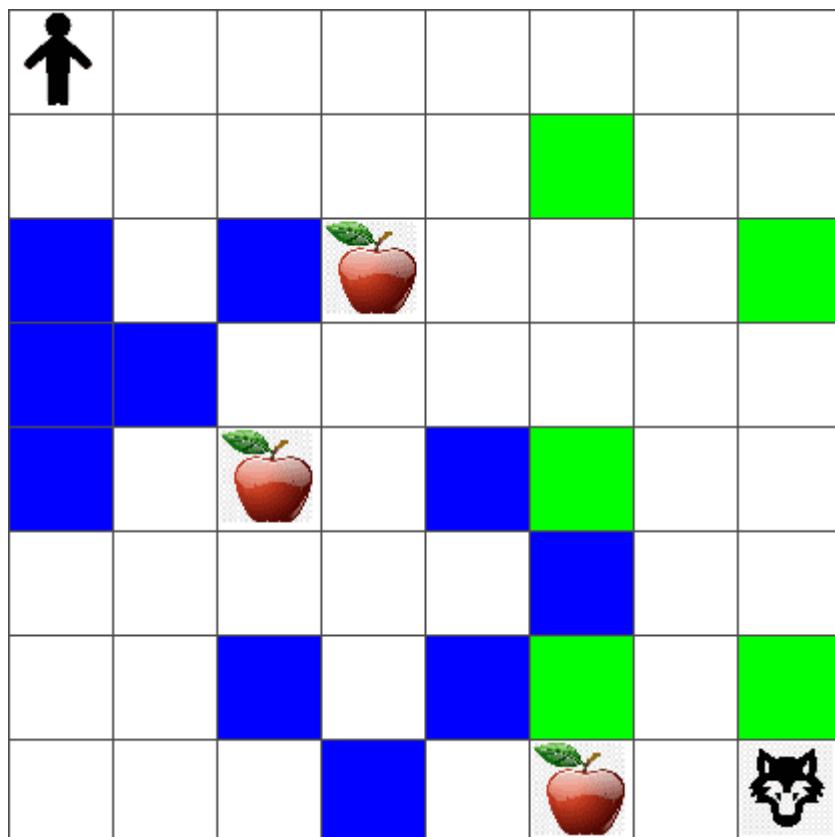
**Note:** If you are opening this code from the cloud, you also need to fetch `rlboard.py` file, because notebook code uses it. Put it into the same directory with the notebook.

## Introduction

**Reinforcement Learning (RL)** is a learning technique that allows us to learn an optimal behavior of an **agent** in some **environment** by running many experiments. An agent in this environment should have some **goal**, defined by a **reward function**.

## The Environment

For simplicity, let's consider Peter's world to be a square board of size `width x height`, like this:



Each cell in this board can either be:

- **ground**, on which Peter and other creatures can walk
- **water**, on which you obviously cannot walk
- **a tree or grass** - a place where you can take some rest
- **an apple**, which represents something Peter would be glad to find in order to feed himself
- **a wolf**, which is dangerous and should be avoided

There is a separate Python module, `rlboard.py`, which contains the code to work with this environment. Because this code is not important for understanding our concepts, we will just import the module and use it to create the sample board (code block 1):

python

```
from rlboard import *

width, height = 8,8
m = Board(width,height)
m.randomize(seed=13)
m.plot()
```

This code should print the picture of the environment similar to the one above.

## Actions and Policy

---

In our example, Peter's goal would be to find an apple, while avoiding the wolf and other obstacles. To do this, he can essentially walk around until he finds an apple. Therefore, at any position he can choose between one of the following actions: up, down, left and right. We will define those actions as a dictionary, and map them to pairs of corresponding coordinate changes. For example, moving right (`R`) would correspond to a pair `(1, 0)`. (code block 2)

python

```
actions = { "U" : (0,-1), "D" : (0,1), "L" : (-1,0), "R" : (1,0) }
action_idx = { a : i for i,a in enumerate(actions.keys()) }
```

The strategy of our agent (Peter) is defined by so-called **policy**. A policy is a function that returns the action at any given state. In our case, the state of the problem is represented by the board, including the current position of the player.

The goal of reinforcement learning is to eventually learn a good policy that will allow us to solve the problem efficiently. However, as a baseline, let's consider the simplest policy called **random walk**.

## Random walk

---

Let's first solve our problem by implementing a random walk strategy. With random walk, we will randomly choose the next action from allowed ones, until we reach the apple (code block 3).

python

```
def random_policy(m):
    return random.choice(list(actions))

def walk(m,policy,start_position=None):
    n = 0 # number of steps
    # set initial position
    if start_position:
        m.human = start_position
    else:
        m.random_start()
    while True:
        if m.at() == Board.Cell.apple:
            return n # success!
        if m.at() in [Board.Cell.wolf, Board.Cell.water]:
            return -1 # eaten by wolf or drowned
        while True:
            a = actions[policy(m)]
            new_pos = m.move_pos(m.human,a)
            if m.is_valid(new_pos) and m.at(new_pos)!=Board.Cell.water:
                m.move(a) # do the actual move
                break
    n+=1

walk(m,random_policy)
```

The call to `walk` should return us the length of corresponding path, which can vary from one run to another. We can run the walk experiment a number of times (say, 100), and print the resulting statistics (code block 4):

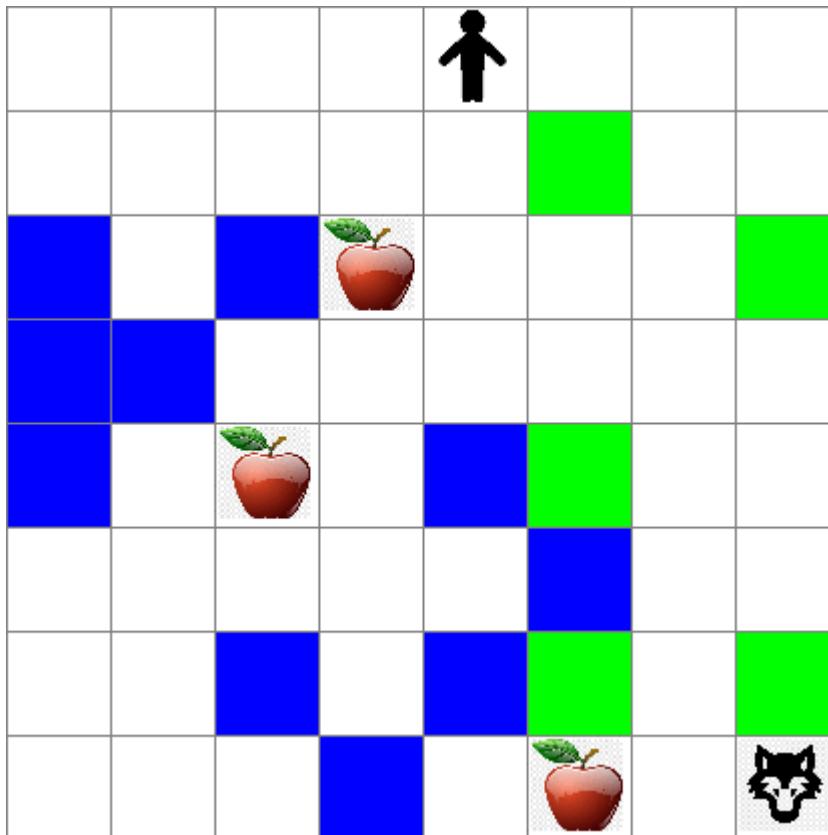
python

```
def print_statistics(policy):
    s,w,n = 0,0,0
    for _ in range(100):
        z = walk(m,policy)
        if z<0:
            w+=1
        else:
            s += z
            n += 1
    print(f"Average path length = {s/n}, eaten by wolf: {w} times")

print_statistics(random_policy)
```

Note that the average length of a path is around 30-40 steps, which is quite a lot, given the fact that the average distance to the nearest apple is around 5-6 steps.

You can also see how Peter's movement looks like during random walk:



## Reward Function

---

To make our policy more intelligent, we need to understand which moves are "better" than others. To do this, we need to define our goal. The goal can be defined in terms of **reward function**, that will return some score value for each state. The higher the number - the better is the reward function.  
(code block 5)

python

```
move_reward = -0.1
goal_reward = 10
end_reward = -10

def reward(m, pos=None):
    pos = pos or m.human
    if not m.is_valid(pos):
        return end_reward
    x = m.at(pos)
    if x==Board.Cell.water or x == Board.Cell.wolf:
        return end_reward
    if x==Board.Cell.apple:
```

```
    return goal_reward  
    return move_reward
```

An interesting thing about reward function is that in most of the cases we are *only given substantial reward at the end of the game*. It means that our algorithm should somehow remember "good" steps that lead to positive reward at the end, and increase their importance. Similarly, all moves that lead to bad results should be discouraged.

## Q-Learning

---

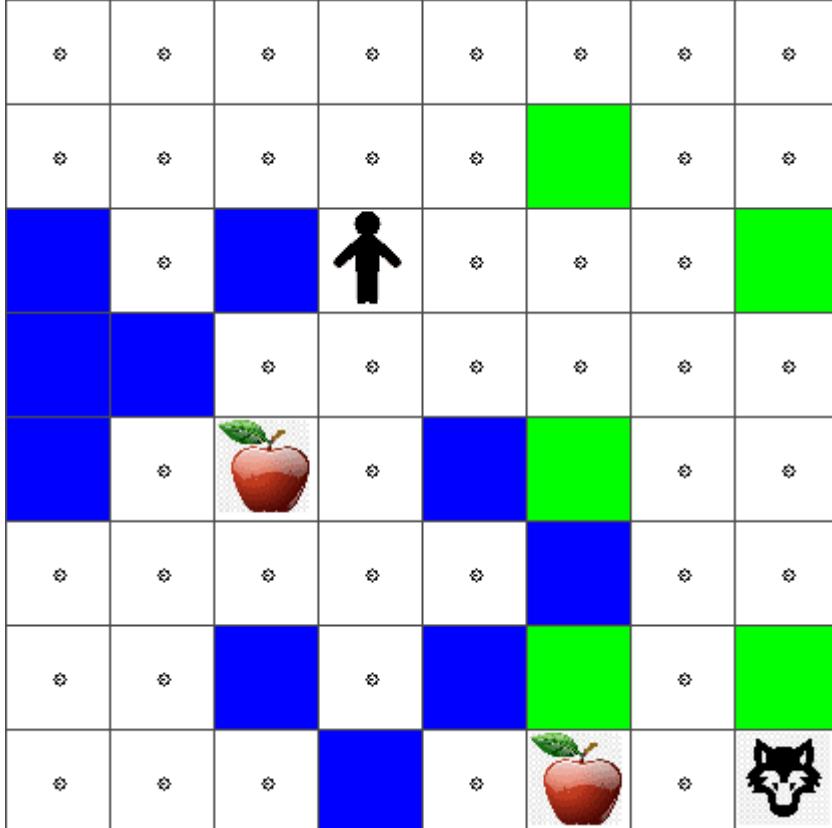
An algorithm that we will discuss here is called **Q-Learning**. In this algorithm, the policy is defined by a function (or a data structure) called **Q-Table**. It records the "goodness" of each of the actions in a given state.

It is called Q-Table because it is often convenient to represent it as a table, or multi-dimensional array. Since our board has dimensions `width x height`, we can represent Q-Table by a numpy array with shape `width x height x len(actions)` : (code block 6)

python

```
Q = np.ones((width,height,len(actions)),dtype=np.float)*1.0/len(actions)
```

Notice that we initially initialize all the values of Q-Table with equal value, in our case - 0.25. That corresponds to the "random walk" policy, because all moves in each state are equally good. We can pass the Q-Table to the `plot` function in order to visualize the table on the board: `m.plot(Q)`.



In the center of each cell there is an "arrow" that indicates the preferred direction of movement. Since all directions are equal, a dot is displayed.

Now we need to run the simulation, explore our environment, and learn a better distribution of Q-Table values, which will allow us to find the path to the apple much faster.

## Essence of Q-Learning: Bellman Equation

---

Once we start moving, each action will have a corresponding reward, i.e. we can theoretically select the next action based on the highest immediate reward. However, in most of the states the move will not achieve our goal or reaching the apple, and thus we cannot immediately decide which direction is better.

It is not the immediate result that matters, but rather the final result, which we will obtain at the end of the simulation.

In order to account for this delayed reward, we need to use the principles of dynamic programming, which allows us to think about our problem recursively.

Suppose we are now at the state  $s$ , and we want to move to the next state  $s'$ . By doing so, we will receive the immediate reward  $r(s,a)$ , defined by reward function, plus some future reward. If we suppose that our Q-Table correctly reflects the "attractiveness" of each action, then at state  $s'$  we

will choose an action  $a$  that corresponds to maximum value of  $Q(s,a')$ . Thus, the best possible future reward we could get at state  $s$  will be defined as  $\max_{a'} Q(s,a')$  (maximum here is computed over all possible actions  $a'$  at state  $s'$ ).

This gives the **Bellman formula** for calculating the value of Q-Table at state  $s$ , given action  $a$ :



Here  $\gamma$  is the so-called **discount factor** that determines to which extent you should prefer current reward over the future reward and vice versa.

## Learning Algorithm

---

Given the equation above, we can now write pseudo-code for our learning algorithm:

- Initialize Q-Table  $Q$  with equal numbers for all states and actions
- Set learning rate  $\alpha \leftarrow 1$
- Repeat simulation many times
  1. Start at random position
  2. Repeat
    1. Select an action  $a$  at state  $s$
    2. Execute action by moving to a new state  $s'$
    3. If we encounter end-of-game condition, or total reward is too small - exit simulation
    4. Compute reward  $r$  at the new state
    5. Update Q-Function according to Bellman equation:  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
    6.  $s \leftarrow s'$
    7. Update the total reward and decrease  $\alpha$ .

## Exploit vs. Explore

---

In the algorithm above, we did not specify how exactly we should choose an action at step 2.1. If we are choosing the action randomly, we will randomly **explore** the environment, and we are quite likely to die often as well as explore areas where we would not normally go. An alternative approach would be to **exploit** the Q-Table values that we already know, and thus to choose the best action (with higher Q-Table value) at state  $s$ . This, however, will prevent us from exploring other states, and quite likely we might not find the optimal solution.

Thus, the best approach is to balance between exploration and exploitation. This can be done by choosing the action at state  $s$  with probabilities proportional to values in Q-Table. In the beginning, when Q-Table values are all the same, it would correspond to a random selection, but as we learn more about our environment, we would be more likely to follow the optimal route while allowing the agent to choose the unexplored path once in a while.

## Python Implementation

---

Now we are ready to implement the learning algorithm. Before that, we also need some function that will convert arbitrary numbers in the Q-Table into a vector of probabilities for corresponding actions: (code block 7)

python

```
def probs(v,eps=1e-4):  
    v = v-v.min()+eps  
    v = v/v.sum()  
    return v
```

We add a few `eps` to the original vector in order to avoid division by 0 in the initial case, when all components of the vector are identical.

The actual learning algorithm we will run for 5000 experiments, also called **epochs**: (code block 8)

python

```
for epoch in range(5000):  
  
    # Pick initial point  
    m.random_start()  
  
    # Start travelling  
    n=0  
    cum_reward = 0  
    while True:  
        x,y = m.human  
        v = probs(Q[x,y])  
        a = random.choices(list(actions),weights=v)[0]  
        dpos = actions[a]  
        m.move(dpos)  
        r = reward(m)  
        cum_reward += r  
        if r==end_reward or cum_reward < -1000:  
            lpath.append(n)
```

```
        break
    alpha = np.exp(-n / 10e5)
    gamma = 0.5
    ai = action_idx[a]
    Q[x,y,ai] = (1 - alpha) * Q[x,y,ai] + alpha * (r + gamma * Q[x+dpo:
n+=1
```

After executing this algorithm, Q-Table should be updated with values that define the attractiveness of different actions at each step. We can try to visualize Q-Table by plotting a vector at each cell that will point in the desired direction of movement. For simplicity, we draw a small circle instead of an arrow head.



## Checking the Policy

---

Since Q-Table lists the "attractiveness" of each action at each state, it is quite easy to use it to define the efficient navigation in our world. In the simplest case, we can select the action corresponding to the highest Q-Table value: (code block 9)

```
python
def qpolicy_strict(m):
    x,y = m.human
    v = probs(Q[x,y])
    a = list(actions)[np.argmax(v)]
    return a

walk(m,qpolicy_strict)
```

If you try the code above several times, you may notice that sometimes it "hangs", and you need to press the STOP button in the notebook to interrupt it. This happens because there could be situations when two states "point" to each other in terms of optimal Q-Value, in which case the agents ends up moving between those states indefinitely.



**Task 1:** Modify the `walk` function to limit the maximum length of path by a certain number of steps (say, 100), and watch the code above return this value from time to time.

**Task 2:** Modify the `walk` function so that it does not go back to the places where it has already been previously. This will prevent `walk` from looping, however, the agent can still end up being "trapped" in a location from which it is unable to escape.

## Navigation

---

Better navigation policy would be the one that we have used during training, which combines exploitation and exploration. In this policy, we will select each action with a certain probability, proportional to the values in Q-Table. This strategy may still result in the agent returning back to the position it has already explored, but, as you can see from the code below, it results in very short average path to the desired location (remember that `print_statistics` runs the simulation 100 times): (code block 10)

```
python
```

```
def qpolicy(m):
    x,y = m.human
    v = probs(Q[x,y])
    a = random.choices(list(actions),weights=v)[0]
    return a

print_statistics(qpolicy)
```

After running this code, you should get a much smaller average path length than before, in the range of 3-6.

## Investigating the learning process

---

As we have mentioned, the learning process is a balance between exploration and exploitation of gained knowledge about the structure of problem space. We have seen that the result of learning (the ability to help an agent to find a short path to the goal) has improved, but it is also interesting to observe how the average path length behaves during the learning process:



What we see here is that at first the average path length increases. This is probably due to the fact that when we know nothing about the environment we are likely to get trapped into bad states, water or wolf. As we learn more and start using this knowledge, we can explore the environment for longer, but we still do not know where the apples are very well.

Once we learn enough, it becomes easier for the agent to achieve the goal, and the path length starts to decrease. However, we are still open to exploration, so we often diverge away from the best path, and explore new options, making the path longer than optimal.

What we also observe on this graph, is that at some point the length increased abruptly. This indicates stochastic nature of the process, and that we can at some point "spoil" the Q-Table coefficients by overwriting them with new values. This ideally should be minimized by decreasing learning rate (i.e. towards the end of training we only adjust Q-Table values by a small value).

Overall, it is important to remember that the success and quality of the learning process significantly depends on parameters, such as leaning rate, learning rate decay and discount factor. Those are often called **hyperparameters**, to distinguish them from **parameters** which we optimize during training (eg. Q-Table coefficients). The process of finding best hyperparameter values is called **hyperparameter optimization**, and it deserves a separate topic.

## Post-lecture quiz

---

## Assignment A More Realistic World

---

# Machine Learning in the Real World

In this curriculum, you have learned many ways to prepare data for training and create machine learning models. You built a series of classic Regression, Clustering, Classification, Natural Language Processing, and Time Series models. Congratulations! Now, you might be wondering what it's all for... what are the real world applications for these models?

While a lot of interest in industry has been garnered by AI, which usually leverages Deep Learning, there are still valuable applications for classical machine learning models. You might even use some of these applications today! In this lesson, you'll explore how eight different industries and subject-matter domains use these types of models to make their applications more performant, reliable, intelligent, and valuable to users.

# Pre-lecture quiz

---



## Credit card fraud detection

We learned about k-means clustering earlier in the course, but how can it be used to solve problems related to credit card fraud?

K-means clustering comes in handy during a credit card fraud detection technique called **outlier detection**. Outliers, or deviations in observations about a set of data, can tell us if a credit card is being used in a normal capacity or if something unusual is going on. As shown in the paper linked below, you can sort credit card data using a k-means clustering algorithm and assign each transaction to a cluster based on how much of an outlier it appears to be. Then, you can evaluate the riskiest clusters for fraudulent versus legitimate transactions.

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.1195&rep=rep1&type=pdf>

## Wealth management

In wealth management, an individual or firm handles investments on behalf of their clients. Their job is to sustain and grow wealth in the long-term, so it is essential to choose investments that perform well.

One way to evaluate how a particular investment performs is through statistical regression. Linear regression is a valuable tool for understanding how a fund performs relative to some benchmark. We can also deduce whether or not the results of the regression are statistically significant, or how much they would affect a client's investments. You could even further expand your analysis using multiple regression, where additional risk factors can be taken into account. For an example of how this would work for a specific fund, check out the paper below on evaluating fund performance using regression.

<http://www.brightwoodventures.com/evaluating-fund-performance-using-regression/>



## Predicting student behavior

Coursera, an online open course provider, has a great tech blog where they discuss many engineering decisions. In this case study, they plotted a regression line to try to explore any correlation between a low NPS (Net Promoter Score) rating and course retention or drop-off.

<https://medium.com/coursera-engineering/controlled-regression-quantifying-the-impact-of-course-quality-on-learner-retention-31f956bd592a>

## Mitigating bias

Grammarly, a writing assistant that checks for spelling and grammar errors, uses sophisticated NLP throughout its products. They published an interesting case study in their tech blog about how they dealt with gender bias in machine learning, which you learned about in our introductory fairness lesson.

<https://www.grammarly.com/blog/engineering/mitigating-gender-bias-in-autocorrect/>

## Retail

---

### Personalizing the customer journey

At Wayfair, a company that sells home goods like furniture, helping customers find the right products for their taste and needs is paramount. In this article, engineers from the company describe how they use ML and NLP to "surface the right results for customers". Notably, their Query Intent Engine has been built to use entity extraction, classifier training, asset and opinion extraction, and sentiment tagging on customer reviews. This is a classic use case of how NLP works in online retail.

<https://www.aboutwayfair.com/tech-innovation/how-we-use-machine-learning-and-natural-language-processing-to-empower-search>

### Inventory management

Innovative, nimble companies like StitchFix, a box service that ships clothing to consumers, rely heavily on ML for recommendations and inventory management. Their styling teams work together with their merchandising teams, in fact: "one of our data scientists tinkered with a genetic algorithm and applied it to apparel to predict what would be a successful piece of clothing that doesn't exist today. We brought that to the merchandise team and now they can use that as a tool."

<https://www.zdnet.com/article/how-stitch-fix-uses-machine-learning-to-master-the-science-of-styling/>

## Managing clinical trials

Toxicity in clinical trials is a major concern to drug makers. How much toxicity is tolerable? In this study, analyzing various clinical trial methods led to the development of a new approach for predicting the odds of clinical trial outcomes. Specifically, they were able to use random forest to produce a classifier that is able to distinguish between groups of drugs.

<https://www.sciencedirect.com/science/article/pii/S2451945616302914>

## Hospital readmission management

Hospital care is costly, especially when patients have to be readmitted. This paper discusses a company that uses ML to predict readmission potential using clustering algorithms. These clusters help analysts to "discover groups of readmissions that may share a common cause".

<https://healthmanagement.org/c/healthmanagement/issuearticle/hospital-readmissions-and-machine-learning>

## Disease management

The recent pandemic has shone a bright light on the ways that machine learning can aid in stopping the spread of disease. In this article, you'll recognize the use of ARIMA, logistic curves, linear regression, and SARIMA. "This work is an attempt to calculate the rate of spread of this virus and thus to predict the deaths, recoveries, and confirmed cases, so that it may help us to prepare better and survive."

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7979218/>

## Ecology and Green Tech

## Forest management

You learned about Reinforcement Learning in previous lessons. It can be very useful when trying to predict patterns in nature. In particular, it can be used to track ecological problems like forest fires and the spread of invasive species. In Canada, a group of researchers used Reinforcement Learning to build forest wildfire dynamics models from satellite images. Using an innovative "spatially spreading process (SSP)", they envisioned a forest fire as "the agent at any cell in the landscape."

"The set of actions the fire can take from a location at any point in time includes spreading north, south, east, or west or not spreading.

This approach inverts the usual RL setup since the dynamics of the corresponding Markov Decision Process (MDP) is a known function for immediate wildfire spread." Read more about the classic algorithms used by this group at the link below.

<https://www.frontiersin.org/articles/10.3389/fict.2018.00006/full>

## Motion sensing of animals

While deep learning has created a revolution in visually-tracking animal movements (you can build your own polar bear tracker here), classic ML still has a place in this task.

Sensors to track movements of farm animals and IoT makes use of this type of visual processing, but more basic ML techniques are useful to preprocess data. For example, in this paper, sheep postures were monitored and analyzed using various classifier algorithms. You might recognize the ROC curve on page 335.

<https://druckhaus-hofmann.de/gallery/31-wj-feb-2020.pdf>

## ⚡ Energy Management

In our lesson on Time Series, we invoked the concept of smart parking meters to generate revenue for a town based on understanding supply and demand. This article discusses in detail how clustering, regression and time series forecasting combined to help predict future energy use in Ireland, based off of smart metering.

[https://www-cdn.knime.com/sites/default/files/inline-images/knime\\_bigdata\\_energy\\_timeseries\\_whitepaper.pdf](https://www-cdn.knime.com/sites/default/files/inline-images/knime_bigdata_energy_timeseries_whitepaper.pdf)

## 💼 Insurance

### Volatility Management

MetLife, a life insurance provider, is forthcoming with the way they analyze and mitigate volatility in their financial models. In this article you'll notice binary and ordinal classification visualizations. You'll also discover forecasting visualizations.

[https://investments.metlife.com/content/dam/metlifecom/us/investments/insights/research-topics/macro-strategy/pdf/MetLifeInvestmentManagement\\_MachineLearnedRanking\\_070920.pdf](https://investments.metlife.com/content/dam/metlifecom/us/investments/insights/research-topics/macro-strategy/pdf/MetLifeInvestmentManagement_MachineLearnedRanking_070920.pdf)

## Fake news detection

Detecting fake news has become a game of cat and mouse in today's media. In this article, researchers suggest that a system combining several of the ML techniques we have studied can be tested and the best model deployed: "This system is based on natural language processing to extract features from the data and then these features are used for the training of machine learning classifiers such as Naive Bayes, Support Vector Machine (SVM), Random Forest (RF), Stochastic Gradient Descent (SGD), and Logistic Regression(LR)."

<https://www.irjet.net/archives/V7/i6/IRJET-V7I6688.pdf>

This article shows how combining different ML domains can produce interesting results that can help stop fake news from spreading and creating real damage; in this case, the impetus was the spread of rumors about COVID treatments that incited mob violence.

## Museum ML

Museums are at the cusp of an AI revolution in which cataloging and digitizing collections and finding links between artifacts is becoming easier as technology advances. Projects such as [In Codice Ratio](#) are helping unlock the mysteries of inaccessible collections such as the Vatican Archives. But, the business aspect of museums benefits from ML models as well.

For example, the Art Institute of Chicago built models to predict what audiences are interested in and when they will attend expositions. The goal is to create individualized and optimized visitor experiences each time the user visits the museum. "During fiscal 2017, the model predicted attendance and admissions within 1 percent of accuracy, says Andrew Simnick, senior vice president at the Art Institute."

<https://www.chicagobusiness.com/article/20180518/ISSUE01/180519840/art-institute-of-chicago-uses-data-to-make-exhibit-choices>

## Marketing

---

## Customer segmentation

The most effective marketing strategies target customers in different ways based on various groupings. In this article, the uses of Clustering algorithms are discussed to support differentiated

marketing. Differentiated marketing helps companies improve brand recognition, reach more customers, and make more money.

<https://ai.inqlne.com/machine-learning-for-marketing-customer-segmentation/>

## Challenge

---

Identify another sector that benefits from some of the techniques you learned in this curriculum, and discover how it uses ML.

## Post-lecture quiz

---

## Review & Self Study

---

The Wayfair Data Science team has several interesting videos on how they use ML at their company. It's worth [taking a look!](#)

## Assignment

---

[A ML scavenger hunt](#)