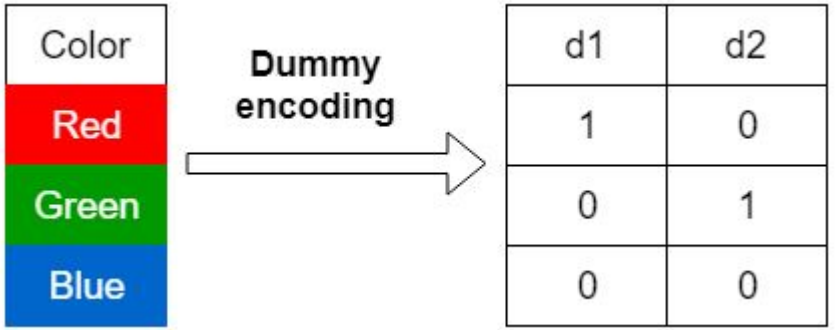
Data Wrangling and Feature Engineering

Agenda - Schedule

1. Identifying “Good” Data
2. Data Cleaning & Normalizing
3. Feature Engineering
4. Break
5. Feature Engineering Lab



We discussed converting numeric into categorical variables, now let’s discuss the reverse.

Agenda - Goals

- ...

Announcements

- TLAB #2 Checkpoint Due 7/9
- Pre-Class Quiz #7 Due 7/14

Data Wrangling

Data Prep

The entire goal of today is to get our data ready for machine learning

Remember, end of the day, machine learning is essentially just a bunch of fancy statistics

That means we need our data to be clean!

Good Data

What does good data look like?

It is:

- consistently formatted (we don't mix numbers and strings if we can help it)
- primarily numerical (unless dealing with language models)
- scaled consistently (we discuss this later)
- minimal/unnecessary outliers are removed
- well labeled

In general, this just means it is not going to have issues and this comes with experience

The exact steps change with each model, today we will discuss **general best practices** for good data

Good Data

To get our data there we will perform **data wrangling** and **feature processing**

Don’t get too caught up in the terminology, these two overlap a lot and are basically the same thing

Data Wrangling focuses on **cleaning** and **standardizing** our data

Feature processing focuses on **creating , removing, or modifying features**

This is prior to **pre-processing for modeling**

Pre-processing tends to do things more closely tied to our model and we’ll give you a taste next session

Wrangle my data?

Yep, we're going to have to herd this data together into something meaningful

Data wrangling (and feature engineering) is more art than science

It takes experience, domain knowledge, and some technical skill

Today, we focus on the intuition

Data Wrangling

In general, data wrangling is the idea of getting raw data ready for machine learning

The key concepts of data wrangling are:

- cleaning
- transforming
- removing outliers

Cleaning Data

Here are some key concepts of cleaning data:

- removing duplicates
- dealing with missing values

Removing Duplicates

This one is pretty easy to understand! If data is duplicated then it will be weighed incorrectly

This is pretty easy to do with data in pandas.

You might want to use something like **df.duplicated()** and **df.drop_duplicates()**

Missing Values

Dealing with missing values is an art, not a science

First, figure out how many missing values there are in each feature:

df.isna().sum()

If there are a lot of missing values, consider dropping that feature

df.drop() or **df.dropna()**

But if there aren't a lot… maybe we can fill in the gaps?

Imputing Missing Values

When we fill in missing values, we call that "**imputing**"

There is no one way. We generally use the method **df.fillna()**

You can either:

- fill in all the missing values with the same value (maybe 0)
- fill in with the average, median, or other statistical measure
- fill in with the value before or after the row

There are merits to each. We should stick to what makes sense for the data

Imputing Missing Values

Missing data means no record? Consider putting in 0

Missing data is due to a forgotten record but is mostly stable? Consider using the mean

Missing data is generally sequential and moves pretty consistently with time? Consider backfill (using value of the next record) or forward fill (using the value of the record prior) to fill in the gap

End of the day, you can try different methods and see what works best for your data

Transforming Data

Here are the key concepts regarding transforming data into something meaningful:

- changing data types
- reshaping data
- converting/encoding categorical data

Changing Data Types

This is straightforward! Have numbers that should be numbers but reading as strings? Convert that to a numerical data type!

Do something like **pd.astype()**

Reshaping Data

Reshaping data is where we can get creative

Remember, the ideal is that each row is a separate record

Maybe instead our data has our variable of interest as columns. Think student performance

Example of wide format:

Student	Math	Literature	PE
A	99	45	56
B	73	78	55
C	12	96	57

Example of long format:

Student	Subject	Score
A	Math	99
A	Literature	45
A	PE	56
B	Math	73
B	Literature	78
B	PE	55
C	Math	12
C	Literature	96
C	PE	57

Reshaping Data

Sure, the wide format is easier for us to understand but it's harder because our model has trouble understanding that "math", "literature", and "pe" are related as one singular thing known as a "subject"

The long format lets us better train our models

Consider using **pd.melt()**

Example of wide format:

Student	Math	Literature	PE
A	99	45	56
B	73	78	55
C	12	96	57

Example of long format:

Student	Subject	Score
A	Math	99
A	Literature	45
A	PE	56
B	Math	73
B	Literature	78
B	PE	55
C	Math	12
C	Literature	96
C	PE	57

Encoding Categorical Variables

Remember, most of our models require the usage of numbers and cannot handle text

In fact, many will refuse to run if there is any string value

Maybe we need the data provided by the string though! How can we still use it?

We encode that data as numerical!

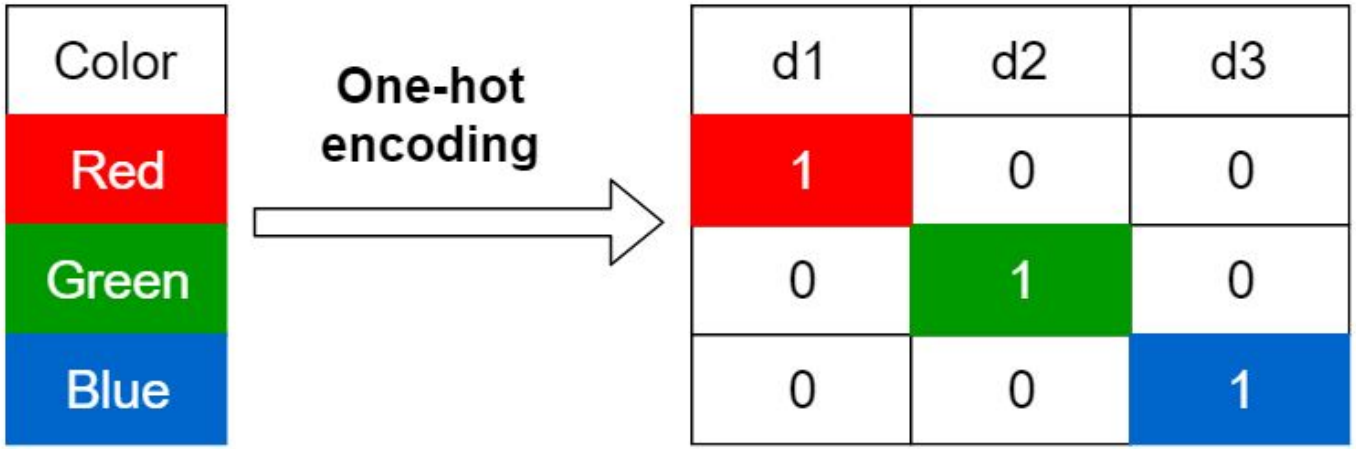
Encoding Methods

There are 2 main methods we will talk about:

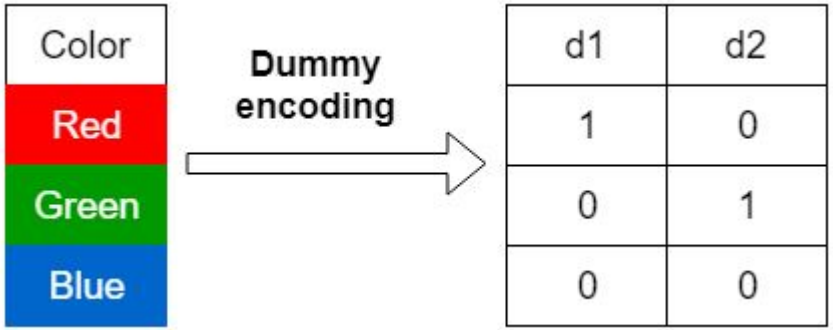
One-Hot Encoding - we turn every category into a ***boolean*** with each row getting a 1 or a 0

Dummy Encoding - the same thing as one-hot encoding but we **leave one variable out** (we'll explain why)

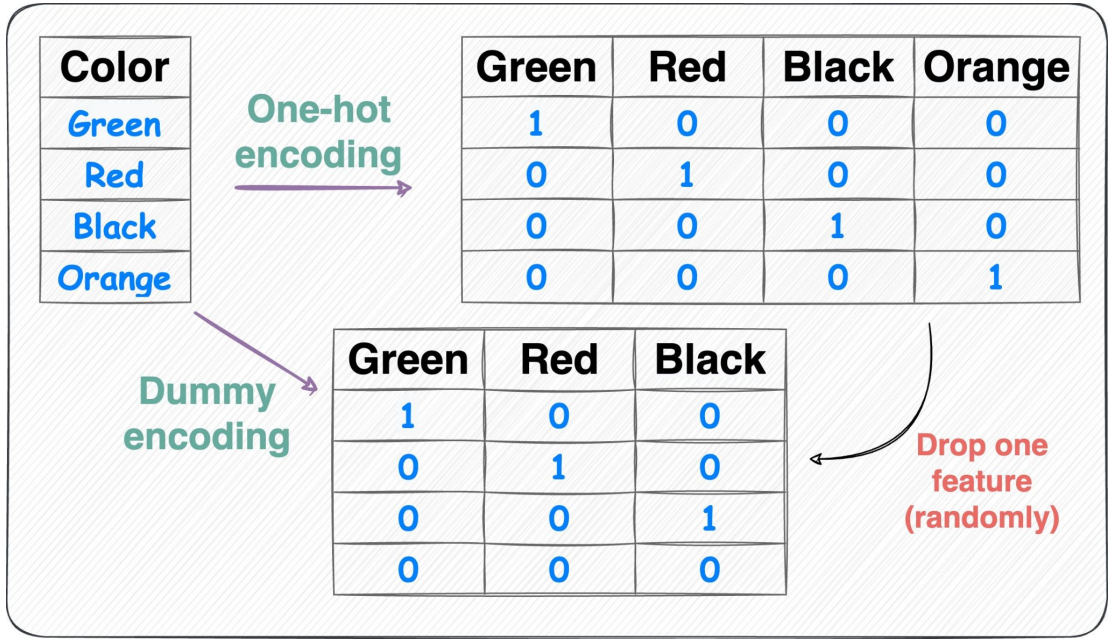
One-Hot Encoding



Dummy Encoding



One-Hot vs Dummy Encoding



One-Hot vs Dummy Encoding

So why choose dummy encoding?

Because of something called **multicollinearity** which basically means that the categories are very closely related to each other and mess up our model

Think of it this way, if there are only 3 options and I know 2 of the options are false, do I need to explicitly state the 3rd option as true?

One-Hot vs Dummy Encoding

No! I can derive that the 3rd option must be true because the other 2 are false

However, if I leave that 3rd option in for the machine learning algorithm then it'll get trapped into thinking that the 3rd option matters more than it does

One-Hot vs Dummy Encoding

To do one-hot encoding in pandas use get_dummies()
https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

pd.get_dummies()

to do dummy encoding:

pd.get_dummies(drop_first=True)

Don’t forget to drop your newly encoded column!

Removing Outliers

Again, this is an art

You can just drop the records that have outliers but use EDA and your best judgment

Maybe the outlier is meaningful!

Okay! Now that our data is in a better place, lets feature engineer

Was this helpful?

Numerical data: Normalization

Send feedback

After examining your data through statistical and visualization techniques, you should transform your data in ways that will help your model train more effectively. The goal of **normalization** is to transform features to be on a similar scale. For example, consider the following two features:

- Feature `X` spans the range 154 to 24,917,482.
- Feature `Y` spans the range 5 to 22.

These two features span very different ranges. Normalization might manipulate `X` and `Y` so that they span a similar range, perhaps 0 to 1.

Normalization provides the following benefits:

- Helps models *converge more quickly* during training. When different features have different ranges, gradient descent can "bounce" and slow convergence. That said, more advanced optimizers like Adagrad and Adam protect against this problem by changing the effective learning rate over time.

For more information on data normalization & outlier removal, read up on the following: https://developers.google.com/machine-learning/crash-course/numerical-data/normalization

Feature Engineering

What is Feature Engineering?

Feature engineering is the process of manipulating your data to make the features more useful

There are three major components:

- adding features
- removing features
- modifying features

All of these take experience, knowledge, and understanding of the data

Removing Features

This is the easiest topic so we'll start here!

Removing features is basically just saying "this column is useless, let's get rid of it"

Maybe it's something like customer IDs (no numerical meaning) or the color of the sky

But also, maybe there's a lot of missing data or the data is extremely skewed

Simply put, we can just use **df.drop()** on these things

Adding Features - Interaction

This is more about feature extraction than really creating features out of thin air

Maybe we have two features like… **sq. foot** and **cost**

These 2 features are probably pretty closely related

Remember **multicollinearity**? We do not want to undermine the variables with each other!

It makes them less useful so why don't we repurpose these 2 features?

Adding Features

We can **add** a feature called "**price per sq. foot**" which is a calculation of price/sq. foot which gives us an idea of what both features bring without having both of them conflict

We can now remove those two columns making our feature space smaller

Adding Features - Polynomial/Interaction

Likewise, maybe instead of dividing our features we **multiply** them

For instance, square footage (it returns!)

Maybe we have length and width, it makes sense to convert that to **square footage** if we are interested in area

Or we have volume and mass which can turn into **density**

Think about the data we have and how to get more information out of it

Adding Features and then some!

There are many more ways to add features, the goal is to enrich our model by **decreasing** the **overall feature space** (combining multiple features into one) while avoid **overfitting**

As we get into more advanced models, we will discuss feature engineering specific to that model

Modifying Features

Modifying features is where we can get a little interesting… some transformations include:

- Log Transformation
- Square Root transformation
- Creating Binary Features

Log Transformation

Modifying features is where we can get a little interesting… two very common transformations include:

- Log Transformation
- Creating Binary Features

Binary Features

Creating Binary Features is a great way of simplifying our features

This also helps if the relationship is non-linear and more-so focused on categorical features

Maybe we just care if someone is “old” or “not old”.

We can do something like: df['is_old'] = (df['age'] > 60).astype(int)

To create a binary “is old” vs “not old”

Log Transformation

The **log transformation** help **normalize** our data

The overall goal of Log Transformation is to bring the relationship between the **predictor** and the **output** variable to become more linear, the same is true of the square root transformation

Log Transformation is useful for when values grow very quickly or disproportionately

Let's go over this article for more detail:

https://www.codecademy.com/article/data-transformations-for-multiple-linear-regression

Other Transformations

Other transformations include the **Square Root Transformation, Box-Cox transformation, Arcsine Transformation**, and more!

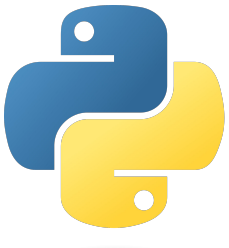
The goal is always to help us better understand the relationship between our **predictor variables** and **output variables**

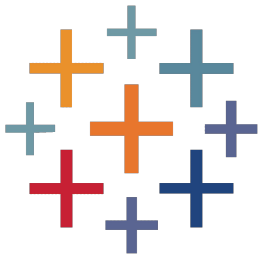
Over time you will learn which ones to use where but for now stick to **log transformations** and **binary**

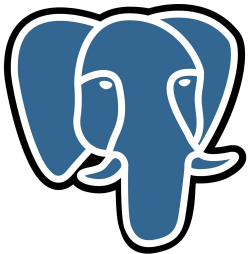
Feature Engineering Lab

Feature Engineering Lab

Download and complete the **Feature Engineering lab** in your pod groups.





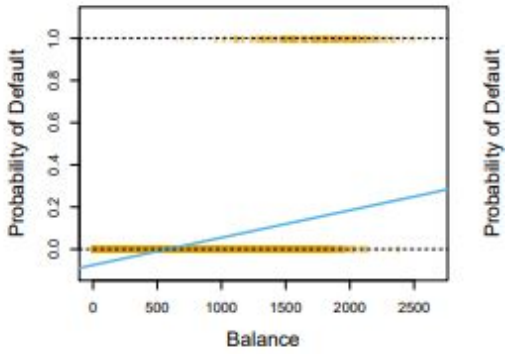


Next Week

Next Week we will learn about a new machine learning algorithm called **logistic regression.**

Pre-class content for **Week 4** will be due **6/25**.

Read the springer textbook (i*t will be difficult!*)
Then, watch the StatQuest videos (*it will be a welcome reprieve*).



If you understand what you're doing, you're not learning anything. - Anonymous