Machine Learning Foundations UNIT 2

Table of Contents

- Watch: Visualize Your Data: Univariate Plotting
- Tool: Plotting Tip Sheet
- Code: Using Seaborn and Matplotlib To Visualize Data
- Watch: Visualize Your Data: Bivariate Plotting
- Code: Bivariate Plotting With Seaborn
- Read: Correlation, Covariance, and Mutual Information
- Module Wrap-up: Explore Your Data

Module Introduction: Find Outliers and Missing Data

- Watch: Outliers
- Read: Common Statistics Refresher
- Assignment: Detecting and Replacing Outliers
- Read: Outlier Detection Methods
- Watch: Missing Data
- Assignment: Finding and Replacing Missing Data
- Assignment: Unit 2 Assignment - Building a Modeling Data Set
- Assignment: Unit 2 Assignment - Written Submission
- Module Wrap-up: Find Outliers and Missing Data
- Assignment: Lab 2 Assignment

Homepage

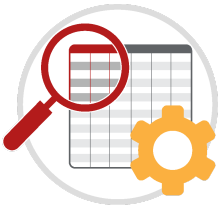
Unit 2 Overview

Video Transcript

We are now going to start digging into the model development lifecycle, and we'll be starting on the data preparation and data exploration phases. Data preparation is the unsung hero of machine learning. Good data prep can be as powerful as some of the best machine learning algorithms out there. Our learning objectives this week will be to cover many important elements of the data preparation and exploration phases. This includes building a data metrics suitable for ML applications, learning how to create an appropriate label for supervised learning, creating features that are suitable for ML applications, understanding your data through exploratory analysis, and then finally identifying and fixing issues with your data.

What You'll Do:

- **Build a dataset suitable for ML applications**
- **Create an appropriate label for supervised learning**
- **Create features that are suitable for ML applications**
- **Use exploratory analysis to understand your data**
- **Identify and fix issues with your data**



Unit Description

One of the most important steps in the machine learning process is data understanding and preparation. It is a step taken prior to model training and it is important to complete because you need to ensure the data selected for your model is appropriate to solve the problem.

In this unit, Mr. D'Alessandro focusses on taking raw data, analyzing and organizing it, and preparing it for the next stage of the machine learning process, which is modeling. You will practice identifying examples, and their features and labels, to prepare for supervised learning. You will also practice organizing your data into a data matrix. You will learn about "feature engineering", which will allow you to transform your data into a format that is most appropriate for your specific model.

The data set you will be introduced to this unit for coding activities is known as the Adult Data Set*.

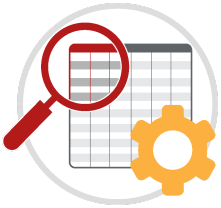
*Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Unit 2 Overview
Video Transcript

We are now going to start digging into the model development lifecycle, and we'll be starting on the data preparation and data exploration phases. Data preparation is the unsung hero of machine learning. Good data prep can be as powerful as some of the best machine learning algorithms out there. Our learning objectives this week will be to cover many important elements of the data preparation and exploration phases. This includes building a data metrics suitable for ML applications, learning how to create an appropriate label for supervised learning, creating features that are suitable for ML applications, understanding your data through exploratory analysis, and then finally identifying and fixing issues with your data.

What You'll Do:

- **Build a dataset suitable for ML applications**
- **Create an appropriate label for supervised learning**
- **Create features that are suitable for ML applications**
- **Use exploratory analysis to understand your data**
- **Identify and fix issues with your data**



Unit Description

One of the most important steps in the machine learning process is data understanding and preparation. It is a step taken prior to model training and it is important to complete because you need to ensure the data selected for your model is appropriate to solve the problem.

In this unit, Mr. D'Alessandro focusses on taking raw data, analyzing and organizing it, and preparing it for the next stage of the machine learning process, which is modeling. You will practice identifying examples, and their features and labels, to prepare for supervised learning. You will also practice organizing your data into a data matrix. You will learn about "feature engineering", which will allow you to transform your data into a format that is most appropriate for your specific model.

The data set you will be introduced to this unit for coding activities is known as the Adult Data Set*.

*Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Back to Table of Contents

Tool: Unit 2 Glossary

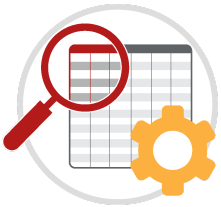
Most of the new terms you will learn about throughout the unit are defined and described in context, but there are a few vital terms that are likely new to you but undefined in the course videos. While you won't need to know these terms until later in the unit, it can be helpful to glance at them briefly now. Click on the link to the right to view and download the new terms for this unit.

Download the Tool

Download and save this **Unit 2 Glossary Tool** to use as a reference as you work through the unit and encounter unfamiliar terms.

Back to Table of Contents

Module Introduction: Build Your Data Matrix



Mr. D'Alessandro has expressed how machine learning is a new paradigm of programming that allows computer programs to learn from data. First, the data must be prepared and formatted for the model. The most common data format for machine learning datasets is data matrices.

In this module, Mr. D'Alessandro will explain how to identify features and labels for a business problem and how to transform these features and labels into a data matrix. It's important to have the right sample data for your model to ensure your machine learning model has the best chance of solving the business problem at hand. Throughout this module, you will practice applying filters and building your own balanced data set.

Back to Table of Contents

Watch: Summary of a Data Matrix

An important step of the machine learning process is data understanding and preparation. Raw data is generally not formatted in a way that is compatible with most machine learning models, so a machine learning engineer needs to transform data with a model in mind. Data preparation for supervised learning involves properly identifying examples, and their corresponding features and labels. In this video, Mr. D'Alessandro explains how data is organized into a data matrix to prepare it for the next stage of the machine learning process: modeling.

Video Transcript

One of the most time-consuming aspects of model development will be data preparation. Outside of academic problem sets, data will come to you in many formats. Data generally isn't generated or stored in a format that is compatible for most machine learning models. Thus, it is up to the model developer to transform the data appropriately. But beyond pure transformations, the model developer also has to prepare data for model performance, where prediction, accuracy, and generalization are your goals. In machine learning, we need to be able to switch between mathematical and programming concepts. Machine learning starts with math. Let's review the two key mathematical constructs that represent data. These are vectors and matrices. We use programming constructs to define these. A vector is just a [1-dimensional] array, such as what is commonly used in NumPy. A matrix is also an array, but it has two dimensions. We can think of it as a set of N same-sized arrays stacked on top of each other. Here, we show more explicitly how the mathematical and programming concepts relate to each other. The data object here is a typical Pandas DataFrame. It has N rows and K columns, where each row and column is a unique array. In our work, we'll do operations on both row and column arrays. For instance, when we make predictions using a trained model, the prediction will be done on a row array. But when we do data preparation work, we often do operations on individual column arrays. Let's now review key items with the DataFrame or matrix. We'll call the individual rows of the matrix "examples." Each row corresponds to the individual entities or units of your analysis. One of the first things you need to define when building a model is who or what you'll be modeling. Defining what makes up your tables' example should be done in the early problem formulation stage. Your data preparation will then be centered on finding the appropriate sample of these units and defining the attributes associated with these units. With the units defined, we would then start to define the columns. These columns can go by several names, such as attributes, features, or variables. We'll use "features" throughout this program. The process of defining and coding up the features is called "feature engineering" There is a lot to this process, so we'll reserve the details for a later module. In supervised learning, one of the columns has a special status. This is the "label," which again is the item we are trying to predict. When preparing data for supervised learning, our main

goal will be to ensure we have a consistent set of features and a label for each example. On top of considering the units of analysis and the features you'll need for modeling, one key decision to make is how many features and records are sufficient. There's no right answer to this question. One general rule is that more is better of each. But the challenge is there is always usually diminishing returns on data set size. Obtaining more examples or engineering more features usually comes at some cost. Doubling either dimension won't necessarily double performance. The right way to answer this question is to perform empirical evaluations. The experimentation in model selection techniques we'll cover will set you up to be able to run such empirical evaluations.

Back to Table of Contents

Watch: Who (or What) Are You Modeling?

It's very important to precisely define your unit of analysis in your initial problem statements and this needs to be considered in all steps of the model building process. Mr. D'Alessandro gives some example problem statements. In this video, we will use “unit of analysis” to refer to a real-life representation of an example.

Video Transcript

Defining your unit of analysis is a very important topic, so I want to dig into this a little more. It's important to precisely define this in your initial problem statement. The unit of analysis needs to be considered in all steps of the model building process. Let's start by giving some example problem statements. In fraud detection, we might say, what is the likelihood that this transaction is fraud? In marketing, we might ask, will this reader click on the ad I'm showing? For recommendations, what are the best tweets to display to this Twitter user? In something we call telemetry, what is the likelihood that this engine will fail in the next 24 hours? Your problem's unit of analysis should follow from the problem statement. Each of the problem statements that I just gave can be solved with a binary classification model and each has a different unit of analysis that is specified within the problem statement. I tried to emphasize the particular words that represented the units. For instance, with the fraud, I emphasized the word 'transaction.' This is what I mean generally by a unit of analysis. The data matrix you will build to train a machine learning model will contain a sample of these units. Each of these samples are what we have been traditionally calling the examples of your data. In the middle two that I specified, I highlighted two nouns, such as 'tweet' and 'Twitter user' or 'reader' and 'ad.' In many applications, the unit is a pair of items, again, a reader in a specific ad or a user in a specific tweet. In those cases, your data matrix will be a sample of specific reader-ad or user-tweet pairs. The database you are working with will likely have special ID columns that correspond to your unit of analysis. All of your data frames or tables should include this ID or set of IDs. When you merge or join individual data frames together you'll need these IDs as your joint keys. Just remember that the ID columns are not features. When you pass data frames into different Scikit-learn algorithms, it will use all of the columns that you send it. So you often have to explicitly filter out the ID columns so you don't inadvertently model using them.

Back to Table of Contents

Assignment: Applying Filters

Sometimes you may may want to obtain portions of data from your data set for analysis. You can apply filters to your data set to accomplish this. In this exercise, you will practice applying a variety of filtering techniques available to you from the Pandas package. You will learn how to apply some of these filtering techniques to your data matrix, such as extracting specific columns (features), rows (examples), and data values, for the purpose of obtaining the data needed for your analysis. In this unit's coding exercises and assignments, you will work with the **Adult Data Set**, which is a data set that contains Census information from 1994.

This exercise will be graded. Graded exercises will contain the bull's eye icon in the exercise title.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

For general information on using Jupyter Notebooks, you can expand the right-hand panel adjacent to the Notebook by clicking on the icon.

Note: To execute a cell in a Jupyter Notebook, press shift-enter or select **Run** in the menu bar. Prolonged inactivity or navigating to a different page will cause the notebook session to timeout. To resume your work where you left off, be sure to re-run all of the notebook cells in order, starting again at the very beginning.

This exercise will be auto-graded.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

Watch: Sampling Techniques

When choosing data for your model to generalize well to new data, you have to select a sample that reflects the larger distribution of data. There are many things to keep in mind to perform this step successfully, such as finding balance and representing each group. Watch as Mr. D'Alessandro explains the importance of having balanced data. He exemplifies with two separate contexts for sampling: the population that we want to model, and managing the number of examples we will use to model. He also uses examples from previous videos to examine how to define and specify a sampling population.

Video Transcript

In this video, we'll discuss sampling, which is the process of extracting subsets of examples from some available universe and data. We'll cover two separate contexts for sampling. The first concerns the population that we want to model. The second concerns managing the number of examples we will use to model. In a previous video, we discussed the step of defining the unit of analysis. To define a sample population, we often need to add a bit more description, though, than just the unit. Specifically, are there any attributes of those units that should be considered when sampling them from the larger pool of units? This, again, should be specified during problem formulation. Let's revisit earlier examples but add specificity around the sampling population. In each example, I'll add an extra attribute that serves as a filter on the full set of units. These attributes should be tied to the core problem you are trying to solve. Oftentimes you don't want your model to apply to every person in every situation. Sometimes you may have different models for different segments of the population, like new versus more tenured customers. So, some examples. For fraud detection, instead of saying, "What is the likelihood of all transactions being fraud?" we might say, "What is the likelihood that a web-based transaction is fraud?" For marketing, we might think, "Will this non-subscribing reader click on the ad I'm showing?" Similarly, with recommendations, "What are the best tweets to display to this new Twitter user?" We can illustrate this concept, too, in a picture. Each circle represents a different population. In this case, the smaller circles are subpopulations of the larger circles. When building a model, your problem statement may dictate using different sample populations. In most cases, you'll be working with your company's customers, but for applications like marketing, you might actually be focused on non-customers, so the whole world is your modeling population. In the marketing example I covered, I specified non-subscribing readers. We can think of those as the set of customers where their subscription status is set to false. Through separate analysis, we might find that subscribers and non-subscribers respond to ads differently for various reasons. Thus, to make effective predictions, it might be better to build a model specific to each of those populations. One reason we want to be very precise about our sample population ties back to our core goal of

generalization. A model train on a given sample whose population is defined by having certain attributes may not generalize and perform well on a sample with different attributes. We can use this picture to illustrate this point better. Within this set of customers from the previous illustration, I have added a second population. This population is customers with a new attribute, Z. Notice there is some overlap with the X sample, and that's OK. If I build a model on customers with Attribute X, this model may not perform well on customers with Attribute Z. X and Z can be the home country of the customer or whether they subscribe to a premium service or not. Within the set of customers with Attribute X, I show two smaller green circles. These are merely subsamples of the same population. We can assume that the two green circles represent distinct sets of customers, but we should expect the two samples to have the same distribution of features. This is what we call IID samples, or independent and identically distributed samples. When taking samples of data from within a population, this idea of IID is critical. Fortunately, it's pretty easy to achieve it by just using a random number generator to select examples from a population. So now we've illustrated a few key points. We always want to be crisp and clear about who should be in our sample, and we want to make sure that we're training a model using the same population where the model will be making predictions. We can then reduce sampling to two high-level steps. The first, again, is to define the population in terms of unit of analysis and key attributes of those units. The second is to do a random selection from that population, where the sample size is determined by what is available or what is minimally needed to get sufficient generalization performance. I always like to call out fairness concerns where they apply to individual model and data design decisions we have to make throughout the model development process. We've been discussing sampling in terms of populations and, to some degree, sample size. One general rule of thumb is that more examples lead to better models. There may be diminishing returns, so we might imagine the sample size versus performance curve to look something like this. Now, the sample population I have to find might have a few subpopulations, where each subpopulation occurs at different rates. Examples where this is usually true are a person's race, religion, or sexual orientation. We should always aim to build models that treat people in different protected classes equally. It is always possible, though, that because certain classes are underrepresented in the data, the models are inaccurate for those particular groups. If the data is available, I always advise evaluating the model on the different groups to test for this. If the model indeed underperforms for smaller subpopulations, one trustworthy way to solve this is to resample the training data so that each group is equally represented. In this case, we might want to deliberately invest in more examples from Subpopulation A in order to lift the performance. In this example, the expected performance for Group A would then be comparable to Group B.

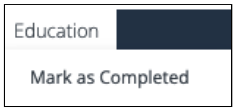
Back to Table of Contents

Assignment: Building a Balanced Data Set

Building a balanced data set is key for training a machine learning model that can make accurate predictions on future, unseen data. You have already performed a simple sampling technique in a previous exercise. In this exercise, you will work with the "adult" data set. You will learn how to analyze your data set for balance, and continue to practice sampling methods, along with other techniques to create a balanced data set.

This exercise will be graded. Graded exercises will contain the bull's eye icon in the exercise title.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

For general information on using Jupyter Notebooks, you can expand the right-hand panel adjacent to the Notebook by clicking on the icon.

Note: To execute a cell in a Jupyter Notebook, press shift-enter or select **Run** in the menu bar. Prolonged inactivity or navigating to a different page will cause the notebook session to timeout. To resume your work where you left off, be sure to re-run all of the notebook cells in order, starting again at the very beginning.

This exercise will be auto-graded.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

Read: Data Preparation Techniques

As we have learned by now, data lies at the core of developing a viable machine learning model. In the real world, however, data is not always suitable for training out-of-the-box. Missing values, bad formatting, outliers, and data redundancy are just some of the common problems a machine learning engineer has to overcome on a frequent basis. The saying "garbage in, garbage out" cannot be more true in the world of machine learning. Below, let us examine an example dataset that contains some of the issues we just discussed. What do we notice?

- The entry for John Doe is repeated
- Income format is not standardized
- Credit score contains two outliers — 0 and 35000, likely to be errors as typical range is between 350 to 850
- The entry for Melon Usk contains a missing value
- Occupation and Job Sector are somewhat redundant as they tell similar stories

☆ **Key Points**

It is essential that raw data go through pre-processing steps such as cleaning, feature engineering, and outlier handling.

Data matrix is the ideal input format for machine learning. The first N-1 columns in a data matrix are features and the last column is the label. Each row in a data matrix is a data point.

Exploratory data analysis (EDA) helps engineers and scientists understand the basic properties of a dataset including its shape, statistical properties, and whether it contains outliers or not.

Sample dataset

Name	Income	Credit Score	Occupation	Job Sector	Loan Status
John Doe	$76,000	650	Engineer	Engineering	Good
Gill Bates	$85,000	760	Nurse	Healthcare	Defaulted
Jane Doe	"95000.00"	0	Banker	Financial	Good
John Doe	$76,000	650	Engineer	Engineering	Good
Melon Usk		810	Flight Attendant	Transportation	Excellent
Barren Wuffet	5000/mo	35000	Contractor	Construction	Defaulted

While data preparation does not get as much spotlight as the actual algorithm of a machine learning model, it is absolutely essential that data preparation is executed appropriately in order

to achieve a viable machine learning model in production. Here we will go over the most common approaches in data preparation for training a machine learning model.

Data Understanding

Define the Problem

This is step zero of any machine learning project. Before we even perform any action, we need to first answer the who and (what) we are modeling. This can be answered by defining the problem statement and breaking it down into units of analysis.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the act of "poking around" the data for any obvious insights. This entails bringing in the data (or a subset thereof) into our coding environment, performing a descriptive summary using common statistical libraries, and making plots such as scatter plots and histogram for any obvious insights.

Visualization

There is a wealth of visualization techniques that are designed to help us understand our data better. These techniques help us determine the skew of data, the shape of its distribution, and if any outlier is present. Incorporating visualization in our data preparation workflow is essential so that we can have better types of cleanup and feature engineering that are needed prior to training the model.

Data Preparation

Sampling

Once we have a concrete idea of the problem we are trying to solve, we need to ensure our data is meaningful in the context of the problem. This can be thought of as having the data drawn from the same environment as production. For example, if we are trying to train a self-driving car, the image we train should be from actual roads and from the same types of hardware as the ones used in production. We wouldn't use images from video games even though they may be more easily obtained. In addition, we want to ensure our data points are independently and identically distributed (I.I.D), meaning that they should be drawn from the same distribution and be independent of each other.

Specifying Label

The training dataset for supervised learning must have some “ground truth” (label) associated with each data point. The label is what we are trying to predict, and this is the goal of our machine learning model. Depending on the use case, we may cast labels from one data type to another. For example, turning a customer review from the range of 1.0 to 5.0 stars into binary values of GOOD or BAD. This is also where we make the determination of whether the model should be a classification model or a regression model.

The Data Matrix

The underlying code for most machine learning models is optimized to process data in an N-dimension table — we call this format the data matrix, whereby each row represents an example, or data point, the last column represents the label, and each of the remaining columns represents a feature. A typical task for a machine learning engineer is to arrange data into this format prior to feeding it into a machine learning model.

Data Cleaning

Most data is inherently “dirty”. This can be caused by problems such as system error, data corruption, and poor quality control. In data cleaning, we ensure our data does not contain missing values or any unwanted outliers. When these conditions are encountered, we perform actions such as dropping the row or winsorization in order to ensure our data matrix is clean and free of data that are non-representative of the problem we are trying to solve.

Modeling

Feature Engineering

The data available to us may not always be readily suitable for training a machine learning model. Sometimes this is due to formatting incompatibility, other times the models themselves are optimized for certain types of input. Feature engineering concerns with bringing data into the right format and representations required by the intended machine learning model.

Back to Table of Contents

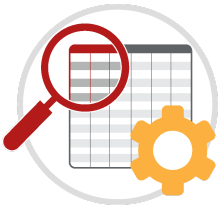
Module Wrap-up: Build Your Data Matrix

Without proper data, it would be difficult for a machine learning model to solve its intended business problem. In this module, Mr. D'Alessandro demonstrated how to identify features and labels from a business problem. You practiced handling the most common format for machine learning datasets, the data matrix, a two-dimensional table where each row is an example (data point) and each column is a distinct feature, with the exception of the last column being the label.

Remember that there are various considerations around sampling the right data for your machine learning model. One critical idea around sampling is the concept of *Independent and Identically Distributed,* which means that your examples should come from the same distribution and contain no interdependence with one another.

Back to Table of Contents

Module Introduction: Create Labels and Features



Features and labels come in different data types. When it comes to the data preparation phase, it is important to understand the difference between the various data types. Mr. D'Alessandro will introduce each type used in machine learning and you will practice obtaining them from your matrix. Mr. D'Alessandro also introduce feature engineering, the process of transforming feature values into other forms. This allows for the machine learning model to be more successful. You will practice determining the right label for your machine learning problem as well as preparing your data to practice feature engineering.

Back to Table of Contents

Watch: Different Data Types

It is important to understand the difference between each data type used in machine learning when you are in the data preparation phase. Watch as Mr. D'Alessandro continues the discussion of data preparation and the importance of columns in a data matrix.

Video Transcript

I think one of the most underrated skills in the model development process is data preparation. The model does the prediction work, and your results and insights and what your colleagues mostly see. Data preparation, though, usually takes the bulk of the time, and it is common for a reasonable percent of your data preparation work to not even make it into the final model. In general, good data prep will be rewarded through strong results. We are going to continue our discussion of data preparation here and start focusing on the columns of your data matrix. The first thing to learn for the data preparation phase are the different common data types. Also, the phrase "data type" can be interpreted in multiple ways. In programming, we might think of the data types as float, integers, and strings. The data types shown here are mapped to those programming types, but here they're translated more from a mathematical point of view. All machine learning methods ultimately operate on numbers so you can think of your data prep goal very simply as creating a matrix full of numbers. But the different data types here do have implications on how you might prepare features and how you can use them as labels for supervised learning tasks. Let's start with a numeric branch of our chart and discuss continuous data types. These are your standard floats or numbers. If a data column is this type, then it is technically ready for machine learning. One thing to be aware of here, though, is the distribution of continuous data types can produce outliers. These outliers can cause unwanted behaviors in certain learning algorithms, so they need to be dealt with in pre-processing. This type can also be used as a label for regression problems. Again, you'll want to be aware of any skews in the distribution and outliers before proceeding to the modeling phase. Integer data types have similar properties as continuous. Sometimes you could treat them the exact same way. For example, income is usually expressed as an integer. We can use it as is as a feature but might need to pay special consideration to skews in the distribution. Similarly, as a label, we can often use integers as is in a standard regression. Just remember that the regression will output a continuous float. If your integer label represents a count of some phenomenon, it may not make sense to have your predictions be a float with decimal points. There are specialized statistical methods for this type of situation, but for most machine learning applications, we can treat them the same. Now moving on to categorical types, the ordinal data type is probably the most unique. An ordinal type is a mix of categorical and numeric. It is usually an integer-based number, but the range is usually so small that we can treat them as discrete categories. One very common

example of this type are the ratings you see in a lot of websites or surveys where people use a 5- or 10-point scale to express an opinion about something. What makes this type special is the absolute ordering matters in terms of being able to compare different values, but the relative difference doesn't matter. For instance, if you rate two films as a 2 and a 4, we can say that you like the 4 better, but we can't say you like it twice as much. This property is why we often preferred to describe this data type as categorical as opposed to a straight number. Because the ordinal types are numbers, they can be used directly as features, but the caveat is your machine learning method is going to treat them as numbers. With standard numeric types, we can do things like extrapolate to values that haven't been observed. We wouldn't want to do this with an ordinal number, though, so it is often best to convert them to binary indicator features, which is a technique we'll introduce in another video. If you're predicting an ordinal value, you can use both regression and classification. However, if using regression, you may get values that don't make sense, like predicting a minus 3 or a 10 for a label that takes on only values 1 through 5. Using classification instead of regression will help you avoid this problem. The nominative type is what we likely think of when we imagine a categorical variable. These are often represented as strings, but that's not often the case. Database systems often encode discrete categories as integers. If you encounter this, it is very important to not treat them as pure numbers. Numbers have orderings and categorical data types do not. Another important thing to note is that most machine learning algorithms cannot directly operate on strings so they always need to be converted to a numeric form. We'll introduce later a method called one-hot encoding that solves this exact problem. Last, nominative data types are the typical basis for classification. When we use them as labels, we call the individual values classes, and our model either directly predicts the class or produces a score that represents the likelihood of belonging to a specific class.

Back to Table of Contents

Code: Inspecting Data Types in a Pandas DataFrame

In this activity, you will learn about the different data types available in Pandas and how to inspect the data types of values in your DataFrame columns.

This activity will not be graded

This content cannot be rendered properly in the Course Transcript, please log in to the course to view.

Back to Table of Contents

Watch: Specifying a Label

When working on a supervised learning task, you will first need to define the label. Sometimes defining a label can be easy. Other times, however, defining the label can be more difficult. Mr. D'Alessandro discusses different instances of specifying a label and walks through some examples.

Video Transcript

When we build our data matrix for machine learning applications, we typically invest a lot of time creating the right columns. Most of the columns will represent features, and we call this process feature engineering. If we are doing a supervised learning task, we need to first define the label. Nearly every aspect of the development process is centered around the label, so defining it is a natural place to start. Sometimes defining the label is pretty trivial. Here are some examples where this is true: if there's a clear set of discrete choices to model; each example maps cleanly to a single class; and the label is directly observable. This entire process starts from the problem statement. Oftentimes, the problem statement represents simple yes-or-no questions on well-defined and observable outcomes. Even if it's not binary, the same may hold true on a bounded set of discrete outcomes. Take advertising, for instance. We usually have easy-to-measure-and-define events, such as viewing and clicking on the ad. With digit recognition, which is a task needed when identifying text and images, we know pretty well in advance that there's a limited set of elements that each example maps cleanly to, such as numeric digits or alphanumeric letters. We can learn to appreciate the easier conditions by considering when these conditions are not true. This is when maybe the problem goal is very subjective; there are multiple relevant labels to consider; the ideal label is harder to work with; or we don't directly observe the label we want to predict. It's important to repeat that defining the label is tightly coupled with the problem statement. In a lot of real-world applications, we want to optimize things we cannot observe. For example, a lot of companies that use recommendation systems aim to maximize a more abstract outcome like user satisfaction. This is intuitively correct but is often challenging to model because it meets all of the harder conditions. When the problem goal itself is a subjective concept, it can lead to multiple different interpretations and possible labels to represent those interpretations. Even if there is consistency in the interpretations, we may not be able to observe the concept directly, so we need to create some approximation of it. This is probably a bit abstract, so let me walk through an example. I'll use an example common in social networking. The context here is feed recommendations, which are pretty universal in most social networks. There is a lot more complexity to this than I think people usually realize. The real problem goal is to maximize long-term user satisfaction. This is what keeps the social network relevant and entertaining, and it's how you retain your customers. But this problem goal meets all of our

difficult conditions — it is subjective and hard to measure at scale — so we often rely on what we call proxies. In this case, the primary proxy is user retention after 90 days from now. But even the proxy here presents challenges. This one is too hard to work with usually because it takes a long time to measure — specifically 90 days — and this isn't helpful for making short-term decisions. So instead, we might use a proxy of the proxy, like various types of short-term engagement; maybe clicks or commenting. All told, there is no single right answer to how to label this problem. Like this, some of the most important decisions we make in model development are subjective and thus can't be directly optimized or measured. One of the harder conditions I mentioned was when the ideal label is harder to work with. Here are a few examples that illustrate this condition and how we can deal with it. And the general idea is to take a harder problem and simplify it so it becomes easier to work with. For instance, predicting an ordinal outcome. like an online review from maybe 1 to 5, can be challenging. Oftentimes, we only care about high versus low ratings. In this case, we can map the ratings to a high-versus-low scale and use binary classification; as in if the rating is 4 or 5, we label it as a 1 and otherwise a 0. There's, of course, some subjectivity in how we might define high and low, but the core idea is to simplify the problem. Another example is when we want to predict the time that something might happen. Every company cares about how long a customer will stay as a customer. Machine learning can be a good tool to predict customer tenure, but working with time-to-event data is often pretty challenging. One way to approach the problem is to convert it to binary classification, where we are predicting whether or not the customer will still be a customer after some pre-specified number of days. We would lose a bit of granularity in our predictions, but this also makes the problem much easier to solve. There may be other situations where this trick applies. The general thing to remember is that, as model developers, we have these tricks to rely on, and we choose them based on our subjective judgment. Of course, remember that anytime you change the label, you are effectively changing the problem statement. In model development, we make a lot of decisions by just setting up experiments and finding the optimal choice. But with alternatives for labels, we are making apples-to-oranges comparisons because each different label is technically a different problem. When subjectivity is an inevitable part of your design decisions, always solicit peer feedback to make sure that you're still aligned with the spirit of the original problem.

Back to Table of Contents

Assignment: Creating Binary Variables

One of the typical tasks in data preparation is to convert a feature or a label that has multiple categorical values into one that has a binary value. This means that instead of having multiple potential values, a feature or a label will have just two potential values. In this exercise, you will practice converting a feature that has multiple values into one that has a binary value.

This exercise will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

Read: Introduction to Feature Engineering

The goal of feature engineering is to help prepare our data for our machine learning model to train on. In feature engineering, we are not so much concerned about the cleanliness of the data such as missing data or outliers, as those will be addressed in other processes. We are focused on mapping the appropriate predictive or causal concepts into a data representation, and then transforming it into a format that can be easily consumed by an intended machine learning model. We are essentially answering, *how do we get from some data source to a column in a data matrix?*

To achieve this, there are two strategies that we take.

1. Select the right data to be used as our features
2. Transform the data into a format that is suitable for the intended machine learning model

☆ Key Points

Feature engineering contains two parts, and they are both integral to the success of a machine learning project.

The first part of feature engineering involves gathering the right data for the job.

The second part of feature engineering involves selecting and transforming features so that they are well-received by machine learning models.

The first step involves the selection, filtering, and fetching of the desired data from some data source into our machine learning environment. Assume you work as a data scientist at a manufacturing plant for a moment. We believe there is some predictive and causal relationship between a machine's condition and its operational status. With that in mind, we decided to have the machine's condition be our **feature** and the operational status be our **label**. The next step is to bring data into our machine learning environment. As shown in Figure 1, there are several data sources involved in order to achieve this. We have the following data sources:

1. A running log of the conditions reported from the machines in real-time
2. A database that contains the machine's model by machine ID
3. A file store that houses all the files that maps a condition code to a human-readable description
4. The API of a machine monitoring service that allows us to query the status of a machine at any given time in the past

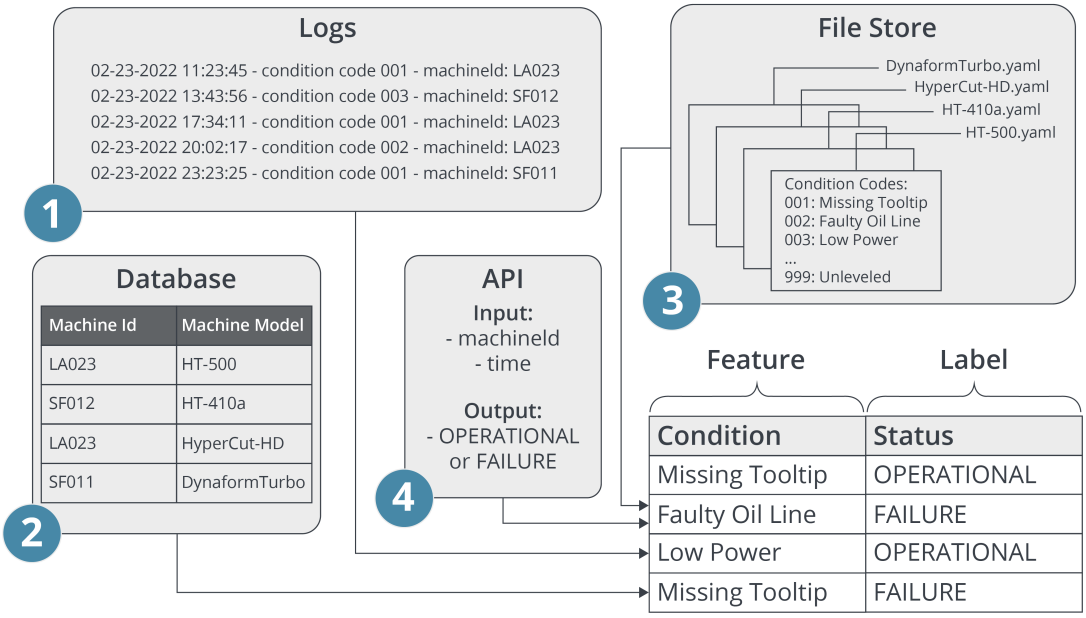


Figure 1. Feature Engineering Strategy

By merging all these data sources together, we arrive at a feature which we believe to be predictive of our label. We can then repeat this step in a similar manner to bring in other features for constructing our data matrix. Once we have all the features in place, we will be ready to move on to strategy two, which is feature transformation. In feature transformation, we are essentially converting our features into a format that is understandable and well-received by our machine learning model. What types of feature transformation to apply depends on the model. Some require the data to be in numeric form and scaled, while others can accept 2-dimensional pixel data in integer type. Common feature transformation techniques include **binary indicator**, **one hot encoding**, and **functional transformation**.

Back to Table of Contents

Watch: The Process of Feature Engineering

After defining your modeling population and label, you can start the process of feature engineering. There are two main strategies for feature engineering: mapping predictive or causal concepts to data representation and manipulating data so that it is appropriate for common machine learning APIs. Watch as Mr. D'Alessandro further explains feature engineering.

Video Transcript

After we define our modeling population and label, we next want to start the process of feature engineering, we could think of feature engineering as consisting of two broad strategies. The first strategy is mapping predictive or causal concepts to data representation. The second strategy is manipulating data so that it is appropriate for common machine learning APIs. This isn't an either/or situation too. Usually, we are going to use both strategies. We will dedicate a separate video to cover the first strategy I just mentioned, which was mapping predictive concepts to a data representation. The second strategy has four main techniques which we have summarized in this table. Each of these represents a strategy we may consider when preparing our data for modeling. We'll assume too that the inputs to any of the four are the outputs of the first strategy. In other words, you likely would have done some work converting log or database information into concepts you think would make for good features. The transformations in this table might represent the last stages of data preparation. These are all very important, but probably the most important one to consider is what we call one-hot encoding. I call this out specifically because categorical data absolutely needs to be converted to numeric forms before it can be used as a feature. The other transformations are optional and are used to optimize the model, whereas one-hot encoding is not optional when you have categorical or text or string data. I want to next elaborate more on how these two strategies fit together. The first strategy is how you usually begin as a model developer. It is usually your job to aggregate your company's various data assets into a single modeling matrix. You will be converting data with different formats and locations into a common structure. This process can take a long time too. The data transformations are easy to automate. Once your code is written, you can reuse it. The first stage is usually helped by creating a conceptual model of your problem and using intuition to guide you. We'll cover more of this in another video. From a performance perspective, both strategies can be useful, but most of your performance usually comes from the first strategy. If you aren't capturing the right concepts in your data, no algorithm is going to make a good prediction.

Back to Table of Contents

Watch: Domain-Driven Feature Engineering

In this video, Mr. D'Alessandro explains the first strategy mentioned in the previous video: mapping predictive concepts to data representation. This will output what you would input into methods such as one-hot encoding (strategy two). Take note of the principles Mr. D'Alessandro lists to guide you in the future.

Video Transcript

There are certain specialized use cases where feature engineering isn't all that important anymore. Two examples are image recognition and language translations. These are both applications where deep neural networks are the state of the art, and deep neural networks have a special property where they can automatically learn feature concepts from raw data. But prior to the common use of deep neural networks, concept based feature engineering was the norm. Outside of these particular applications, content-based feature engineering is still critically important. In all of these cases, you'll need to rely on your domain knowledge and intuition about the problem. The stages of feature engineering where we map concepts to data representations is very much an art. There isn't a robust theory or well-defined methodology to share, so instead, I'm going to cover this with a few principles. The first principle is to think like a scientist. Scientists study how the world works, usually with the goal of understanding cause and effect. In other words, based on your knowledge of the event you're trying to model, identify likely causal factors and test them empirically. The best features in the model are the ones believed to actually cause the outcome. You don't have to limit yourself to proven causes, though. Even factors that are merely correlated with the outcome are appropriate as features for prediction tasks. The next principle is an extension of the first. You are likely not the only expert on the problem at hand. Make sure to seek the input of others who may have many years of experience and know the problem space very well. These don't have to be technical experts, either. Oftentimes, people who interact with customers have the most knowledge and can provide a lot of information about how to approach a problem. Another reason to do this is to understand what features not to build. There are often both legal and system constraints that make a feature ineligible for use in your model. Knowing this in advance can save you a lot of time down the road. The last principle is meant to be super practical. Your and others' imaginations dictate the limit on how many features could be built. In reality, each feature may take the same amount of time to code up but there will be massive differences in their predictiveness. Unfortunately, you often won't know in advance which will actually become the most predictive. Use your collective intuition and knowledge of the domain and plan ahead. You can create a list of all the features you plan to build and seek input on their expected value. Then build these features and evaluate them on your data as you go along. You will likely find you're hitting a point of diminishing returns, and

this may signal that it's time to stop and move on. Let's try to illustrate these points through an example. The example I'll use is Twitter's recommendations on who to follow. Each of these examples is someone that Twitter's algorithms thought would be a good match. We can assume that the label is whether or not I follow the person after being recommended. Let's now try to think like a scientist for this case, and remember, we aren't trying to come up with some new theory about human behavior. Instead, we are creating concepts that either cause or are highly correlated with the outcome of interest, which again is following someone. Here are a few example concepts for the feature engineering process. I pose it as simple questions to explore. Why do I follow someone? As a Twitter user myself, I can think of, what thought process I have when I decide to follow someone. Sometimes such processes are abstract, such as, do I find the person interesting? We usually want to avoid such abstract and subjective concepts. Instead, we want to define factors that are specific enough that they can be coded up in an unambiguous way. Some example answers of why I might follow someone. The person posts about topics that I am interested [in]; we could represent that as a match between topics that they post and topics that I have explicitly followed. Or the person follows me. Maybe we could assume that I would reciprocate. Also, people I follow also follow this person. If my friends or people that I consider interesting also consider someone else interesting, maybe I would as well. Or very simply, the person is very popular. If a lot of people consider this person interesting, maybe I would as well. These are all ambiguous concepts that we can encode as specific features for our problem. All of these examples meet our condition, and you can see how we might translate those features that make it into our data matrix. Let's assume we did this exercise long enough to get a sizable list of potential features. In a frictionless world, we would build all of the features and test them out to choose the best set. Let's also assume that each feature has a roughly fixed engineering cost. At some point they won't be adding much value, either because they're not predictive or they're redundant with other features. If we were to plot feature development cost with model performance, we would likely get a chart like this. Notice too how I keep revisiting this same chart, but I only change the axis descriptions. This idea of diminishing returns is pretty common, so I like to reinforce it when I can. We can think of the lines representing efforts one and two take the same time to build about 50 candidate features. If our intuition is really good, we can get a lot more performance per unit time in the first effort than the second, meaning the first 50 features are more predictive than the second 50 features. The lesson here is we want to use our collective subjective judgment to prioritize what to build, and then empirically test what we build as we go along. Once we start to hit those diminishing returns, we can then evaluate how much additional effort do we really want to put into the feature engineering.

Back to Table of Contents

Read: Feature Transformations

☆ **Key Points**

Sometimes features need to be transformed in order to be understood by its intended machine learning models.

Some feature transformation techniques are optional but are ideal as they lead to better results in models' performance.

One-hot encoding is the most important feature transformation technique that turns categorical values into binary vectors.

Different machine learning models are intended and optimized for different input. This means machine learning engineers need to first transform the features into the appropriate format or values in order to train the models properly. For example, some algorithms may require that all the features be normalized to the same scale prior to training, whereas some do not. On the other hand, all of the algorithms essentially require any categorical values to be converted to numerical format. Below, we explore some of the most widely used transformation techniques in machine learning.

▼ Binary Indicator

In some cases, it makes sense to represent numeric or categorical data as binary values. The benefit of doing this is to make our data smaller, which means lower computing cost. It also makes our final model simpler, which can be desirable if we want our model to ultimately be more general rather than specific. Some of the examples of binary indicators are as follows:

- Cast Movie rating greater than or equals to 3 to *Good*, and movie rating less than 3 to *Bad*
- Cast Glucose level of 100 or higher as *Abnormal* and below 100 as *Normal*.
- Group countries into *English-Speaking* or *Non-English Speaking*

▼ One-Hot Encoding

One-hot encoding is one of the most important feature transformation methods for categorical data. Machine learning algorithms operate on numerical inputs, therefore we have to transform categorical data into some form of numerical representation. One-hot

encoding is an excellent candidate for this task because it is easy to understand and straightforward to implement. The idea is to convert K categories into an array of K binary values prior to being consumed by a machine learning model.

To illustrate this, let's say we have a feature called *weather*, which has five distinct values: *Sunny, Overcast, Rain, Snow*. We also have two other features called *elevation* and *zip code*. Both of these have numerical types and therefore do not need to be one-hot encoded. In order to one-hot encode our *weather* feature, we would replace our feature (column) *weather* with four new features (columns): one new feature (column) for every distinct value in the *weather* column. As such, we would have four new columns: *Sunny*, *Overcast*, *Rain*, and *Snow*. Each new feature would have a binary value: either 1 or 0. For every row in which it appeared in the original *weather* column, it would have a value of 1. Otherwise it would have a value of 0. For example, in Figure 1 the second training example (second row) has a *weather* value of *Overcast*. Therefore, the new *Overcast* column has a 1 in its second row, and a 0 in all other rows.

Data before One-Hot Encoding

Elevation	Zip code	Weather
1000	92651	Sunny
40	12834	Overcast
500	85532	Rain
232	43645	Snow
157	23426	Rain

One-Hot Encoding

Data after One-Hot Encoding

Elevation	Zip code	Sunny	Overcast	Rain	Snow
1000	92651	1	0	0	0
40	12834	0	1	0	0
500	85532	0	0	1	0
232	43645	0	0	0	1
157	23426	0	0	1	0

Figure 1. One-Hot Encoding

▼ Functional Transformations

In functional transformation, we apply a function to convert one numeric value to another based on some function. Some classic examples of this include converting a feature to log scale or to the square.

As illustrated in Figure 2 below, if we were to train a model using the feature values as is, our model has to be more complex in order to capture the relationship among features

and labels. Not only we would need more data to train our model, but the model will be slower in making predictions. If we were to transform the feature values so that the features and labels have a linear relationship, we would simplify the model and improve the speed of predicting.

Feature	Label
-3	8.99
-1.98	4
-1	0.98
0.1	0
1.1	1
2	4
3	0

Square()

Feature	Label
9	8.99
3.9204	4
1	0.98
0.01	0
1.21	1
4	4
9	0

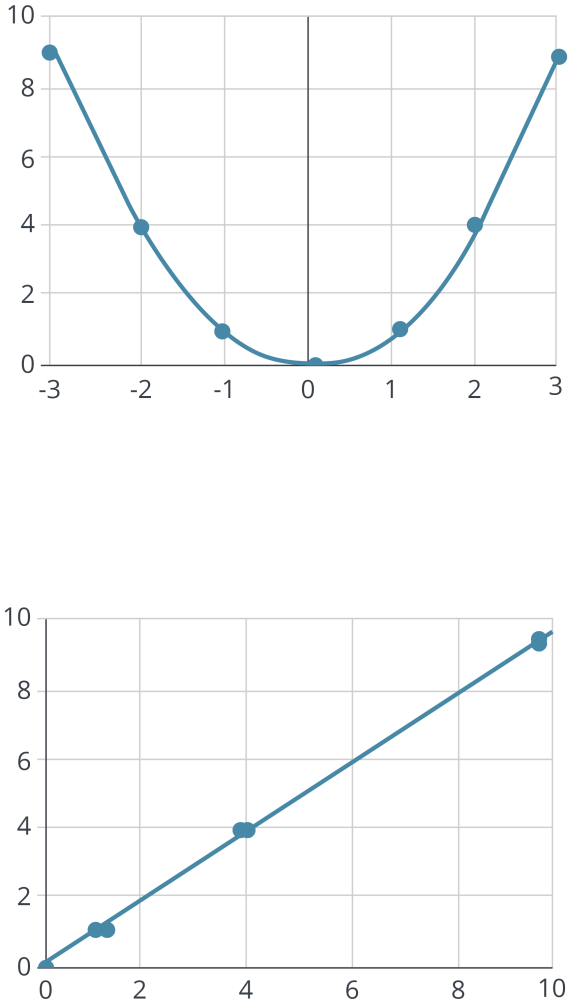


Figure 2. Feature Transformation Using Square

While this is a rather simplified and contrived example, you can imagine when we are dealing with millions of examples, having tens and hundreds of features, and making thousands of predictions per second - it really makes a difference to first transform our features.

▼ **Interaction Terms**

In some cases, the combined effect of one or more features leads to a stronger predictor than each feature alone. In such cases, we can form an additional feature by multiplying the individual features (or by performing another mathematical operation) to arrive at a third feature.

X1	X2
1	2
4	6
2	3

X1	X2	X1X2
1	2	2
4	6	24
2	3	6

Figure 3. New interaction terms created by multiplying X1 and X2

▼ Binning

Sometimes we want to convert numerical values into discrete “bins” when the ordinal nature of the numerical values themselves aren’t necessarily indicative of the label. For example, binning the weights of individuals into groups in an attempt to predict their risk of heart diseases. This can be useful if we want to reduce the complexity of the model in order to achieve better generalization, a concept we will go over in more details. Binning is typically visualized as a histogram as shown in figure 3 below, where weights are being grouped into seven distinct categories.

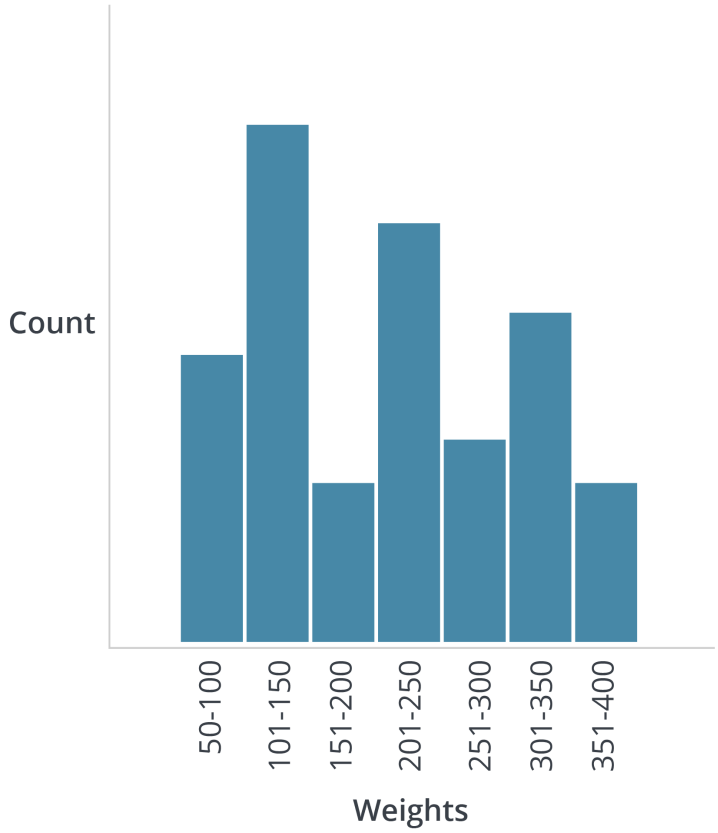


Figure 4. Histogram For Binning Weights

▼ Scaling

Some models perform better when features are scaled to some standard range, while others have no problem accepting an input of a wide range. The two most common types of scaling approaches are the standard scaler (also known as standardization) and the min-max scaler (also known as min-max normalization).

In the standard scaler approach, we transform the values of a feature to have a mean of 0 and a standard deviation of 1.

In the min-max scaler approach, we transform the values of a feature to a range between some min and some max value (often times 0 and 1), while keeping their relative distance the same.

Summary

This table summarizes the expected input and output data format for various feature transformation techniques. Using the right feature transformation technique ensures our model will behave as expected and also have the best chance of being accurate and generalizing well to new, previously unseen data.

Expected input and output data format for various feature transformation techniques.

Technique	Input	Output
Binary Indicator	Categorical or Numeric	Binary (0/1)
One-Hot Encoding	Categorical	Binary (0/1)
Functional Transformation	Numeric	Numeric
Interaction Terms	Numeric	Numeric
Binning	Numeric	Categorical
Scaling	Numeric	Numeric

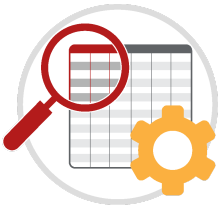
Back to Table of Contents

Module Wrap-up: Create Labels and Features

Having the right features and label is the basis for training a successful machine learning model. In this module, Mr. D'Alessandro introduced various feature and label data types intended for different situations. He also introduced feature engineering, which is the technique for transforming feature values into other forms that are more conducive to a successful machine learning outcome. Some of these feature engineering techniques include binary indicators, one hot encoding, functional transformations, and interaction terms.

Back to Table of Contents

Module Introduction: Explore Your Data



Another important aspect of data management in machine learning is understanding the data through exploratory data analysis. In this module, Mr. D'Alessandro introduces different approaches to understanding your data through descriptive statistics, multivariate analysis, and plotting. You will practice gathering statistics on your data and generating plots to better visualize what it is describing. The outcome of these techniques is to identify any trends, patterns, and interdependence among features and labels.

Back to Table of Contents

Watch: Introduction to Exploratory Data Analysis

Let's continue working through the data preparation stage of the model development workflow. So far, there's a representative sample of users and a structured data set that includes features and, if appropriate, a label. This stage is also commonly called exploratory data analysis. There are two main goals. First, you want to ensure to have high-quality data. The second goal is aimed at delivering insight so that you can become really knowledgeable about the problem.

Video Transcript

We are going to continue working through the data preparation stages of the model development workflow. Let's assume we've gotten to the point where we've pulled together a representative sample of users and have a structured data set that includes features and, if appropriate, a label. We can think of this stage as us becoming data investigators or explorers. In fact, the stage is most commonly referred to as exploratory data analysis. We usually approach our data investigation phase with two main goals. These are to first ensure we have high-quality data, and bad-quality data might be data with a lot of missing values or with outliers that we shouldn't ignore. Another way to interpret good versus bad quality is in terms of the data's predictive utility. That is a valid interpretation, but in this module, we'll reserve the word "quality" for missing-ness in outliers. The second goal is aimed at delivering insight so we can become really knowledgeable about the problem. There is no limit to what insights we want to get out of our data during this phase. We call it exploratory data analysis, or EDA for short, and the word "exploratory" may assume that we're just randomly poking around the data, hoping to find something interesting. There's no rule that says you can't do this, but usually we have a time constraint. There are several key questions worth asking of your data, and these are oriented around supervised learning. These questions are, first, how is the data distributed, which informs us about outliers or skews we should address. Second, which features are redundant? This informs us about which features we may or may not want to cut. And then finally, how do different features correlate with our label? This informs us about features to keep and offers high-level explanations on how our model might be working. These three questions translate to specific methods we can apply, and they're made with direct utility to the model-vetting process. Ultimately, we want to end up with well-distributed independent features that are predictive of the label, and this process will help us get there. The first step you should always take when working with data is to literally look at it. You might do this in several ways. The first is when you're given access to some flat text file. Before you load it into Python, you might want to check for a few things. Here are some basic must-learn operations to do with Linux commands. The first command, called "head," prints out the first few lines. This immediately provides a lot of value. We can see that the file has a header along with the names of each of the columns. We can see the

delimiter, which in this case looks like a tab. We can also see that one of the columns has only missing values. Two other ways we may want to immediately inspect our data are to look at its size, both in terms of number of columns and rows. Here again, we can use Linux commands. The top command here gives us the line count. the command "cat" basically prints the contents of the file. The vertical line after the file name is what we call the pipe command and lets you chain two commands such that the output of the first is the input to the second. The next command that you see is a counter, and with the option -l, we're asking it to count the number of lines that are being passed to it. The second line is how we can get the number of columns. We've already discussed head, the pipe command, and WC. The command in the middle translates the first option to the second. So we're replacing the delimiter with a new line. We're effectively transposing the column head, some lines, and counting the lines here. We can do these exact operations if the data has already been loaded in Python. Here's an example on the same data using Pandas in Jupyter Notebook. Now, we might use Linux commands when we first download a data file to check it out where it is stored. When working in Python, I recommend taking previews of your data like this at each stage of processing. We can think of this as a visual unit test. Inspecting data visually at each processing stage can be a great way to quickly identify errors in our processing logic, particularly errors that lead to missing values or records.

Back to Table of Contents

Code: Using Command Line Tools To View Data

There are different ways to explore your raw data. One way is to explore data in your Jupyter Notebook using the Pandas package. You have already been doing this. Another way is to explore raw data stored on a server using command line tools. This activity will focus on some common Linux commands that can be used to explore data stored on a Linux server.

This activity will not be graded.

This content cannot be rendered properly in the Course Transcript, please log in to the course to view.

Back to Table of Contents

Watch: Using Pandas to Inspect Your Data

Remember that one of the main goals is to check for data quality issues. In Pandas, you can use an easy method to get a lot of information at once, which will tell you a lot about the data's quality. This method is the Pandas Dataframe describe() method. Mr. D'Alessandro walks through an example.

Video Transcript

Let's recap that when performing exploratory data analysis, one of our main goals is to check for data quality issues, where data quality, in this context, refers to the presence of missing values, or outliers. There is an easy method to get a lot of information at once in Pandas, and this information will tell us a lot about our data quality. This method we'll use is the Pandas data frame describe method. There isn't much introduction needed for this particular method. It is easy to use; I'll get right into the example. Here in this example is a sample data set. We can ignore the data context for now. The output format is the same for any data, so we'll cover the output as well as show how to pull useful insights from it. I have highlighted how to use this method, which is just an excerpt from Jupyter Notebook. It is very easy, and in this case I filtered it to a limited set of columns. You don't need this filter, but it does make the output easier to read. The second highlight shows the output. These are standard distribution statistics, showing count, mean, standard deviations, three key quantiles, and then min and max. We can learn a lot with these basic stats. For instance, the count is supposed to tell us the count of records that have non-null values. This immediately lets us know that one of the columns here contains missing values. In this case, over 95% of the column's values are missing. This is something we would definitely want to look into. Averages are probably the most common way to understand data. When I look at this, two things jump out to me. First, let's look at the column y_buy. I didn't say this earlier, but this is actually a binary label. The average is incredibly low. Generally, if your binary label is 1 less than 1% of the time, we call this a class imbalance problem. This can cause modeling problems. We won't discuss this now, but just know that this is something to be aware of when you're doing your data exploration. The second insight is in the num_checkins column. When the mean is much greater than the median, we know that this is a highly skewed column. High skew is related to having outliers. Basically, features with extreme values can sometimes negatively impact your model performance. Next, looking at the min here, we see that two of the columns have negative values. Negative values aren't a problem necessarily, so it depends on the definition of the feature. The uniq_urls column is meant to be a count of something. A negative value should raise a red flag here. This can be an encoding error, or at some point someone could have converted missing values to negative ones. In either case, we probably would need to do some pre-processing to address this particular concern. Now, there is more we can learn from

this one example. Of course, every sample data set will provide its own unique insights. The point here wasn't to enumerate all of the specific insights; instead, I wanted to highlight how we can get a lot of insight from looking at basic summary statistics using a simple one-line command from Pandas. I highly recommend doing this with every new data set. Additionally, the output that the describe method displays is stored as a data object. You can definitely create automated unit tests to check for the type of issues we found visually here.

Back to Table of Contents

Assignment: Get Summary Statistics Using Pandas describe() Method

In this exercise, you will continue working with the `describe()` method to explore and analyze your data. In particular, you will focus on obtaining statistical information about your data.

This exercise will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

Watch: Visualize Your Data: Univariate Plotting

Watch as Mr. D'Alessandro describes some basic plotting techniques. The use of plots is to understand two basic things: how a given feature is distributed and how two or more features are distributed together. In this video, we'll examine histograms.

Video Transcript

In this video, we'll cover some basic plotting techniques. Plotting your data gives a lot of information that summary statistics just don't provide. We usually use plots to understand two basic things. The first is how is a given feature distributed? This is what we would call a univariate insight. As discussed in other videos, the distribution plots will tell us the range of the data, its central tendency, skew, and variance. Knowing this information will inform us on what kind of preparation techniques we might need to use. The other goal is to know how two or more features are distributed together; a plot that tells us this can help us know if two features are really correlated or if they're redundant. If one of the features we're plotting is the label, then these plots give us powerful insights into how the models work. These plots will help us with feature selection, where we want to keep features that are correlated with the label and remove features that are redundant. This video will introduce two common plots for understanding univariate distributions. The main plot and concept I want to illustrate is the histogram. A histogram is a type of bar chart where the x-axis represents the range of the data, specific feature usually, and the y-axis tells us how much of the data is present in that range. We can start with this simple example here. The x-axis here is a feature called uniq_urls that has values from 0 to 200. The y-axis in this case is "Count," so the sum of the bar should add up to the total number of records. The count can also be normalized to represent a percent instead of a frequency. This plot reveals a unique insight that the Pandas describe function didn't show; specifically, we have a multi-modal distribution where there's a peak near zero and another peak around 180. The importance of that particular insight would depend on the problem goal. If this were our label for a regression task, for instance, we might want to investigate whether there are different segments of examples that have very different behaviors; and if true, we might want to build separate models for each segment. The histogram is a great way to identify features with a heavy skew. The left histogram here is the num_checkins feature in our data set. This has a very characteristic, long tail shape. The range is pretty wide, but most of the data is in the first 5 or 10%. The right-hand chart shows the distribution when we take the log of the feature. This looks like a typical bell curve for normal distribution, which is generally a well-behaved distribution to work with. This is a good example of a feature with which we may want to form a functional transformation, with the function here being the natural logarithm. When a feature is skewed like this, some modeling algorithms will make wild predictions on higher values of the feature. In this

case, using the log of the original feature might yield better model performance. We can use empirical tests to verify which option is better. Another nice feature of the histogram is that it works both for numeric and categorical data. Here's an example on a categorical feature with three levels. We can see most of the examples are in the left-hand column. I'm using Seaborn and Matplotlib to produce these samples, and the method to produce the plot is agnostic of whether the x-axis is numeric or represented by a categorical string. At this stage, we've covered two different ways to understand the shape of a feature. These were the Pandas describe method, to get high-level summary statistics, and the histogram, to visualize the full distribution of the data. Neither of these are perfect substitutes for each other, so I advise to use them regularly — or use them both.

Back to Table of Contents

Tool: Plotting Tip Sheet

Download the Tool

Use this **Plotting Tip Sheet** to help you with plotting in Python.

The Python ecosystem for data science provides an astonishing set of capabilities for visually representing different aspects of complex data sets. Once you get more familiar with using these tools, you can often figure out how to do something you want by analogy, using something else you have done previously, without having to dig through the documentation. In the meantime, the attached tip sheet can be a useful resource for your data visualization needs.

Back to Table of Contents

Code: Using Seaborn and Matplotlib To Visualize Data

An important aspect of the data understanding and preparation phase of the ML process is data visualization. To accomplish this, you will be using two common visualization packages: Matplotlib and Seaborn. In this activity, you will practice working with both packages to plot data contained in your data set.

This activity will not be graded.

This content cannot be rendered properly in the Course Transcript, please log in to the course to view.

Back to Table of Contents

Watch: Visualize Your Data: Bivariate Plotting

In this video, Mr. D'Alessandro explains how to visualize the relationship between two data columns. This is called bivariate plotting. The main goal here is to understand if two columns are correlated with each other.

Video Transcript

Here we'll discuss plotting methods that help us visualize the relationship between two data columns. These techniques technically extend to more than two dimensions, but four-dimensional plots aren't much of a thing, so we'll limit the introduction to bivariate distributions. Our main goal here is to understand if two columns are correlated with each other. When these columns are model features, we can use the insight to justify dropping one of the columns. As a general rule, we don't want to use highly correlated features in a model. Most of our plotting effort, though, will center on the bivariate relationship between a single feature and the label of your problem. These insights directly contribute to our model-building effort and they help us explain how our model is working to our colleagues. For numeric data, the most common way to visualize the relationship between two numeric columns is the scatter plot. This example shows four different pairs. The number shown in the title of each subplot is the correlation between x- and y-axes. We can see here how the shape of the scatter plot is related with the correlation. In general, the more dispersion, the less correlated. When building and selecting features, we usually want to see close to zero correlation between individual features and correlation close to 1 or minus 1 between features and labels. We can't always rely on correlation statistics alone, though. Take this example, where we plot a variable against its square. It is clear from the scatter plot that Y has a deterministic relationship with X. The correlation statistic is close to 0, though, which means there's no correlation. How do we explain this? The problem is that the standard correlation statistic, also called Pearson's correlation, assumes a linear relationship between the two variables. When that linear assumption isn't true — and it often isn't in real data — correlation is an inaccurate measure. We'll introduce other measurement techniques to get around this problem. But that's not the focus for now. Let's now move on to bivariate plots where one of the variables is the label. In this case, the feature is categorical and the label is a binary outcome. The plotting method we use here is a bar plot, and this example uses Seaborn's bar plot method. The y-axis shows us the label average for each value of the categorical feature. This is an example of a highly predictive feature, and we can see clear differences across the values. Seaborn lets you automatically include error bars that you can show that these differences are statistically significant. The process that gets us to this plot isn't machine learning, but it certainly is insightful. It is pretty common as a machine learning engineer to have to communicate your work to non-technical colleagues. While your overall goal may be to build a model to make

accurate predictions on your data, your colleagues will likely appreciate a good story. Plots like these help build insight around a particular problem, and the insights can help to build confidence around your model. There are certain applications where model predictions take time to evaluate, such as credit lending or insurance underwriting. In those cases, the underlying insights like this are how you might prove to your colleagues and regulators that your modeling is indeed correct. So we built this last plot of our categorical x-axis. Of course, a lot of our data won't be categorical in practice. We can apply a useful hack to get such a plot on numeric data. In this example, we take a numeric feature that has values between 0 and 180. I mapped the numbers into equal-size bins and then ran the bar plot. By binning the data and plotting, we can compute label rates, get the error bars, and also visualize the underlying trend. In this case, I used a simple floor function to bend the x-axis into bins of width 20. There are plenty of other plot types we can learn and use in the course of our work. Data visualization is broad enough that one can take an entire course devoted just to it. My main goal here was to teach some of the minimum techniques to cover our three basic EDA goals, which, again, are understanding how the data is distributed, understanding which features are redundant, and understand how different features correlate with our label.

Back to Table of Contents

Code: Bivariate Plotting With Seaborn

You have already learned how to use Seaborn to plot information about one variable (feature). In this activity you will practice how to use Seaborn to plot information about many features containing different data types.

This activity will not be graded.

This content cannot be rendered properly in the Course Transcript, please log in to the course to view.

Back to Table of Contents

Read: Correlation, Covariance, and Mutual Information

☆ Key Points

Understanding the dependencies between the variables within our data is a key step in formulating our approach to an ML problem.

Both covariance and correlation find the linear dependencies between variables, but correlation is generally easier to work with because it's standardized.

Mutual information tells us how knowing about one variable improves our understanding of another variable.

As part of our data exploratory process, we are interested in the dependencies between various random variables (e.g., features-to-features and features-to-labels). This is commonly achieved mathematically by calculating the correlation and mutual information between these variables.

Intuitively speaking, correlation finds the strength and direction of *linear dependency* between two random variables whereas mutual information finds the amount of reduction in uncertainty one variable provides for another. Let's go over each concept in greater detail and with examples.

▼ Correlation and Covariance

There are several correlation formulas that try to achieve the goal of identifying linear dependencies among variables. In order to understand correlation, we need to first learn about covariance. For two randomly distributed variables, the covariance formula is given by the following equation:

$$Cov(x, y) = \frac{\sum_{i=0}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

Intuitively speaking, a large magnitude of covariance means the variables are highly linearly dependent on each other. A covariance closer to zero means the variables are less linearly dependent on each other. The sign of the covariance value tells us whether two features are directly dependent on each other or inversely dependent on each other.

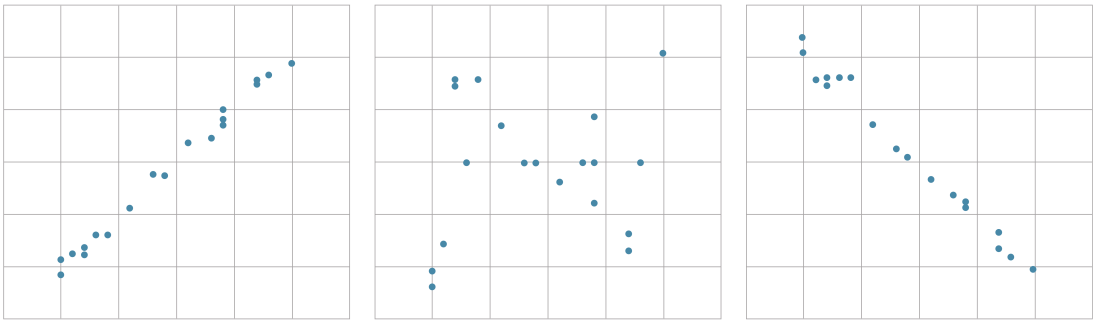


Figure 1.

Left: high positive covariance - high linear dependency
Center: Near 0 covariance - little linear dependency
Right: High negative covariance - high linear dependency

Now, covariance itself is harder to work with because the magnitude is unbounded and can be arbitrarily large depending on the data you are working with. To make covariance easier to work with, we introduce the idea of Pearson correlation, which standardizes the range of value for covariance to always be between -1 and 1. This is calculated by dividing the covariance by the product of the standard deviations of the two variables. The reason we choose Pearson is that it is the most widely used correlation formula.

$$Corr(x,y) = \frac{\sum_{i=0}^{N}(x_i-\bar{x})(y_i-\bar{y})}{\sigma_x \sigma_y}$$

By standardizing the covariance, we can easily compare the degree of linear dependence between variables. For example, if we have a correlation of 0.9 between Feature 1 and the label and a correlation of 0.6 between Feature 2 and the label, we can conclude confidently there is a much higher degree of linear dependence between Feature 1 and the label than between Feature 2 and the label.

▼ Mutual Information

Mutual information helps us understand the amount of *reduction in uncertainty* that one random variable provides for another. Mutual information takes on the range between 0 and 1 and is built upon the concept of uncertainty. The concept of uncertainty is quantified in information theory as entropy — commonly denoted by H. An entropy of 0 means a variable is completely predictable whereas an entropy of 1 means a variable is completely uncertain. For example, a fair coin toss has an entropy of 1 since we have no certainty as to what the outcome will be. We know it will be between 0 and 1 but we have

no way of predicting it other than pure chance. On the other hand, a coin rigged to always flip heads has an entropy of 0, since we always know the outcome.

To calculate the mutual information between two variables is to calculate how much we reduce the uncertainty of one variable by observing the other variable. Mathematically, mutual information, I, is calculated as follows.

$$\boldsymbol{I(x_1, x_2) = H(x_1) - H(x_1|x_2) = H(x_2) - H(x_2|x_1)}$$

Here $\boldsymbol{H\left(x_1|x_2\right)}$ is the uncertainty of variable $\boldsymbol{x_1}$ given variable $\boldsymbol{x_2}$. The detailed calculation of H itself is rather dense and is not something we would need to be aware of outside of the conceptual understanding that it falls between 0 and 1.

Let us assume for a moment that $\boldsymbol{H(x_1)}$ is completely uncertain, having an entropy of 1. We then introduce another variable $\boldsymbol{x_2}$, which helps us bring down the entropy of $\boldsymbol{x_1}$ down to a value of 0.1 denoted by $\boldsymbol{H\left(x_1|x_2\right)}$. This means that our mutual information is 1 - 0.1 = 0.9. Because knowing $\boldsymbol{x_2}$ greatly reduces the uncertainty (entropy) of $\boldsymbol{x_1}$, we conclude that there is high mutual information between the two variables. If $\boldsymbol{H\left(x_1|x_2\right)}$ was 1, meaning we are as uncertain as we were before observing $\boldsymbol{x_1}$, then mutual information, in this case, would be 0, which means knowing one variable does not help us understand the other variable whatsoever. This can be visualized better with a Venn diagram.

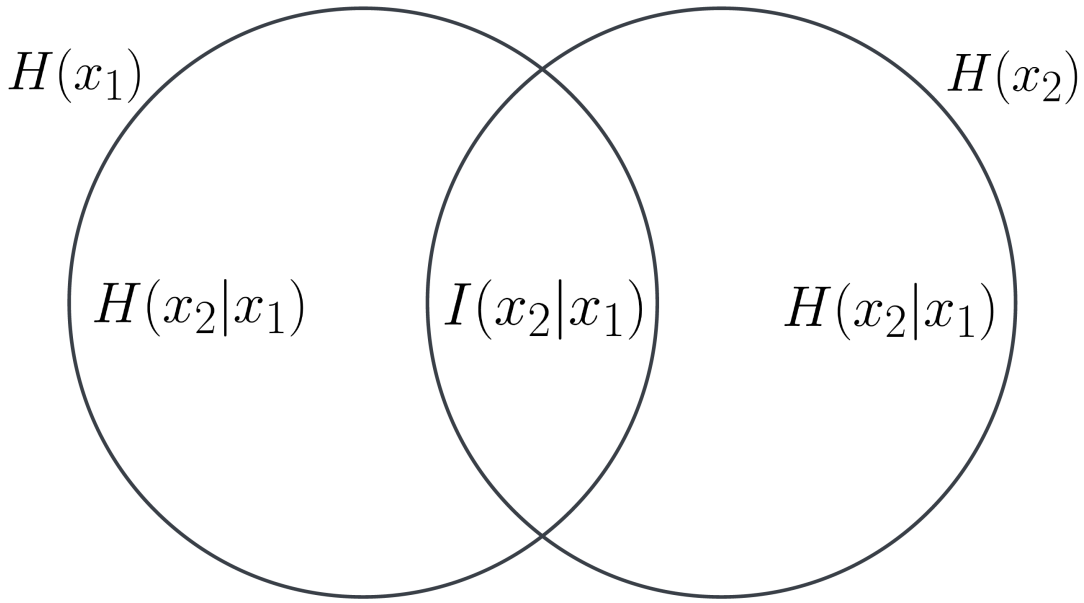


Figure 2. Mutual Information Venn Diagram Between Variable $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$

If we have a large overlapping region between $H(x_1)$ and $H(x_2)$, then we have high mutual information. Whereas if the two circles are completely non-overlapping, then there is no mutual information between the two variables.

How to Use Correlation and Mutual Information

Correlation and mutual information are particularly important in feature engineering. When we have a large number of features to work with, we want to select the features that are most relevant in predicting our label. Having too many features not only increases computing cost but also potentially reduces the generalizability of our model as it would try to learn from features that are less relevant. The proper approach is to select the features with highest correlation and mutual information against our label. In addition, we may consider any pairs of features that are highly correlated or have a high mutual information to be redundant, in which case we may choose to remove one of the features from our dataset.

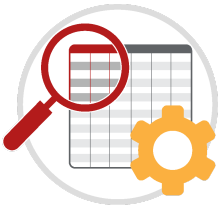
Back to Table of Contents

Module Wrap-up: Explore Your Data

Now that you've completed this module, you can see how exploratory data analysis can be used to further understand any patterns or interdependence among features and labels. You discovered the purpose of having descriptive statistics to quickly identify the degree of skewness and missing values. Mr. D'Alessandro also explained how to use univariate plotting techniques such as histograms to visualize skewness in a single feature or label. He also introduced bivariate plotting, which helps identify any feature-to-feature dependence as well as feature-to-label dependence.

Back to Table of Contents

Module Introduction: Find Outliers and Missing Data



Data are often “dirty” prior to processing. A common recurring theme in data management is the handling of outliers and missing data. In this module, Mr. D’Alessandro will focus on methods for identifying outliers and missing data as well as the methods for handling them.

Back to Table of Contents

Watch: Outliers

Outliers can pose a problem in machine learning, and statistics in general. Let's look at some examples of outliers in data and a few common ways to detect them.

Video Transcript

Now we're going to discuss a common data issue, which is the presence of outliers. An outlier, in simple terms, is a data point that is far from all of the others. The determination of outliers is subjective in nature, but nonetheless we can use exact data-driven methods to detect them. I'll provide examples in this video of both outliers and a few common ways to detect them. Here are two examples of outliers present in data. The left-hand plot is a univariate histogram. We can see that there is a small cluster on the right-hand side that is well separated from the bulk of the rest of the data. The right-hand plot is bivariate scatterplot. There is a small cluster towards the top middle that seems to be different from the distribution of the rest of the points. These two plots first demonstrate that outliers can be both univariate and multivariate in nature. Earlier, I said there was some subjectivity to the detection of outliers. On the right-hand plot is two lines. Each of these can represent our cutoff point for considering something an outlier. The subjectivity lies in the fact that there is a continuum of thresholds, and it is up to us to determine what is appropriate. Later, when we learn model selection techniques, we can consider outlier thresholds to be a feature parameter that we ultimately choose through experimentation. But if that seems daunting, a good rule of thumb is you want to only really consider no more than the top 1% to be an outlier. Outliers pose a few problems in machine learning and the related field of statistics. A lot of machine learning and evaluation relies on statistics, such as means and standard deviation. Outliers tend to skew mean values towards the outlier, and they increase the variance, which makes error bars larger. Last, an outlier can often be a clue that something went wrong in the data collection process. If you're in a position to investigate the source of your data, researching the outlier points may expose some error in the overall data processing. Additionally, machine learning algorithms generalize by extrapolating expected values between nearby points. Outliers, by definition, are alone, and the general geometric area where they lie is pretty sparse, so any extrapolation around these points is likely to be error prone. There are several ways to detect if a point is an outlier. The two methods we can consider here are the Z-score method and the interquartile range. Both methods use the natural variance of the data and outliers are defined based on how much they deviate from the normal variance. The Z-score method starts by computing the Z-score for each point. The Z score is just the point minus the average for that feature divided by the standard deviation. This normalizes all features to the same range. And so if the absolute value of the Z-score is greater than some threshold K, we would consider it an outlier. The interquartile range method starts by computing the interquartile range, which is the

distance between the 25th and the 70th percentiles in the data, and then any point that is K times greater than the interquartile range from the 25th or 70th percentile is considered an outlier. Again, here, we use K as our threshold. Here are two plots illustrating both methods on the same data. The left shows what we call the Z-score on the x-axis. The Z-score is the results, again, of transforming a variable by subtracting the mean from each point and dividing by the data standard deviation. In a normally distributed variable, most of the mass should be — or should have a Z-score less than 3. In this case, we see Z-scores as high as 6. On the right, we are showing a box and whisker plot. The whiskers here are tuned to two times the interquartile range, which is the range between the 25th and the 70th percentiles again. We can see here a cluster of points well outside of that. In both cases, we would want to define a threshold that determines what is an outlier and what isn't. Choosing this threshold, again, is a subjective call, but it's one that we can test. Now let's discuss what to do with an outlier or those outside the thresholds. The simplest approach is to just discard the example that has the outlier. This may seem extreme, but if outliers are pretty rare and especially if you suspect that the outlier is driven by an error with the data-generating process, then removing that example may actually serve you best. The other approach that I like is called winsorization. This is done by first identifying the outliers and then replacing them with a high but acceptable value. Here's an example of the same data being winsorized. Here I chose the top 99.9 percentile, which had a Z-score of around 3. I replaced the outlier points with this value, which leads to the spike on the right. The spike is a bit anomalous itself, but at least there isn't as heavy a skew in the data anymore.

Back to Table of Contents

Read: Common Statistics Refresher

This course will use common statistical terms and ideas. The explanations on this page will refresh your understanding of key concepts.

Statistics uses various methods to measure the centrality of data, which is a way of summarizing a lot of information in some kind of descriptive way. The three basic ways are mean, median, and mode.

Mean

A mean is an average of a series of numerical values obtained by adding up the numbers and diving by the number of values. There are other kinds of means (geometric, harmonic) that are calculated differently, but the term "mean" usually refers to the arithmetic mean, which is the sum of all the observations divided by the number of observations.

Median

A median is the middle number in a series of numerical values sorted from lowest to highest. If there is an even number of values in the series, the median is the mean (halfway point) of the two middle numbers.

Mode

A mode is the most frequent value in a series. If a series has more than one mode, it's considered multimodal. Multimodality is best demonstrated with a histogram that has two peaks, where each peak can be considered a "local" mode.

Other than centrality itself, it's often important to understand the spread of the data and the relationships of the individual values to some measurable centrality. These characteristics include variance, standard deviation, percentiles, z-score, and outliers.

Variance

Variance (S^2) is a measure of how spread out the data is from the mean. It's given by the equation $S^2 = \frac{\sum(x_i-\mu)^2}{n-1}$, where x_i is the value of each observation, μ is the mean of the observations, and n is the number of observations. Lower values mean the data is closer to the mean, higher values mean the data is more spread out from the mean.

Standard Deviation

Standard deviation (σ) is another measure of data spread. It is the square root of the variance (S^2) and described by the equation $\sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{n-1}}$, where x_i is the value of each observation, μ is the sample mean, and n is the number of observations. Like the variance, lower values mean the data is closer to the mean, higher values mean the data is more spread out from the mean.

Distribution

A distribution describes the way the data is spread out across its range (the lowest value to the highest value). This is easily depicted as the frequency (counts) of every unique value in the range of a series and typically visualized using histograms. Understanding the distribution lets you understand the concentration of values in your data. Some data distributions are so common that they are formally described by equations and parameters that govern their shapes. Examples of these include the Normal (Gaussian), Gamma, Poisson, and Direchlet distributions.

Percentile

A percentile is a value by which a given percentage of observations below. As an example, the 90th percentile is the value in a series below which 90% of all the data exists. The term Quantile is the same thing as a percentile but written as a decimal (90th percentile = 0.90 quantile). The term Quartile is a short-hand for the 25th, 50th, and 75th percentiles, where Q1 is the 25th percentile, Q2 is the 50th percentile, and Q3 is the 75th percentile. The Inter-Quartile Range (IQR) is the difference between the Q3 and Q1 percentiles given by $IQR = Q_3 - Q_1$.

Outlier

An outlier is a data point that differs significantly from other observations. This could be due to error (like putting a decimal in the wrong spot) or a true pattern that needs further investigation. There are different statistical ways of identifying outliers and a common one is to flag values that are less than or greater than $1.5 \times IQR$.

Z-Score

The z-score is a way of translating data to understand how many standard deviations away from the mean each point is. If a data point has a z-score of 1, that means it is 1 standard deviation above the mean. If a data point has a z-score of -1.5, that means it is 1.5 standard deviations below the mean. The z-score for each point is calculated by subtracting a value by the mean and dividing by the standard deviation. This is shown in the equation $Z = \frac{x-\mu}{\sigma}$: where x is the value, μ is the mean, and σ is the standard deviation.

Winsorization

Outliers can be removed from a dataset (and often are), or they can be modified in some way to prevent them from having a disproportional impact on the characteristics you are trying to

describe in your data. Winsorization is the method of clamping outlier data points at specific values derived from the data itself, like a percentile. Data can be Winsorized from both tails of its distribution by choosing a lower/upper percentile and capping points at those percentiles.

Univariate

Data is univariate if it contains one variable (uni = 1, variate = variable). An example of univariate data is measured dog heights without any other demographic information (e.g., age, breed).

Multivariate

Data is multivariate if it contains more than one variable. An example of multivariate data is measured dog heights along with age, sex, weight, and breed.

Back to Table of Contents

Assignment: Detecting and Replacing Outliers

In this exercise, you will practice finding and replacing outlier values in your data set.

This exercise will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

Read: Outlier Detection Methods

Data is not always clean for various reasons. In many cases, data can contain extreme outliers that are not a good representation of the dataset's distribution. Some of the possible reasons outliers can occur include the following:

- Human error (e.g., entry error)
- System error (i.e., integer overflow)
- Legitimate value, but is unrepresentative of the typical scenario

Typically a machine learning engineer would start out by plotting a feature or the label as a histogram, or in the bivariate scenario, a scatter plot. These plots allow us to easily visualize if outliers are present and what strategies we may need to utilize to further handle them.

☆ Key Points

An outlier can be present due to errors or legitimate reasons. In either event, they need to be handled as necessary.

Z-score tells us how far away a point lies from the bulk of its mass by using mean and standard deviation. Typically a z-score of 3 or above is considered an outlier.

IQR, accompanied by a box-and-whiskers plot, tells us how data points are distributed. Typically, points that lie beyond 2*IQR from Q1 and Q3 are considered outliers.

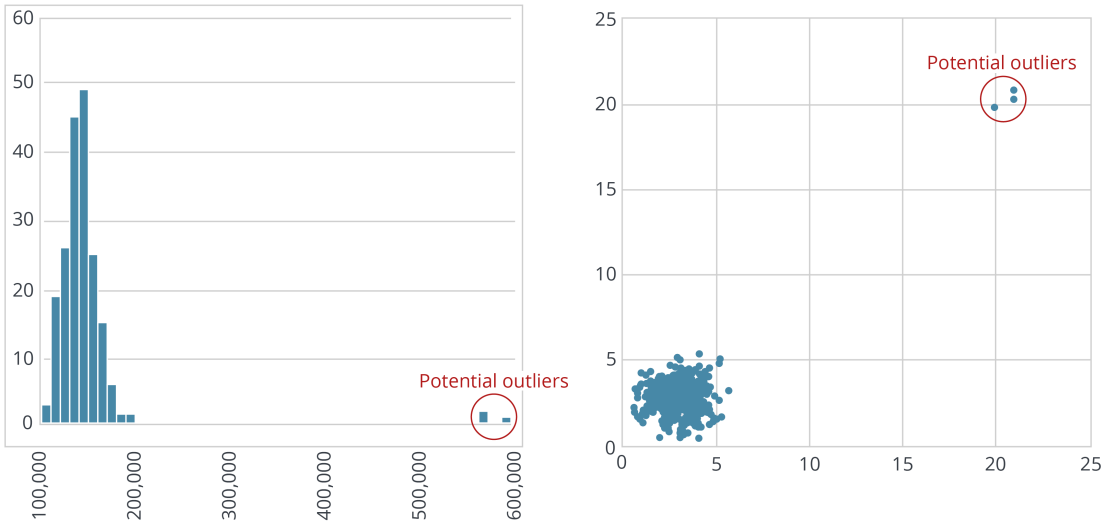


Figure 1.

Left: Histogram

Right: Scatter plot

There are two common approaches for detecting and separating out outliers — z-score and IQR.

Z-score

The idea of z-score is to take a data point x, subtract it by the mean of the dataset, and then divide it by the standard deviation of the dataset. Typically when a data point has an absolute value of the z-score above 3, it is considered an outlier. In a normal distribution, 99.7% of all points in a dataset are expected to fall within a z-score of 3. Z=3 is also an acceptable threshold, but the MLE can use any threshold and test them.

$$z = \frac{x - \bar{x}}{\sigma}$$

IQR

IQR or Interquartile Range seeks to identify a range where the majority of the data points lie.

IQR is defined by the following formula:

IQR = Q3 - Q1

Here Q3 is the median of the upper half of the dataset and Q1 is the median of the lower half. After we find the IQR, we then proceed to calculate the range of acceptable values by taking Q1 - K*IQR to get the lowest acceptable value and Q3 + K*IQR to get the highest acceptable value.

For example, let's say we have an array of numbers, [1, 10, 35, 37, 38, 40, 45, 46, 50, 84, 85]. The median of the array, in this case, is 40. From there, we split the array into a lower half and an upper half, with the lower half being [1, 10, 35, 37, 38] and the upper half being [45, 46, 50, 84, 85]. We take the median of each half and arrive at 35 and 50 respectively, where 35 is our Q1 and 50 is our Q3. The IQR is then 50-35=15. We then set K to be 2, and calculate K*IQR to get to 30. We then subtract 30 from Q1 to get 5 and add 30 to Q3 to get 80. From there, we conclude that any number that falls outside of 5 and 80 are considered outliers in our case. The choice of K depends on the nature of the problem and can be tuned often through experimentation.

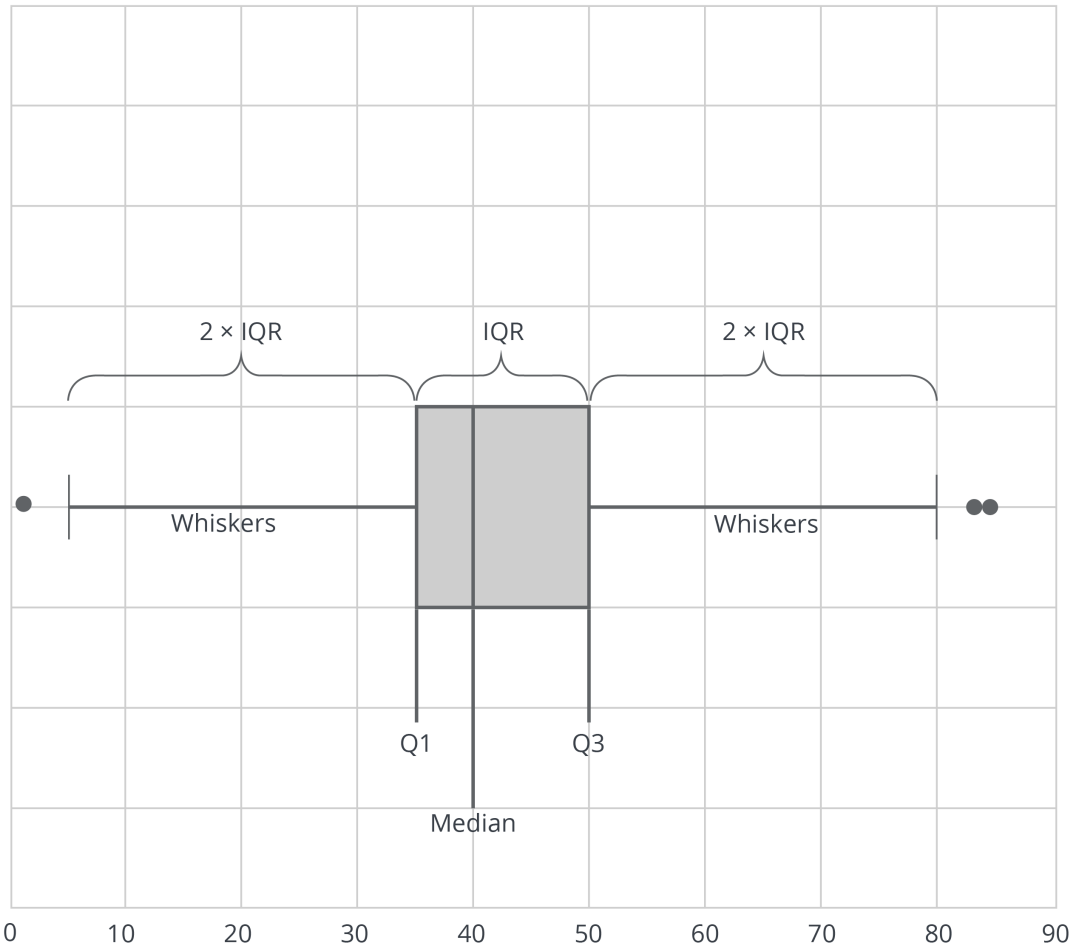


Figure 2. Box and Whiskers Plot

A box-and-whiskers plot is a plotting technique that utilizes IQR. From this plot, we can easily identify the general trend of our data's distribution. In a box-and-whiskers plot, 50% of the data points fall within the box and any other data points that fall within the whiskers are considered acceptable. Data points that fall outside of the two ends of the whiskers are considered outliers.

Back to Table of Contents

Watch: Missing Data

You will often have missing values in your data. This can happen due to system failures or timeouts, illegal values being passed into a data field, or incomplete forms or surveys. In this video, Mr. D'Alessandro will review what missing data looks like and how to address it.

Note: NaN means "not a number." It is the default marker for a missing value in Pandas.

Video Transcript

Having missing values and data is a pretty common problem. It can happen for a host of reasons, such as system failures or timeouts when logging data; illegal values being passed into a formatted data field; or actual missing-ness, such as from incomplete web forms or surveys. Knowing the cause is important if you have the ability to investigate. If you find consistent errors in your data, you probably want to trace the source and fix it. If you're handed a data set, you won't be able to do this, so you'll want to at least know how to work with the missing values. Let's review what missing data looks like. The indicator of missing-ness often depends on the system. Traditional databases will use "null" as a value; in text files, it could just be an empty field. When data is loaded into Pandas, missing values are typically represented by the NaN — which I call "naan" — data type. This is the ideal scenario because it makes detection the easiest. There is always the possibility that missing values are an acceptable condition by the system and that some developer at some stage encoded them in a special way. For instance, if a data point is supposed to be a strictly positive integer, you might see missing values encoded as negative numbers. Missing value detection should be trivial if the missing-ness is in the form of null or NaN values. We discussed in another video how the Pandas data frame describe method indicates missing-ness. There is also the ISNA method that can be used to detect missing nest at the record or the column level. Here I show how to display whether a particular column in your data frame has missing values with just one line of code. One of the first things you should do when you have a new data set is check for missing-ness. You usually want to know two things: Which columns have missing values and how many missing values are there? When we detect missing values, we'll want to definitely address them. Most machine learning software packages can't explicitly handle the missing values. They'll detect them, but they'll usually just drop the full record. Here are three most common ways to handle missing values. The first is to just drop the record or column. If the missing value count is very low, or if there are many missing values in the same example, you may just want to drop those specific examples. The latter case indicates that maybe some systematic error in the data collection happened, and I just wouldn't trust those data points anyways. Also, if a particular feature has a high percentage of missing values, you may also want to just delete that feature. If most values are missing, then the feature itself may not be

adding that much value to your model. If deleting is not an option, the next easier thing to do is called imputation. When you impute, you replace the missing value with another chosen value. The standard method is to use the mean or median. Once you choose the median — often you choose the median when the data is highly skewed, but you can really choose any value you like. The third option is called interpolation. When there is correlation amongst your features, you might be able to predict the real value from the other features. This amounts to building a model for every feature that has missing values, where you are using the remaining features as predictors. In these interpolation models, we treat the future of interest as a label and the other features as predictor features. We can then predict what the missing value should be based on the values of the other features. This last option can be seen as an extension of the middle option or imputation, but instead of replacing the missing value with the global mean, we replace it with the mean conditioned on some other features. Similar to processing outliers, when we clean missing values, there are subjective choices to make. These choices include the method and the values we might use for imputation if we use imputation. We can also think of these choices as pre-processing parameters that we ultimately test in our modeling experiments. Once we choose a pre-processing method, we need to ensure that any pre-processing we do on our training data must be performed exactly on any other data sets where we apply the model.

Back to Table of Contents

Assignment: Finding and Replacing Missing Data

In this exercise, you will practice how to find and replace missing data in your data set.

This exercise will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

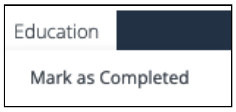
Back to Table of Contents

Assignment: Unit 2 Assignment - Building a Modeling Data Set

In this assignment, you will put into practice everything that you have learned this week to build a modeling data set. You can accomplish this assignment by using all of the techniques you practiced and implemented in the different exercises and activities this week.

This assignment will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This assignment will be graded by your facilitator.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

Assignment: Unit 2 Assignment - Written Submission

In this part of the assignment, you will answer six questions about building a model dataset and understanding your data through analysis and visualization.

The questions will prepare you for future interviews as they relate to concepts discussed throughout the week. You've practiced these concepts in the coding activities, exercises and coding portion of the assignment.

Completion of this assignment is a course requirement.

Instructions:

1. Download the **Unit 2 Assignment document.**
2. Answer the questions.
3. Save your work as one of these file types: .doc or .docx. No other file types will be accepted for submission.
4. Submit your completed Unit 2 Assignment document for review and credit.
5. Click the **Start Assignment** button on this page, attach your completed Unit 2 Assignment document., and then click **Submit Assignment** to send it to your facilitator for evaluation and credit.

Before you begin:

Please review **eCornell's policy regarding plagiarism** (the presentation of someone else's work as your own without source credit).

Back to Table of Contents

Module Wrap-up: Find Outliers and Missing Data

In this module, Mr. D'Alessandro explained how to identify outliers using histogram, scatter plot, and z-score as well as how to handle these outliers by discarding them or with Winsorization. When it comes to missing data, you also practiced how to identify them using built-in pandas functionality and how to further handle them by discarding the missing value or with imputation.

Back to Table of Contents

Lab 2 Overview

In this lab, you will practice building a modeling dataset where you will load data, remove and modify features, as well as Winsorizing outliers and building plots. You will be working in a Jupyter Notebook.

This 3-hour lab session will include:

- **5 minutes** - Icebreaker
- **30 minutes** - Concept Overview + Q&A
- **30 minutes** - Breakout Groups (Big Picture Questions)
- **15 minutes** - Sharing of Big Picture Group Responses
- **15 minutes** - Break
- **80 minutes** - Breakout Groups (Lab Assignment)
- **5 minutes** - Survey

By the end of Lab 2 you will:

- Load data and identify the number of records & columns
- Remove features that are not currently useful for analysis
- Modify features to make sure they are machine-comprehensible
- Build a new regression label column by winsorizing outliers
- Replace all missing values with means
- Identify two variables with the highest correlation with the label
- Build appropriate bivariate plots between the highest correlated features and the label

Back to Table of Contents

Assignment: Lab 2 Assignment

In this lab, you will continue working with the Airbnb NYC "listings" data set.

This assignment will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. After submission, the Jupyter Notebook will always remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This lab assignment will be graded by your facilitator.

The full contents of this page cannot be rendered in the course transcript. Log in to the course to view.

Back to Table of Contents

This content cannot be rendered properly in the Course Transcript, please log in to the course to view.