Introduction to Data Visualization with Python

by YK Sugi

Start Course

Bookmark Add to Channel Download Course

Course Overview

Course Overview

My name is YK, and welcome to Introduction to Data Visualization with Python. I am a software developer and a data scientist with experience at various software companies, including Google and Microsoft. Data visualization is often the first essential step in many types of data analysis projects, including scatter score analysis, machine-learning analysis, and exploratory analysis. In this course, we're going to learn several essential data visualization tools, what they are, when you should use them, and how to implement them in Python. Some of the topics we'll cover in this course include, line charts and time series, scatter plots, and what you should do when your data is too big. Before taking this course, you should be familiar with at least one C-like programming language, whether it's C++, Java, JavaScript, or Python. Extensive knowledge in Python is not required, and any prior knowledge in data visualization or statistics is not required either. So let's now get started with Introduction to Data Visualization with Python on Pluralsight.

Course Introduction

How This Course Is Structured

Hi, everyone; my name is YK and welcome to Introduction to Data Visualization with Python. I've worked at various software companies as a data scientist and a software developer in the past. I had many data analysis projects, and I used Python in many of them. This is a course I wish I had had before working on those projects. Not only will it give you tutorials on how to use Python to visualize data, but it will also give you fundamentals of data visualization in general. Each module from Module 4, the Histogram module, to Module 8, which is on what to do when your data is too big, is centered around a single topic; and each module consists of two components. The first video is about why that particular tool or topic is important and when you should use it. The second few videos after that explore how to implement it in Python. So if you'd like to get a quick overview of what kind of analysis you can conduct with data visualization, you can just watch the first video in each of these modules. If you are only interested in learning about how to implement it in Python, then you can watch the full-length videos after that. Of course, you can watch both of them as well. Module 3, the module right after this, will introduce you to the Python libraries we're going to use throughout this course; and Module 9 will give you some example problems to practice what you're going to learn throughout this course. Feel free to use them, as well, as you see fit. As you go through this course, I'd highly recommend practicing what you learn throughout this course with either your own data or with the data that I'm going to provide. If you have comments or questions about this course's material, feel free to post comments as well so learners can help each other.

Why Data Visualization?

Throughout my experience at various software companies, including Google and Microsoft, I realized that data visualization is often the first step in any kind of data analysis project. This could be A/B testing or any other kind of statistical analysis, exploratory analysis, data mining, or even machine learning analysis. In all of these cases, data visualization tends to be the first step in your analysis; and it's often the most crucial part. This is because data visualization gives you an intuitive understanding about your data and a quick way to test any hypothesis that you might have. It will also give you a different perspective as you slice data in different ways, which you can do quickly and effectively with data visualization. We're going to see some examples of these throughout this course.

Why Python?

What makes Python a great tool for data visualization and data analysis purposes? First of all, it's a solid general programming language and it's pleasant to work with. It also has many science and math libraries, which will make working with data much easier compared to many other languages. Python has many other kinds of libraries as well, and you or your team might already work with some of those. If so, if you use Python for data visualization purposes, it will be relatively easy to integrate with your existing infrastructure and skill set.

Who Is This Course For?

This course is intended for software developers who are looking to learn how to visualize data, data scientists or data analysts who might already be familiar with data visualization but want to learn how to implement it with Python, and any other IT professionals or learners who are interested in learning more about data analysis and data science. Before taking this course, you should be familiar with at least one C-like programming language, whether it's C++, Java, JavaScript, Ruby, or Python. Basic knowledge in Python will be useful, but it's not required before you take this course; and no prior data visualization or statistics knowledge is required either. So let's get started!

Introduction to Jupyter, Pandas, and Matplotlib

Installing Libraries Through Anaconda

In this course we're going to use the following tools. Python, which is one of the most popular programming languages. Particularly for data visualization purposes. Matplotlib which is a popular data visualization library for Python. Pandas which we're going to use for importing and processing data. And Jupyter which is an interactive Python notebook. Now, to install these tools, I recommend using something called Anaconda. Anaconda is a package manager for a scientific library in Python. And when you install it. It comes with all the tools I mentioned earlier, including Python. So there is no need to separately install Python or any other tools when you install Anaconda. Whether or not you have Python on your computer already. So let's see how we can install these tools using Anaconda. Anaconda is provided by the company called Continuum Analytics. So first find their website by searching for Anaconda Python. Or by directly going to

continuum. io/downloads. Once you're on Anaconda's download page. Scroll down and make sure that the right platform is selected. Whether it's Windows, Mac or Linux. And click the button for an installer. I recommend using a graphical installer and Python 3 instead of Python 2 since Python 3 is more up to date. Once the installer is downloaded. Open it and follow through the installer dialog with the default options. And Anaconda will be installed. The installation process will take a while so come back to the video when it's done. Once Anaconda is installed you'll be able to launch Jupyter. The interactive Python notebook application through something called Anaconda navigator. So launch Anaconda navigator as you would launch any other application on your computer. Dismiss any pop-up dialog that might show up. And click launch on Jupyter notebook. We'll look into how to use Jupyter notebook in the next video.

Using Jupyter for the First Time

Jupyter notebook gives you a notebook like interface to write, test and organize your code in a really nice way. And it's composed of two parts. The first part is a local server that starts running with a Python interpreter once you start Jupyter notebook on your computer. It's combined with an interactive browser interface on whatever browser you'd like to use. You don't have to worry too much about how it works exactly. But the only thing to keep in mind is that when you start Jupyter notebook you might see a command line interface window like this. If you accidentally close this window it might stop the local server and your Jupyter notebook might stop working. So be careful not to close it by accident. So let's see how to use Jupyter to create a new notebook file and Hello World script as well as how to use two different types of cell in Jupyter notebook. First, launch Jupyter notebook through Anaconda navigator. Or with any other method you might be familiar with. And notice that the url that's shown on the browser should be something like localhost:8888. This is a local port that Jupyter notebook uses. So if you close the browser tab or window. You can open Jupyter notebook again just by going to the same url. Now let's create a new notebook file. Currently we're on the home directory as you can see the home icon on the UI. Navigate to the directory where you want to create a notebook file. Let's say desktop. Find the new button on the right and select the Python version you'd like to use. I'm going to pick Python 3 here. Let's rename this file to something more meaningful than untitled. Say using Jupyter for the first time. If you now go back to desktop you see that a new file is created and it's called using Jupyter for the first time. ipymb.. ipymb is the extension for Jupyter notebook. You can write and execute regular Python code in a Jupyter notebook. For example, let's write print hello world in the cell. You can execute this code by clicking the run cell button at the top. If you define a variable by writing say a = 2 it is shared across all cells. So you can write

print(a) and it will print out the variable of a. Now there are two main kinds of cell in Jupyter notebook. Code and markdown. What we've been using so far is code. Markdown cells are used for as you might guess formatting text. In a markdown cell for example. You can write # Header 1 # # Header 2. This is a markdown cell. When you click the run button at the top. It's formatted in a really nice way. And there are few other useful commands to remember. To add a new cell you can click the plus button at the top. And to delete a cell you can click the scissor button at the top. Also, there are many keyboard shortcuts for the most common access. I'm going to cover some of them throughout this course. But to view them you can go to the help menu. And click keyboard shortcuts.

Using Matplotlib for the First Time

Matplotlib as I mentioned earlier is a popular data visualization library for Python. And the reason we're going to use it is because it's fairly easy to use. And out of many Python data visualization libraries it's the most commonly used one. With Matplotlib you'll be able to create many different types of charts. Including scatter plots. Bar charts and line charts or time series. So let's see how to create a line chart with Matplotlib and how to save a chart that you created as an image file. To use Matplotlib you'll first need to import the pyplot module from the Matplotlib library. You can do this with from matplotlib import pyplot as plt. This says import the pyplot module from the Matplotlib library and call it plt. Note here that the pyplot module is just a simple interface for using Matplotlib. Once pyplot is imported you can create a plot with plt. plot(This says create a plot with 1, 2, 3 on the X axis and 1, 4, 9 on the Y axis. To show this plot you need to write plt. show and run the cell. I'm going to use a keyboard shortcut for running a cell here. Shift enter. You can add labels for the X axis and the Y axis as well with plt. xlabel this is x axis and plt. ylabel this is y axis. You can add a title with plt. title. And you can see that the plot is updating with the access labels and the title. It's also possible to put multiple lines on the same plot by calling plt. plot multiple times. To clarify which line represents what. You can just add a legend with plt. legend. Let's call these two data sets data set 1 and data set 2. Note that plt. legend takes a list of strings as an argument. Once you run the cell again you'll see the legend. Now, there are two ways to export a plot that you created as an image. The first method is just by downloading image. Using the browsers built in function. For example in Chrome you can right click on an image and click save image as. Another way to save a plot as an image is by using the plt. savefig function. Instead of calling plt. show just call plt. savefig. This function takes a file name as an argument. So let's call the exported file exported image. And when you run the cell this file is

exported as a pmg image in the same directory as the Jupyter notebook file that you're editing. In the next video we'll learn how to use Pandas to create a new data frame object.

Using Pandas for the First Time

Pandas is a Python library that helps you import, organize and process data. If you're familiar with R, you'll see that it's similar to dataframes in R. In fact, the primary data structure in Pandas is called dataframe. So let's see how to create a dataframe using Pandas. Select data with something called Boolean indexing. And make plots using data in a dataframe. Let's first import pyplot as plt and Pandas as pd. These labels by the way are standard conventions. But you don't necessarily have to follow them. Suppose here as an example you hosted a series of events in 2008, 2012 and 2016. Let's say you have information about how many attendees were there each year. And the average age of those attendees. To create a dataframe to store this information. We're going to create some dummy data as a dictionary with three attributes. Year, attendees and average age. This reads for example in 2008 you had 112 attendees and the average age of those attendees was 24. We can parse this dummy data as a dataframe with pd. dataframe with this dictionary as the argument. We'll save the resulting data in a variable called df and let's see what it looks like. You can see that there are three columns corresponding to the attributes in our dummy data. Year, attendees and average age. You can select a single column out of this dataframe for example with df['year'] This selects the year column. The type of this new data is something called a Pandas series. It's similar to a regular Python list and also to a NumPy array if you're familiar with NumPy. You can apply an inequality operation on the series with df['year'] < 2013 This gives back a series of Boolean values. True where the year is earlier than 2013. False when the year is equal to or larger than 2013. Let's store this data in a variable called earlier than 2013. Using the Boolean series you can select only the part of the data where the year is earlier that 2013 with df This is called Boolean indexing. Now let's say that you want to examine how the number of attendees has changed for the last three events. To do this you might want to plot number of attendees against the year. There are a few different ways to do this. But I recommend using the plt. plot function as a way saw in the last video. Since it's the simplest way to do it. So just write plt. plot(df['year'], df['attendees']) This of course says plot the year on the X axis and plot the number of attendees on the Y axis. If you want to plot the number of attendees and the average age on the same plot. We can just call plt. plot multiple times. So here, the second line will be created with plt. plot(df['year'], df['average age']) And let's also add a legend to clarify which line represents which attribute as well with plt. legend(['attendees', 'average age']) In the next video we're going to examine another way of creating a Pandas dataframe.

Importing Data with Pandas

In the last video we learn how to create a Pandas dataframe from a Python dictionary. In this one, I'm going to show you how to create a Pandas dataframe by importing data in a CSV file. The sample data that we're going to use here is called courtries. csv. It contains the information about countries and their basic demographics for each year. Years ranging from 1952 to 2007 for every five years. So I'm going to show you how to import this data as a Pandas datagframe and plot it using Matplotlib. As usual we need to first import Pandas spd and pyplot as plt. To import our csv data, countries. csv. First make sure that this distributor notebook file is in the same directory as countries. csv. Then use pd. readcsv to import this data as a Pandas dataframe. And you can use a tab completion here to import the file name. So just press tab after typing a few characters. Let's store this in the new variable called data. And let's check what's inside. You can see that we have the data from countries. csv as we expected. To check the content of the data you can also write data. head. Which gives us the first five rows of the dataframe. Now let's say we want to plot how gdp per capita has changed over time in Afghanistan. To do that we need to isolate the data about Afghanistan from the data variable. To select the country column you can write data['country'] Or you can also write data. country. Either syntax does exactly the same thing. Using this and Boolean indexing you can select only the data about Afghanistan with data[data. country == 'Afghanistan'] Let's store this in the variable called Afghanistan. And we're going to plot it with plt. plot(afghanistan. year, afghanistan. gdpPerCapita). And that's how to import data from a csv file using Pandas and plot it using Matplotlib. In the next module we'll examine how and when to use histograms.

Finding Distribution of Data with Histograms

When to Use Histograms

Suppose you're a software developer who's in charge of a website. And you're in charge of making sure that pages on your website always load fast. One day, you notice that the average page loading time in June is significantly slower than any previous month. In this chart, you can see that the average page loading time in June is almost 1. 4 seconds, whereas in any previous month, it's been just about one second. What should you do about it? There are a few choices here depending on the situation, but one of the those options is to create a histogram of page

loading times. If you create a histogram of page loading times in May, the month before you saw the sudden increase, it might look like this chart. It shows you the distribution of page loading times in this month. As you can see, there are almost 1, 000 occurrences of a page loading time that's less than 0. 1 seconds, and about 900 occurrences of a page loading time between 0. 1 seconds and 0. 2 seconds and so on. And note here that any value larger than three is bundled into the last bin. This is a common practice when there are extreme outliers that make it difficult to analyze this data. Suppose you want to compare this histogram with one for June, when the average page loading time suddenly became much slower. And suppose that the amount of traffic that you get in June is about the same as the one in May. What you might see is a histogram like this. This tells you that the page loading time has been affected and became slower across the spectrum. Alternatively, you might see a histogram like this one. If that's the case, you can see that only those on the slower end have been affected, and have become even slower. Once you have this information, you might then focus your analysis effort on those users, perhaps users on slower internet connections, so you can find the root cause of this performance regression. This is an example that's somewhat exaggerated, but in general, histograms help you understand the distribution of a numeric value in a way that you cannot with simple mean or median alone.

Creating Histograms with Matplotlib

Now let's see how we can create histograms with Matplotlib. Suppose as an example, we want to compare GDP per capita of different countries in Asia and Europe in 2007 using histograms. As usual, we're going to use our sample data, countries. csv. The important columns here are continent and year, so we can isolate the data on Asia and Europe in 2007. And of course, GDP per capita. In case you are not familiar, the GDP per capita is the annual GDP in the given country divided by its population, and it's shown in 2005 U. S. dollars in our data. In this demo, we're going to see how to create histograms with Matplotlib, and how to use Matplotlib's subplot function to compare multiple plots in the same graph. First, import pandas and pyplot as usual, and import countries. csv with pd. read_csv. To check which continents are included in the data, we can use Python's set function. Set data. continent will give us a set of unique values out of the data. continent series. You can see that we have five continents in this data, Africa, Americas, Asia, Europe and Oceania. Now we want to select the data on Asia and Europe in 2007. Let's do this in two steps. First select the data for 2007 with data square brackets, data. year equals 2007. Let's put this in a new variable called data_2007. Then select the data for Asia out of data_2007 with data_2007, square brackets, data_2007. continent equals Asia. Put this in asia_2007 and do

the same for Europe. Make sure that the right data is selected with asia_2007. head and europe_2007. head Here you can see that europe_2007 only includes data from Europe and 2007, as we expected. To check how many countries there are in each of these data sets, you can first find the set of countries in say asia_2007 with the set function. Then use the length function to find the length of the set or the number of countries. And do the same for Europe. As you can see, there are 33 countries in the Asian data set while there are 30 countries in the European data set. This is nice because having roughly the same number of countries in each region makes it easier to compare the distribution of GDP per capita. Now let's first find the mean and median of GDP per capita in Asia and Europe in 2007. You can do this with asia_2007. gdpPerCapita. mean and asia_2007. gpdPerCapita. median And do the same for Europe as well. From this, you can see that the mean and median GDP per capita in Europe is much higher than that of Asia. For example, the median GDP per capita in Asia is about 4500 dollars, while the median GPD per capita in Europe is about 28000 dollars. But from mean and median alone, it's hard to tell what the distribution of GDP per capita is really like. So let's create some histograms. To create a histogram of GDP per capita in Asia, you can write plt. hist asia_2007. gdpPerCapita. And let's show this graph with plt. show. To make it look nicer, let's make the edge color black. And adjust the number of pins. The number of pins is given as a second argument to the hist function. If we set it to 20 for example, that means that there are 20 buckets or bins in which each occurrence of GDP per capita can fall. Now to compare this with a histogram of GDP per capita in Europe, we're going to use pyplot's subplot function to create subplots within the plot. Plt. subplot takes three arguments. The first two arguments express the structure of the subplots. If we write 2, 1 then that means that we want a canvas that can contain two rows and one column. Then the third argument specifies the index of the subplot that we want to select where the index starts at one. So plt. subplot two one one says you want to create a canvas where there will be two rows and one column of subplots and that you want to select the first one of those for the histogram for Asia. You might also see this written as subplot two one one, but it does exactly the same thing. To plot two histograms on the same chart, we'll need to call plt. subplot twice and the second time is going to be after creating a histogram for Asia. And this time with the arguments two, one and two. This says the graph needs to be able to contain two rows, and one column of subplots, and that we want to select the second one of those. Once the correct subplot is selected, we can then create a histogram for Europe with plt. hist europe_2007. gdpPerCapita, 20 edge color=black. Once you run this cell, you see the two histograms on top of one another. And let's add white labels to make it clear which graph represents which continent. Let's also add a title. I'm going to call this graph distribution of GDP per capita. Now you'll notice that because the ranges of the X axis are different for these two histograms, it's hard to compare these two charts.

So let's manually specify the range for each of these histograms with the range argument. The range argument is one of the arguments in the plt. hist function and we'll set it to 0 to 50, 000 by writing range equals (0, 50000). And we'll do the same for Europe. Once these two histograms have the same range, it's much easier to compare them. Examining the plots, you can see that while there are wealthy countries in Asia too, many of them are extremely poor with about 20 countries whose GPD per capita is less than 5000 dollars a year. As we discussed earlier, this is an example of how histograms give you much more information than plain mean or median.

Practice Problem 1 - Histograms

To practice what you've learned in this module so far here's a practice problem for histograms. The problem is compare Europe and America's life expectancy in 1997 using countries. csv. Americas here include North America, Central America and South America. To solve this problem, use Matplotlib's hist function to create histograms and the subplot function to compare multiple histograms in a single chart. If you want to try solving it on your own, pause the video right here and come back for an example solution when you're done. And here's an example solution. First of all, of course, import pandas and pyplot as well as our data, countries. csv. Then select the data for Americas and Europe in 1997 in two steps. Let's first put the data on 1997 in data_1997. Then put the data on Americas and Europe in americas_1997 and europe_1997. And let's check the number of countries in each data set. We can do this by finding the length of the set of americas_1997. country and by doing the same thing for Europe. As you can see, there are 25 countries in americas_1997 and 30 countries in europe_1997. Now let's find the mean and median life expectancy in Americas and Europe. We're going to do this by printing americas_1997. lifeExpectancy. mean and median, and also the same for Europe. From this, you can see that the mean and the median life expectancy in Europe are slightly higher than those of Americas. Let's now plot histograms of life expectancy in Americas and Europe on the same chart using the subplot function. We're going to plot a histogram of life expectancy for Americas with plt. hist americas_1997. lifeExpectancy and with edge color black. And we'll do the same for Europe. Let's also add a title and the Y labels here. I'm going to call this chart distribution of life expectancy. And I'm going to add Y labels for Americas and Europe. Let's set the range of the X axis to say from 55 to 85 so it's easier to compare these two histograms. We can try different values of the number of bins to make the chart easier to read as well. Let's try using 20 bins here. As you can see by comparing these two histograms, there are many more countries on the higher end in terms of life expectancy in Europe than in Americas. You might also notice that there are even a few outliers that have very low life expectancy in Americas. And you might wonder which

countries these are. We can find which countries they are by selecting only those countries that have low life expectancy of less than 65, with americas_1997 square brackets americas_1997. lifeExpectancy less than 65. And it turns out, those countries are Bolivia and Haiti. That's it for this module, and in the next module we're going to look into line charts and time series, one of my favorite tools in data visualization.

Creating Time Series with Line Charts

When to Use Line Charts / Time Series

In this module, we're going to look into line charts and time series. They are some of my favorite tools when it comes to data visualization, so I would say it's important for anyone interested in data visualization to master them as well. Now you're probably already familiar with line charts, but what is time series exactly? For the purpose of this course, a time series is any chart that shows a trend over time, and it's usually a line chart. Here's an example of a time series that's been created with Matplotlib. It shows how President Obama's approval ratings changed over the course of eight years in office, and as you can see, it's a combination of a line chart and a scatter plot. So when should you use line charts and time series? As an example, suppose you are a marketing manager for an online store. You started selling some kind of popular product recently, and you want to see what kind of customers are buying this product. So you started analyzing the sales data, and you found this piece of data. When you looked at the sales volume on a particular Sunday, it turns out there are a lot more male buyers than female buyers. There are about 450 units sold for male customers, versus about 350 for female customers. So you might conclude, this product is more popular with males than females. With this information in mind, you might then start targeting male customers in your marketing strategy. But can you actually conclude from this graph alone that this product is more popular with male customers than female customers? Well, not necessarily. For one thing, there are only about 800 units sold in total here, so the sample size is very small. And even if the difference is statistically significant, it's possible that male customers tend to buy this product more than female customers only on Sunday. So to make your analysis much more robust, one thing you can do is to plot a line chart over time, and make a time series for sales for male customers and female customers. When you do that, you might see a chart like this. If you see a chart like this, you can be more confident of

your conclusion that male customers buy this product more than female customers, because the difference is consistent over time. However, you might also see a chart like this one. Then you wouldn't be able to make the same conclusion anymore. Perhaps it was just a coincidence that more male customers bought this product on Sunday, or perhaps more male customers buy this product only on Sunday, but not on any other day. To summarize the main reasons why time series and line charts are so powerful are because first of all, it's a convenient way to examine the trend over time. And if you have a particular hypothesis that you want to test, or an experiment that you're running, time series and line charts allow you to test it on a variety of conditions, for example on different days of the week, and not just on Sunday. This makes your analysis much more statistically robust, and it reduces misinterpretation of your data.

Creating Line Charts with Matplotlib

Now, let's see how to create line charts and time series with Matplotlib. Suppose you want to compare GDP per capita growth in the U. S. and China. Since we want to compare GDP per capita's trend over time, we're going to create a time series with a line chart. As usual, we're going to use countries. csv, and the important columns to you are Country, so you can select the right data for the U. S. and China, as well as Year and gdpPerCapita to find the trend of GDP per capita over time. Let's now dive into a demo in which you are going to learn how to create line charts with Matplotlib, how to use the iloc syntax to select an item in the panda series, and how to multiply or divide with a scalar. First, as usual, import pandas and pyplot as well as the countries data. After that, let's first examine how the GDP per capita in the U. S. has grown over time. You can select the data for the U. S. With data[datacountry == 'United States'] Let's put this in the variable called us. Then, you can plot the U. S. 's GDP per capita growth over time with plt. plot(us. year, us. gdpPerCapita). This says, plot us. year on the x-axis, and us. gdpPerCapita on the y-axis. Let's show this plot with plt. show. And let's add a title, an x label and a y label to make it easier to read. As you can see, the GDP per capita in the U. S. was about $15, 000 per year in 1950s, and it grows to more than $40, 000 in 2000s. As I mentioned earlier, the unit here is the 2005 U. S. dollar. Let's now compare this to China's GDP per capita growth. We'll grab the data for China with data[data. country == 'China'] Let's put this in the variable called China. We can plot the U. S. 's GDP per capita growth and China's GDP per capita growth on the same chart by calling plt. plot multiple times. First with plt. plot(us. year, us. gdpPerCapita) and then do the same for China. Show this with plt. show. Let's also add a legend to make it easier to understand which line represents which country, and let's add an x label and a y label as well. From this graph, we can see how GDP per capita has grown both in China and the U. S., but it's hard to

know which one has grown faster on the relative term because the absolute values are so different. What if, instead of comparing the absolute values, we wanted to compare the growth instead? To do this, let's set the GDP per capita in the first available year to 100, and compare how it grew in the U. S. and China with respect to that year. So what we want to do here is, we want to divide each year's GDP per capita by the first available year's GDP per capita. To select the first available GDP per capita, you can just write us. gdpPerCapita. iloc Iloc stands for Integer Location, and it allows you to select an item with an integer index in us. gdpPerCapita, which is a panda series. So us. gdpPerCapita. iloc will give us the first value in the us. gdpPerCapita series. If we just write us. gdpPerCapita, it'll give us this series, which represents GDP per capita for different years in the U. S. And us. gdpPerCapita. iloc gives us the first value of the series, which is the GDP per capita in the first available year. You can divide each value in the us. gdpPerCapita series with this scalar value, with us. gdpPerCapita divided by us. gdpPerCapita. iloc As you can see with this, the first value is set to one. We wanted to set the first value to 100, so we'll need to multiply this whole series with 100. Now the first value is set to 100, and the subsequent GDP per capita shows the relative value to the first available year's GDP per capita. Let's store this in a variable called us_growth. Do the same with China, and store the resulting series in china_growth. We can compare GDP per capita growth on the relative term by plotting U. S. growth and China growth on the same graph, just like we did before. Notice here that by calling plt. plot function twice, we're plotting US growth and China growth on the same graph, instead of the raw GDP per capita values. From this graph, it's now obvious that the GDP per capita in China has grown much faster, relatively speaking, than that of the United States. In the next video, I'm going to introduce you to an example problem to practice what you learned in this video.

Practice Problem 2 - Line Charts / Time Series

Here's an exercise problem for practicing what you have learned in this module so far. The problem is, compare population growth in the U. S. and China. For this problem, you can use the population column in the countries. csv data. And try comparing both absolute numbers and relative growth of a population. And if you want to try solving this problem on your own, pause the video right here. And here's an example solution. After importing pandas pyplot and countries. csv, select the data for the U. S. and China with us = data[data. country == 'United States'] and china = data[data. country == 'China'] Store them in variables called U. S. and China. Then you can plot the populations of the U. S. and China with plt. plot(us. year, us. population) and the same for China. Show this graph with plt. show. It's hard to read raw data since the numbers are so large, so let's divide the populations by one million, or 10 to the power of six, so

we can get numbers in millions instead. And let's add a title, a legend, x and y labels to make this graph easier to read. This is how to compare the population growth on an absolute term, but how can we compare the population growth on the relative term? To do this, let's see how the population has grown in the U. S. and China compared to the first year in our data. Just like in the previous example, we're going to set the first year's population to 100, and we'll compare how much it's grown since. We can do this with us. population divided by us. population. iloc which is the first available year's population in the U. S., times 100. Let's put this in the new variable called us_growth and let's do the same thing with China. Let's now plot us_growth and china_growth as line charts with plt. plot(us. year, us_growth), plt. plot(china. year, china_growth). And let's add a title, a legend, x and y labels as well. From this graph, you can see that the population grew in the U. S. and China at about the same rate up until early 1960s, but after that the population in China grew much faster than in the U. S., relatively speaking. This is the end of the module, and in the next module we're going to examine scatter plots.

Examining Relationships in Data with Scatter Plots

When to Use Scatter Plots

In this module, we're going to examine when and how to use scatter plots. Scatter plots give you a convenient way to visualize how two numeric variables are related in your data. And here's a simple example of a scatter plot. It shows how weights and heights are related in a hundred people. Obviously, you can see that the taller someone is, the more he or she tends to weigh, but there's some variation, as you can see. So this is a very simple example. But when should you use scatter plots in a real world situation? Suppose as an example you're a sales executive at a Fortune 500 company, and you're in charge of your company's entire sales force, about 200 salespeople in total, and you want to see how to improve their performance. And in particular, you want to see how the amount of experience is related to each salesperson's performance. One way to analyze this is with a scatter plot. You can plot years of experience at your company, say, Company X on the X axis, and the sales volume for the last quarter in dollars on the Y axis. Then you might see a graph like this. From this chart, you can see that in general, the more experience a salesperson has, the more sales he or she brings. But you can also see that there are a few outliers who have very little experience compared to other salespeople but still have some

extraordinary performance in the last quarter. Once you identify who they are, you might then interview then to see what they do differently that allowed them to do so well. Whatever their techniques are, perhaps you can spread that knowledge to other salespeople in the company to increase the sales figures for the entire company. So, in summary, scatter plots help you better understand relationships between multiple variables, for example, weight and height or years of experience and the sales volume. And in the process, sometimes it helps you find outliers as well. And in the next video, we're going to see how to create scatter plots using Matplotlib.

Creating Scatter Plots with Matplotlib

We're now going to examine how to create scatter plots with Matplotlib. Suppose, as an example, you want to find how GDP per capita and life expectancy are related to each other in different countries. To do this in our countries data, the columns you'll need to use are lifeExpectancy and gdpPerCapita, as well as year so you can find the relationship between life expectancy and GDP per capita for each given year. Let's now dive into a demo in which we're going to create scatter plots with the scatter function of Matplotlib. We're also going to use the log10 function in the Numpy library as well. If you used Anaconda to install Jupyter and other libraries, you should already have Numpy installed, because it comes with Anaconda. If not, make sure to install the Numpy library before you follow this demo, and we're going to use Python's for item in syntax to run a for loop over the given years. First, as usually, import Pandas and Pyplot. We're also going to import Numpy as np. Let's import our data with pd. read_csv as well. After that, let's first examine how GPD per capita and life expectancy are related in 2007, the most recent year that's available in our data. To do this, we're going to select the data for 2007 with data[data. year == 2007] and we'll put it in a variable called data_2007. You can create a scatter plot with gpdPerCapita and lifeExpectancy in 2007 with plt. scatter(data_2007. gdpPerCapita, data_2007. lifeExpectancy.) Show this with plt. show. You can adjust the size of each dot in the scatter plot with a third argument in plt. scatter. We can try different values to see which one looks right. Let's set it to five here. And let's add a title, an X label, and a Y label to make the graph easier to understand. As you can see, the higher the given country's GDP per capita is, the higher its life expectancy tends to be. We can double-check this by finding a correlation between GDP per capita and life expectancy in 2007. We can do that with data_2007. gpdPerCapita. corr for correlation, with the argument (data_2007. lifeExpectancy). As you can see, GDP per capita and life expectancy in 2007 are highly correlated. Now, if you go back to this scatter plot, you might notice that GDP per capita and life expectancy don't seem to have a linear correlation. The life expectancy goes up very quickly when GDP per capita goes up in the lower range, but as the

country has more GDP per capita, an additional increase seems to have less effect on its life expectancy. For example, this is evident if you compare the jump in life expectancy from a very low GDP per capita to $10, 000 to the increasing life expectancy from $30, 000 of GDP per capita to $40, 000. So, you might think that you should look at GDP per capita not in a linear scale, but in a log scale. To do this, we're going to use the np. log10 function. Np. log10 is a Numpy function that computes the log of the given numbers with the base 10. So, if we write np. log10 with the argument being a list of 10, 100, and 1, 000, it will give us one, two, and three. This is because of course 10 to the first is 10, and 10 to the second is 100, and 10 to the third is 1, 000. Let's now make a scatter plot with gdpPerCapita on the log scale with plt. scatter(np. log10 of data_2007. gdpPerCapita,) data_2007. lifeExpectancy. This says draw a scatter plot not with raw gdpPerCapita, but with the log of gdpPerCapita with the base 10. Show this with plt. show. Now that the GDP per capita is shown in a log scale on the X axis, if you look at three on the X axis, it's 10 to the power of three in raw gdpPerCapita, which is $1, 000, and if you look at four, it's 10 to the fourth, which is $10, 000. This time, the graph seems to show more of a linear correlation. Let's find a correlation between the log of gdpPerCapita and lifeExpectancy in 2007. It turns out, it's about 0. 8. Comparing it to the correlation between raw GPD per capita and life expectancy, you can see that the log of gdpPerCapita is more highly correlated with lifeExpectancy in 2007. So this seems to be a more accurate model here. Now, let's run this analysis for each available year in our data. To do this, we're going to run a for loop for each year. We're going to first find the set of all the years that are available with the set function. Then we'll make sure that it's in the correct order with the story function. Let's put this in a new variable called years_sorted. After that, we can run a for loop for each year with for given year in years_sorted. Select the data for this year with data_year = data Then, make a scatter plot of gdpPerCapita and lifeExpectancy with plt. scatter(data_year. gdpPerCapita, and data_year. lifeExpectancy, and five. Let's set the title of this chart to given_year and show the chart with plt. show. Once you examine these plots, you see that the ranges for the X axis and Y axis are considerably different depending on the year. For example, in 1972, the X axis shows up to more than $100, 000, while in 1977, the X axis shows up to only about $60, 000. This makes them hard to compare with each other. And there's also an outlier with GDP per capita of more than $100, 000 a year, even in 1952. So, let's set a consistent range for the X axis and Y axis and ignore the outlier with plt. xlim zero to 60, 000, which sets the range for the X axis to zero to 60, 000 and plt. ylim 25 to 85, which sets the rang for the Y axis. Let's also add an xlabel and a ylabel as well. After all that, these graphs are much easier to compare with each other, so let's export them as images, examining how they have changed over time. We're going to do this with the plt. savefig function instead of running plt. show. The first argument in this function specifies the file name, and we're going to use str of

given_year here. We need to use the str function to convert the given_Year to a string, because plt. savefig expects a string argument. And remember that dpi stands for dots per inch. This number specifies the resolution of the exported image. After exporting this image, we'll need to clear the current plot before drawing the next one with plt. clf. Once this for loop finishes running, let's see what the exported plots look like. In 1952, most countries' GDP per capita is less than $20, 000. As the years go by, GDP per capita keeps growing in many countries, and life expectancy keeps improving as well. Let's see this again. Personally, I think looking at these charts is very cool, so I would highly recommend reproducing this result by yourself if you haven't done so yet. Earlier, we ignore the outlier whose GDP per capita was over $60, 000, but you might ask, which country was that one? Let's find it out with data[data. gdpPerCapita > 60000] It turns out it was Kuwait. They had close to $100, 000 in GDP per capita from 1952 to 1972. Now, let's draw a scatter plot for each year with lifeExpectancy and the log go gdpPerCapita as well. We're going to make a scatter plot with plt. scatter, and the only difference in this line from the previous cell is the fact that we are using np. log10 of data_year. gdpPerCapita instead of the raw values. Just like before to deal with outliers and for more consistency, we're setting a consistent range for the X axis and Y axis with plt. xlim and plt. ylim. And just like before, we're going to export these plots as images with plt. savefig. We're going to set the filename to be something like log_1952 for each year with log_ + str of given_year. Let's examine what the exported plots look like. With these charts, we see the same pattern of continuous improvement that we saw earlier, this time with GDP per capita on the log scale instead of a linear scale. As we saw earlier, you can see that these charts show more of a linear progression between the GDP per capita on a log scale and life expectancy. In fact, the log of GDP per capita has a higher correlation with life expectancy than the raw value of GDP per capita every year. I would recommend checking it yourself. In the next video, I'm going to give you an example problem to practice what you have learned in this video.

Practice Problem 3 - Scatter Plots

Here's a problem for practicing what you've learned so far in this module. Examine the relationship between GDP and life expectancy in 2007. We're talking about GDP and not GDP per capita here. To find GDP, you can multiply population and gdpPerCapita in a country's data, because gdpPerCapita is, of course, GDP per person. With Python and Pandas, you can write data. gdpPerCapita * data. population to compute GDP, assuming that the data variable contains the country's data as a data frame. Make a scatter plot with GDP and life expectancy, and try using both raw GDP and the log of GDP, just like we did in the last video. As usual, if you want to practice on your own, pause the video right here and come back when you're done. And here's an

example solution. First, import Pandas, Pyplot, and Numpy, as well as the countries data. Let's find the data for 2007 with data And let's find GDP by multiplying gdpPerCapita with population. You can do this with data_2007. gdpPerCapita * data_2007. population. These are going to be fairly large numbers, so let's divide them by 10 to the power of nine, which is one billion, so we can find GDP in billions. Let's put this and GDP in billions. Make a scatter plot with this with plot. scatter(gdp_in_billions and data_2007. lifeExpectancy, and use five for the point size. And show it with plt. show. It looks like countries with higher GDP tend to have higher life expectancy as well, but the effect is not so clear. So, let's try using a log scale with plot. scatter(np. log10(gdp_in_billions), data_2007. lifeExpectancy). The positive correlation is a little bit more clear here. Let's compare this with the scatter plot we made in the last video with the log of gdpPerCapita. So here, instead of using log10 of GDP, I'm using log10 of gdpPerCapita. It seems like the correlation is more clear with GDP per capita than with GDP. Let's see if we can confirm this by comparing the correlations of life expectancy with the log of GDP as well as the log of GDP per capita. As you can see, the log of GDP per capita is more highly correlated with life expectancy than the log of total GDP for each country. This makes sense if you think about it, because if a country has a large population, each person might be poor, but the GDP of that country could be high. And it's probably more important that each person be well off for him or her to have a long, healthy life than his or her country be wealthy in total. You've reached the end of this module. And in the next module, we're going to cover bar charts.

Comparing Data with Bar Graphs

When to Use Bar Graphs

A bar graph, of course, is a convenient way to compare numeric values of several groups. Let's see an example of when to use a bar graph. Suppose you're working in the finance division of a company that provides software as a service. You're trying to better understand the company's finances. The company has several software as a service products, but you're not sure which ones are the most important to the company's bottom line. In a situation like this, a bar graph could be useful. Let's say you decided to use a bar graph to compare the revenue from the different products that your company offers. You might have products for email marketing, client management, survey creation, and so on. You can create a bar graph that shows the revenue from

each product like this. From this graph, you can see that most of your revenue comes from the email marketing product and the client management product. But you might say, what about the costs? You can show that in a bar graph as well. Once you have a graph of revenue and cost by product category, it might look like this. From this graph, you can see that the client management product is the most profitable one, more so than the email marketing product. You can see that the document editing product is actually losing money because the cost is higher than the revenue. If you only have this information say, in a table format, it would have been much harder to grasp the situation. Putting this kind of information in a bar graph, helps you understand it more intuitively. So to summarize, bar graphs help you compare numeric values, and you can use them to compare multiple numeric values of several groups as well, for example, revenue and cost. Putting data in a bar graph sometimes gives you a new insight that you might have missed otherwise. This is true in general for any other data visualization tool as well.

Creating Bar Graphs with Matplotlib

Let's now examine how to create bar graphs with Matplotlib. To see how to do it, we're going to compare the populations of the 10 most populous countries in 2007. As usual, we're going to use countries. csv. We're going to use the year column to select the data for 2007 and the population column to find the 10 most populous countries in that year. I'm now going to dive into a demo to show you how to sort a Pandas DataFrame with the sort values function and how to create bar graphs with Matplotlib's plt. bar function. After importing Pandas and Pyplot, as well as countries. csv, find the data for the 2007 with data square brackets, data. year equals 2007, and put it in a variable called data_2007. Let's make sure that we selected the right data with data_2007. head. Then, to select the 10 most populous countries, we're going to sort this data according to the population column. To do this, you can use the sort values function. This says, sort this DataFrame, Data 2007, according to the population column, and in a descending order, and not an ascending order, because we want the largest values to come to the top. Here you can see that the most populous countries came to the top, with China and India at the top. To select the top 10, you can just add. head(10), which gives us the first 10 row of this new DataFrame. Let's store this result in a variable called top10. To make a bar graph of populations of these countries, we'll need to first call range of 10, and store it in the variable called x. This is basically equivalent to writing x equals zero, one, two, up until nine. So, range of 10 is similar to a list with the elements 0 through 9. Then you can create a bar plot with plt. bar(x, top10. population). This says plot bars with their position specified with x or 0 through 9 and their height specified with top10. population. Show it with plt. show(). As you can see right now this graph doesn't have any

country labels. We can add them with plt. xticks(x, top10. country, rotation='vertical'). The first argument in xticks, in this case x, specifies the coordinates of the labels that you want to place and the second argument, top10. country in this case, specifies the actual strings that you want to show. Rotation='vertical' means that you want to show these labels vertically instead of horizontally. To make this graph easier to read let's divide the populations with 1 million, or 10 to the power of 6, so we can see the populations in millions instead. Finally, let's add a title and a y label to make this graph easier to read. Looking at this graph, it's now easy to compare the populations of the 10 most populous countries. For example, China and India each have much more population than any other country after that. In the next video, I'm going to introduce you to a problem for practicing what you've learned in this module so far.

Practice Problem 4 - Bar Graphs

Here's a problem for practicing what you've learned in this module so far. The problem is, compare the GDP of the 10 most populous countries in 2007. For this problem, remember that you can find the GDP of each country by multiplying the population column with the gdpPerCapita column. To do that you can write data. gdpPerCapita times data. population assuming that countries. csv has already been imported into the variable called data as a Pandas DataFrame. And then, you can use the subplot() function to show two bar plots on the same graph. One for the population and the other one for the GDP. If you don't remember how to do this you can review the histogram module, and pause the video right here if you want to try solving this problem on your own. Here's an example solution. After importing Pandas pyplot and countries. csv just like in the last video, select the data for 2007 with data square brackets data. year = 2007. Then sort this DataFrame according to the population with data_2007. sort_values ('population', ascending=False) and select the top 10 with. head(10). Let's put this in a variable called top10. After that, make a bar graph of GDP with plt. bar(x, top10. gdpPerCapita times top10. population divided by 10 to the power of 9. This says, plot bars with their heights being the GDP and with their locations being x, where x is range of 10. And we're dividing the GDP with 10 to the power of 9 here so we can see the GDP in billions of dollars. Then, add xtick labels with plt. xticks(x, top10. country, rotation='vertical'). This says of course add xtick labels with the country column being the strings that you want to show and with x representing the coordinates of those labels. Show this with plt. show. Now let's add a bar chart for population to this graph as well as a subplot. To do this we're going to use the plt. subplot function. First, call plt. subplot with the arguments two, one, one, which means our graph will contain two rows and one column of subplots and that we want to select the first one of those. Then, before we create the bar graph

for GDP call plt. subplot with the arguments two, one, two. This means, of course, the graph will contain two rows and one column of subplots and that we want to select the second one of those. Once you run this code you see that there's no graph in the first position. Let's add a graph there with plt. bar(x, top10. population divided by 10 to the power of 6) to show a bar graph of population in millions. Now, to remove the default xtick labels in the first subplot you can call plt. xticks with two empty lists. Then, let's clean up this graph by adding a title and a few legends. Now, using this graph you can easily compare the populations and the GDP of the 10 most populous countries in the world. For example, even though China and India have a much higher population than the US, the US still dwarfs them in terms of GDP. Now, this solution works, but there's an alternative solution to this problem. It's in the After folder in the file called Alternative Solution to Practice Problem 4. With this solution, we're showing the GDP and the populations side by side. This is just a different way of showing the same set of information. You might actually prefer this one, but I decided not to present this as a main solution because it's much harder to create this graph. But, if you're curious about how to create it feel free to play around with this file Alternative Solution to Practice Problem 4 in the after folder. That's it for this module. In the next module we're going to examine what to do when your data is too big for some of the data visualization techniques we've covered so far.

What to Do When Your Data Is Too Big

Introduction to Data Aggregation and Sampling

Suppose you designed and launched a new app recently. Let's say you're product manager of this app and you're in charge of making decisions about what kind of features should be on there. So you started analyzing when people use this app and what the demographics are like. If you have 1, 000 users, you might see a graph like this. This shows when users use your app between 9 a. m. and 1 a. m. and what their ages are. From this graph, you can see that many people use this app sometime between 1 p. m. and 9 p. m. and that younger people tend to use this app at a later time than older users. What if, instead of 1, 000 users, if you had 100, 000 users? What would this graph look like then? It might look like this. As you can see, it became just a big blob and you can no longer easily tell that the peak traffic happens sometime between 1 p. m. and 9 p. m. You can still see the correlation between time and user's age, but because you have more data here, you

actually get less information from this graph. So what should you do in a situation like this? The two main options here are data aggregation and sampling. Let me first introduce you to the concept of data aggregation. Data aggregation is, of course, a way to aggregate your data so that important information is easily seen. For example, to get an idea about the age range of users at different times, you might compute the mean and percentiles of users' age for each hour. The summarized view might look like this. You can also summarize the amount of traffic that you have for each hour or the number of users who use the app for each hour. If you do that, you might get a graph like this. This clearly shows that the peak traffic hours are between 1 p. m. and 9 p. m. The other primary option for when you have too much data is sampling, or sometimes it's called random sampling. The idea is you can just select a random subset of the original data to get an idea about the properties of the original data. When polling firms give out surveys about what people think about particular political candidates, they use a similar technique because it's impossible to survey everyone in the given country. So in this example, instead of trying to plot all 100, 000 points, you might randomly select, say, 1, 000 points to sample and plot. You can see that this graph is similar to what we had before, showing a thick cluster of traffic near the center of the graph, between 1 p. m. and 9 p. m. When doing a sampling analysis, I'd recommend sampling a few different times to make sure that result you're seeing doesn't depend on the particular samples that you selected. So when your data is too big to plot and make sense of easily, you can try aggregating it to find summary statistics such as mean and percentiles. And you can also try sampling a random subset of your data. In fact, the fact that you can sample a subset of your data and still get meaningful results suggests that perhaps you don't need to have collected all that data in the first place. So if you're in the position of collecting data in the first place, I'd be careful about how much data you really need. For example, if your website has, say, 100 million users, it might not be necessary to keep track of all of their behaviors on the website. It might be sufficient to sample, say, only 10 percent of the users for data analysis purposes.

A Data Aggregation Example with Python

In the next few videos, we're going to see examples of how to aggregate data and how to randomly sample data in Python. Let's first see how to aggregate data. For this, we're going to use a new data set, called obama. csv. It's in the Before folder of the demo files for this module. We have four columns in this data, as you can see. Year-month, survey organization, approve percent, and disapprove percent. It shows how President Obama's approval ratings changed over his four years in office. And this data comes from various survey organizations. For this example though, we're only going to use the year-month column and the approve percent column to see

how his approval rating changed over time. Let's now dive into a demo, in which we're going to create a scatterplot of approval ratings. And we're going to aggregate this data to show mean and median of approval ratings for each month. First, import pandas and pyplot as usual, and import obama. read. csv with pd. read. csv. When importing obama. csv, you need to use the parse dates argument with parse dates equals square brackets year-month. This was, pandas is going to interpret the year-month column as a series of timestamps instead of as regular strings. We'll need to have it as a series of timestamps so that we can use them to plot a time series more easily. Let's make sure that we got the right data with data. head. Then, plot Obama's approval ratings over time with pt. plot data. year_month, data. approve_percent, and string "o. " The third argument in the plt. plot function, lowercase o, means that we want to plot the points only as small circles, without any lines between them. Show this with plt. show. Right now, the points are too large, so to make them smaller, use the markersize argument in the plot function, let's set it to 2 here. You can also make the points transparent to get a better idea of where the points are clustered. We can do this with the alpha argument. Let's set it to 0. 3. With this, you can see that some points are clustered around the center of the band. Now, what if we wanted to somehow aggregate this data and summarize it? One way to do this is by showing the mean approval percentage for each month. For this, we're going to use the groupby function. You can just write "data. groupby('year_month'). mean. " This says, group the dataframe called "data" by the year_month column and find the mean for each month. As you can see, it computed the mean approval percentage and disapproval percentage for each month. Let's store this new dataframe in a variable called "data_mean. " We can plot this with "plt. plot(data_mean. index, data_mean. approve_percent). " Data_mean. index gives us the year_month column, which is treated as an index column in the data_mean dataframe, so with this we can plot Obama's approval ratings over time. Let's also add a string, read as the third argument. This will make the line red. Show this with plot. show. Then, we can show this line along with the individual data points by plotting the approve_percent data points, as we saw earlier. Just as we saw earlier, we're going to write plt. plot(data. year_month, data. approve_percent), and so on. What if we wanted to show median as well as mean? We can find the median in a similar way. Data. groupby(year_month) groups the data by the year_month column, and we can find the median for each month with ". median. " Store this in a new variable called "data_median. " Let's plot this with plt. plot, data_median. index, data_median. approve_percent, green. To make it clear which line represents mean, and which line represents median, add a legend with plt. legend. What if we wanted to find percentiles as well, for example, 25th and 75th percentiles? We can do that with data. groupby('year_month'). quantile(0. 25). This finds the 25th percentile of approval ratings for each month. Put this in data_25, and do the same for the 75th percentile. Let's show the median, the

75th percentile, the 25th percentile, as well as the individual data points together by called plt. plot several times. Also add a legend to show which line represents what. Show this whole graph with plt. show. This was an example of how to aggregate data with Python. I would recommend recreating this example yourself for practice, and in the next video, I'm going to show you how to randomly sample data using Python.

A Random Sampling Example with Python

Now, let's see an example of how to use random sampling with Python. We're going to use a new dataset for this example called, obama_too_big. csv. This is similar to obama. csv that we saw earlier, but I added a bunch of rows that are just fake, generated data, to simulate what happens when your data is too big. This file is also in the Before folder of the exercise files for this module. In this demo, we're going to create a scatter plot of approval ratings, just like we saw in the last video. Then, I'm going to show you how to randomly sample some rows from our data. After importing pandas and pyplot, import obama_too_big. csv with pd. read_csv. Set the parse_dates argument to square brackets, year_month, to parse the year_month column as timestamps. Put this in the variable called "data_big, " and do the same with obama. csv, and put the result in "data. " Let's make sure that we got the right data with "data. head, " and "data_big. head. " You can see that in data_big, some generated data has been added. Let's now compare the size of data and data_big. You can find the size of the dataframe in the variable data, with data. shape. This says data has 1, 530 rows, and 4 columns. What about data_big? It has 65, 499 rows, and 4 columns. So data_big is much larger than data. Let's plot approval ratings from data_big against time, just like we did in the last video. Remember that the string "o" says, we want to only show points without any lines in between them. Let's set the marker size to 2, and alpha, or the transparency parameter to 0. 3. Show this with plt. show. As you can see, because there are too many points here, the graph becomes just a giant blob, and it's hard to know how the ratings are really distributed. In a situation like this, sampling comes in handy. To sample some rows from the data_big dataframe, you can write data_big. sample(frac=0. 1). Frac stands for fraction, and this says sample 10 percent of the data from data_big. Store this in a variable called "sampled. " Instead of frac equals 0. 1, you can also write n=10, 000, to specify the number of samples instead. Let's plot this sample data just like before, with plt. plot, and plt. show. Once we plot the sampled data, it's much easier to get a sense of the distribution of the data. In this graph, you can see that the approval ratings are clustered around the center of the band, and this is the power of sampling. You've reached the end of this video and this module. I would recommend reproducing

this result on your own for practice, and in the next module, we're going to use some of the tools we've learned so far to solve some practical problems.

Solving Real-world Problems with Visualization

What's the Richest Country in the World?

Welcome to the last module of this course, Solving Real-world Problems with Visualization. In this module, we're going to combine some of the skills we've learned so far throughout this course to solve the kinds of problems you might encounter in real-life. As the first example, let's say we want to answer this question. What's the richest country in the world on a per-person basis? For simplicity, let's just compare different countries' GDP per capita to try to answer this question. We're going to go back to using our countries. csv data. Just remember that GDP per capita here is shown in what's equivalent to 2005 U. S. dollar. Pause the video right here if you'd like to try answering this question yourself. I would also encourage you to post your solution as a comment to this course, so that other people can learn from your example. Here's an example of how I would go about answering this question. I'm going to use some of the concept we've learned throughout this course. For example, aggregating values with the groupby function, sorting values, making line charts, and using subplots. After importing pandas, pyplot, and countries. csv, I'm going to find the mean GDP per capita for each country. I'm going to do this with data. groupby, country. mean where the variable data contains countries. csv as a pandas data frame. This line says group this data frame with a country column and find the mean values of all the other columns for each country. As you can see, it gives us the mean year, which doesn't really mean anything, as well as the mean life expectancy, population, and GDP per capita for all the available years combined. We'll need to grab the GDP per capita column from this new data frame with. gdpPerCapita. So these values are the average GDP per capita of all the available years for each country. Let's store this in a new variable, called mean GDP per capita. Using these values, we can sort the countries according to the mean GDP per capita. We can do this with mean gdp per capita. sort values ascending equals false so that we'll see the countries with the highest GDP per capita first. Let's grab the top five with. head. It seems like Kuwait stands out of these five countries with over $65, 000 dollars of average GDP per capita. Let's store this piece of data in the variable called top5 for now, and let's look into Kuwait in more detail. So, is Kuwait

really the richest country in the world on a per-person basis? To find out, let's first isolate the data for Kuwait with data square brackets data. country equals Kuwait, store this in a new variable called Kuwait. Show it with kuwait. head. And let's look at how the GDP per capita in Kuwait has changed over time with plt. plot, kuwait. year, kuwait. gdpPerCapita. Add a title, and show the plot with plt. show. As you can see, between the 1950s and the early 70s, Kuwait's GDP per capita was very high, and it was over $80, 000. However, after that, it's GDP per capita dropped very quickly. What happened? Since GDP per capita is GDP per person, let's plot Kuwait's GDP per capita, GDP, and population on the same graph using this subplot function. As you can see, we're making three subplots here on top of one another. Right now, the subplots titles are not shown correctly. To fix this, you need to call the plt. tight_layout function to fix the layout. From this graph, you can see that in the late 70s, Kuwait's GDP dropped significantly. It turns out, they experienced an economic crisis at that time. Their GDP eventually picked up, but at the same time, their population kept growing, partially due to the high level of immigration there. How much faster did Kuwait's population grow relative to the GDP? It's hard to tell from the graph, so let's try comparing their relative growths in a single chart. We're going to show the population growth with plt. plot kuwait. year kuwait. population divided by kuwait. population. iloc square bracket zero, which selects the population figure in the first available year, times 100. This way, the first year's population is set to 100 as the basis for comparison. Do the same for GDP and GDP per capita, where kuwait_gdp is simply kuwait. population times kuwait. gdpPerCapita. Add a title and a legend, and show this graph with plt. show. As you can see, up until the early 1970s, the population and the GDP grew roughly at the same rate, but after that, the population kept growing but the GDP never caught up with it. Now, the top five countries with the highest mean GDP per capita were as follows. One of the was the US. Let's compare how the GDP per capita in the US has changed over time with that of Kuwait. Select data for the US with data square brackets data. country equals United States and store it in a variable called US. And plot the GDP per capita growth in the US and Kuwait on the same chart with plt. plot us. year us. gdpPerCapita, and the same for Kuwait. As you can see, the GDP per capita was much higher in Kuwait until the 1980s, but after that, it's been much closer to that of the United States. So, back to the question that I asked earlier, is Kuwait really the richest country in the world on a per-person basis? It's not so clear now. That's it for this video, but if you tried answering this question yourself, I would encourage you to post your solution as a comment to this course so that other people can learn from your example. In the next video, I'm going to give you a few more problems to practice with.

Solve Your Own Problems

I'd recommend practicing what you've learned so far by solving one or two more problems. You can come up with your own question for either your own data or countries. csv. Or you can try answering one of the following questions using countries. csv. What's the fastest growing country in the world in terms of GDP? Which country resembles the US the most, with regards to the information we're given in this data? How is the population related to GDP per capita? Again, I would recommend that you share your results in the comments section of this course, so that other can learn from your example.

Conclusion and the Journey from Here

In summary, what you've learned so far is, a strong basis for you to start analyzing real world data using data visualization techniques, and how to make various types of plots using python, pandas, and McLaven. If you're interested in learning more about data visualization, I would first recommend that you start using what you learned in this course to start answering questions with some real world data. If you don't have a strong background in statistics, I'd also recommend learning some statistics fundamentals as well. Taking a few introductory statistics courses online, or at a university, should be a good start. If you have any questions regarding the analysis you are doing, please feel free to comment below or contact me directly at csdojo. io. Good luck, and thank you so much!

Course author

YK Sugi

YK Sugi the creator of CS Dojo, a popular programming education YouTube channel. He is also a software developer and data scientist with experience at various software companies, including Google...

Course info

Level	Intermediate
Rating	★★★★★ (170)
My rating	★★★★★

Duration	1h 28m
Released	27 Jul 2017

Share course