Doing Data Science with Python

by Abhishek Kumar

Start Course

Bookmark Add to Channel Download Course

Course Overview

Course Overview

Hi everyone. My name is Abhishek Kumar with Pluralsight, and welcome to my course on Doing Data Science with Python. Data science is one of the hottest fields these days, and no wonder data scientist has been termed as the sexiest job of the century, because with the help of data science you can unravel meaningful insights, and generate data-drive evidences that can benefit organizations in a significant way, and provide them a competitive edge. So if you also want to make a jump-start in this fascinating field of data science, or if you are already in this field and want to learn the standardized way of tackling end-to-end data science project cycle using Python, then this course is for you. In this course, we will dive into various phases of a data science project such as data extraction, data processing and visualization, building, evaluating, and fine-tuning predictive models, and finally, exposing your predictive models as APIs for real-time integration. The only prerequisite to this course is that you should be familiar with the basics of programming with Python. This course will not only help you to build the foundations of data science, but to also help you learn to implement the concepts through lots and lots of demos. by the end of this course, you'll be in a position of knowledge and skills necessary to kick start your data science journey in the Python world. So please join me in this very exciting course on Doing Data Science with Python, at Pluralsight.

Course Introduction

Course Introduction

Hi, this is Abhishek Kumar, and welcome to this course on doing data science with Python. Data science is one of the most exciting fields these days, and one of the key reasons behind this excitement is the massive amount of data that is getting generated across the world. Looking at some numbers, currently the whole world is generating roughly 2. 5 quintillion bytes, that is 2. 3 trillion gigabytes per day, and with this rate, the world will generate approximately 40 zettabytes, that is 43 trillion gigabytes of data by the year 2020. Not only this, the data is getting generated from multiple sources such as transactional data from various enterprises, or process data from various industries. Data is also coming from various social networks in the form of tweets, posts, videos or images. It is also getting generated through internet connected devices also known as Internet of Things. But what would be the data, it is no use if we cannot extract some value out of it. And there comes data science into the picture. So what is data science? Well, broadly speaking, data science provides a set of fundamental principles that guide the extraction of knowledge of data. And the companies around the world are using these principles to gain edge over their competitors by extracting valuable data-driven insights and creating value-added data products. And this makes data scientists as one of the most desired assets for the companies, as they can help companies to generate valuable insight and patterns from data. No wonder Harvard business Review has termed the data scientist as the sexiest job of the 21st century. So if you're also interested in the field of data science and want to learn how to extract knowledge and patterns from data, then this course is for you. In this course, we will learn to cover the path from raw data to meaningful value and insights through different stages of data science project cycle. Along with the basics of data science project cycle and its components, you will also learn to tackle different stages using Python and Python-based tools and packages. And the best way to learn something is by getting your hands dirty. Therefore, we will take various hands-on exercises so that you can follow along with me and learn to use these Python tools to tackle real-world data science problems. Along with several small exercises, there will be an overarching case study that will encompass the rest of the modules, where we will take a raw dataset and then we will work on it step by step. This case study will help you to connect different dots, and you can use this case study as a common framework for your future data science projects. You will also learn some of the best practices for tackling data science projects and problems. This can be very fruitful for you in the long run. I will also provide some tips and tricks along the way that can be

very handy at times. The case study we will take in this course is related to Titanic disaster. We will use this interesting dataset to go through each stage of the data science cycle one by one. I'm sure you must have heard of the Titanic disaster that happened way back in 1912 when the Titanic ship sank after colliding with an iceberg. In this disaster, almost 1, 500 people died out of roughly 2, 200 passengers including crew members. We picked up this case study due to a coupe of reasons. First of all, as most of us are aware of this incident, therefore there is no specific domain knowledge required to understand this case study. Another reason is that this case study is both simple and complex at the same time. Simple because it has smaller number of variables and you can easily understand the problem. Also, you can get good results with relatively simple models. But at the same time, through this simple case study we will unveil the complex nature of complete data science project cycle. Not only this, this case study is also a Kaggle competition challenge. Those who are not aware of Kaggle, it is a very famous platform for machine-learning competitions. So towards the end of the course, we will also see how you can participate in Kaggle competitions. This can be very effective as you put your knowledge and understanding to a real-world challenge. You can also apply the learning of this course to tackle other exciting data science problems too. So if you're also excited about data science just like me, then buckle up and hop on to this roller coaster ride to take a dive into this very exciting field.

Target Audience

This course is targeted for both data science aspirants, as well as data science professionals. On one hand, this course will help data science aspirants to know everything they need to know to make an elegant entry in the data science world. On the other hand, data scientists and the data professionals who may be using a different set of tools or framework, will learn the usage of Python-based tools and packages to accomplish their data science related tasks. So this leads to a very important question. That is, what are the skills required to follow this course?

Course Prerequisites

Here are the course prerequisites. As this course is built on the Python language and framework, so the basic understanding of Python is required. You do not have to be a Python guru to follow this course as we will be using only some basic Python constructs in this course. But if you are not comfortable with Python at all, then you can take any Python fundamental course. A few great Python courses are already available in the Pluralsight library, so you can take any of them. Also, this course will assume that you know some basic high school mathematics. Don't worry, we

will not be talking about advanced mathematical concepts in this course. And if you have some background knowledge of statistics and machine learning, then you will be rocking during this course. But even if you do not have such background, then don't be scared. I'll explain all the relative concepts to you so that you can easily follow along in this course. And in case you face any trouble while understanding any concepts of code, then you can always post your queries in the discussion forum, and I'll try to reply back as soon as possible. So now we have set up our expectations, let's dive into the data science world and have an overview of the data science project cycle and its components.

Data Science Project Cycle Overview

So as mentioned previously, data science helps us to make a journey from data to valuable insights, but it has several components along the way. Let's examine these components one by one from the bird's eye view perspective. The first piece is the extract phase. We need to get the data first before we do anything with it. And data extraction, or data acquisition, is the first step towards our data science journey. Data may have to be acquired from one or multiple sources. Once we have extracted the data, the next building block is to organize or pre-process the data, because rarely you will get data in the required format. You may have to perform various data wrangling tasks to clean it and then put it into the right shape. Once we have organized the data, our next step is to analyze and create model. As you can see, we have used a cycle or loop kind of thing here, because it is an iterative step. In the real-world settings you may may have to try different models and then analyze them and evaluate them over and over again until it reaches the desired level. Once you have your model and analysis ready, then the next step is to present the results. Results could be in different formats. It could be a report or presentation that you may want to show to your stakeholders. It could be a blog or website that you want to put it on the web. It could even be an application that has to be deployed in a production environment. But many times it can happen that the key insights you have drawn through all these steps are not good enough, and you may want to further build upon it. Then you may have to repeat this whole cycle again with some new and additional data with different processing or different analysis or modeling techniques, or different approaches of presenting the results. So this outer loop is used to represent the iterative nature of the whole data science cycle. This shows that it is a continuous ongoing activity where you incrementally build upon your previous works. Now the next and very obvious question is that why we have picked up Python for tackling this data science project cycle in this course? What benefits does Python provide for data science work? Let's address this key question in the next clip.

Why Python for Data Science?

So what are the reasons behind Python's rapidly increasing popularity for data science projects? Let's see a few of the key reasons here. First of all, Python as a language itself is fairly easy to learn, as its syntax is similar to other common languages. Also, Python code are also easy to read and are quite intuitive in nature. Another key reason for Python popularity is the availability of various open source free tools and libraries for various data science- related activities. These tools are very helpful in making a head start with data science projects. You will find the complete ecosystem in Python's stack for data science-related activities right from the extraction phase to the presentation phase of the data science project cycle that we have seen in the previous clip. We will learn about several such tools and packages in depth during this course. Also, in the real world we will face issues and challenges during our work and may need help. Python has a very strong and active community, and you can easily get help if you run across any problem or issue. Python is also quite fast and scalable with respect to some other languages and frameworks that are typically used in the data science world, such as MATLAB or Stata or R. I would not say that Python is the most scalable and the fastest language in the world, but there is a very active push in the area of scalability by the developers, researchers, and the community to make it even more faster. Last, but not the least, what makes Python great is its ability to get into the production environments, especially for companies that are already having Python-based application stack. So your data science projects don't have to remain on your personal laptops. It can get into the production environment and can be leveraged to create meaningful data-driven products or applications. So now we have set up our expectations for this course, let's look into the course outline in the next clip to see how we have structured this course for you.

Course Outline

Here is the course outline. We will first set up our working environment in the next module. You will learn about various Python distributions that are typically used by data scientists. You will also learn to set up Jupyter notebook on your machine. Along the way, we will also explore some of the basic features and tips and tricks of Jupyter notebook. Further, we will talk about some of the best practices such as following a data science project template, and using a versioning system. Then module 3 will be focused towards data extraction phase, and we will learn how we can extract data from different types of sources, such as databases, publicly available APIs, and websites, through web scraping. We will also extract the Titanic dataset in this module. That will be used in the consecutive modules for exploration, and building predictive models. From the tools perspective, we will look at a bunch of Python libraries, including several database

connectors to connect to different databases and request library to work with APIs. We will also use another very popular Python library, BeautifulSoup, for our web scraping work. Then we have three modules dedicated on the organize step, where we will cover various aspects of exploring and processing data. In part 1, we will focus on basic exploratory data analysis techniques using Python tools such as NumPy and pandas. Then, in the second part we will focus on advanced analysis techniques, and along the way we will learn several useful Python NumPy and pandas functions. In the third part of the exploring and processing data series, we will talk about several techniques for data munging, where we will identify different issues in the data and resolve the issues using standard techniques. You will also be taken to the interesting world of feature engineering and we'll also learn to create compelling visualizations. Form the tool's perspective, we will extend our NumPy and pandas journey, and we'll also learn another really useful Python library, that is Matplotlib. After data exploration and processing, we will get into the exciting world of predicting models. That we will cover in two parts. In the first part, we will work on the foundation of machine learning and we'll build and evaluate basic machine learning models. We will also submit our predications to Kaggle platform as a part of the presentation phase, and along the way we will leverage the Python Scikit-Learn library to accomplish our goals. Then, in the second part we will learn some important concepts related to model tuning that will help us to fine-tune our model performance. We will also learn ways to take the machine learning model to production setup so that your machine learning model can be a part of full-blown data solution. In this direction, we will cover the model persistence to persist the trained machine-learning model to the disk, and then we'll also develop a machine-learning API on top of it, so that it can be integrated with any system. In this module, other than the Python Scikit-Learn library, we will also cover a couple of useful Python libraries, so we will use Pickle for our model persistence and Flask to develop the machine-learning API. So as you can see, we have lots of fascinating stuff to cover in this course, so let's get into the exciting world of data science.

Summary

So in this module, we have made some basic foundation and understood why data science is so important in the modern era where the data is getting generated in abundance, and we must use it to extract meaningful knowledge and value. Then we turn our attention towards the typical data science project cycle and its components, such as data extraction, data processing, data analysis, modeling, and presentation. We also looked at some of the key benefits offered by Python framework to handle real-world data science problems and projects. Then, we set our expectations and looked at the contents of different modules that we will cover in this course. So

having built a solid foundation, let's set about our working environment in the next module that we will leverage in the rest of the course. So see you in the next module.

Setting up Working Environment

Introduction

Hi, this is Abhishek Kumar, and welcome to the second module of the course on doing data science with Python. For tackling any data science project, you need to have a working environment, and Python provides a very rich set of tools to create such an environment very quickly, so in this module you will learn to set up a working environment that we will use for this course. But you can use similar working environments for your future data science projects as well.

Overview

So in this module, we will first look at some commonly used Python distributions that are typically used by data scientists. Then we will pick one of them and then install it on our local machine. We will then set up Jupyter Notebook, which has become an essential component in every data scientist's toolkit. We will then go through the basics of Jupyter notebooks, and we'll learn some tips and tricks. Then towards the end of this module we will talk about some of the best practices for handling any data science project, such as following a standardized data science project template and using a versioning system.

Python Distributions for Data Science

In this clip, I'll talk about Python distributions for data science. Well, you already know Python, and may already have it installed on your local machine along with your favorite Python packages, so why do you need a Python distribution for data science? Well, typically for handling data science related tasks in Python you have two options. The first option is that you install base Python, and then install other required packages one by one. However, sometimes installing some packages could be tedious, and you may encounter some issues related to packages versions or

dependencies. A much better way, especially for beginners, is to use specialized Python distributions that comes with pre-installed and optimized Python packages. This is very useful, especially when you are starting your data science journey, because you can set up your environment very quickly, and you can simply start right away without worrying too much about versions and dependencies. A couple of very famous and commonly used Python distributions that are typically used by data scientists are Anaconda and Enthought Canopy. These Python distributions come with pre-installed packages that are useful for data science related activities, but you can always install other Python packages that are not available in these Python distributions. You can use a free version of either of these distributions to get started, but you can also look into their commercial products if you are interested in dedicated customer support or enhanced features. In this course, we will be using the free version of Anaconda distribution, but you are free to pick either of these two Python distributions. However, one major question that you will encounter when downloading any of these Python distributions is that whether you should work on Python 2 version or Python 3 version, so in the next clip we will briefly list out pros and cons of using Python 2 or Python 3 versions.

Python 3.x vs. Python 2.x

So why do we need to worry about Python versions? Well, Python 3 has some breaking changes from Python 2, therefore, you will have to pick one path before moving ahead, so which one should you pick? Well, Python 3 is cleaner and much faster than its predecessor, and is definitely the future. However, some packages have still not moved to Python 3, so Python 2 offers stable third-party packages, and because it's been there in the arena for a long time it has better community support also. Not only this, some special features of Python 3 have backward compatibility with Python 2, so you can use Python 2 and still get those features. In this course, we'll be using Python 2 version for our work, but you can pick up Python 3 version also without any issue. So now you have some solid idea about Python distributions and its versions. So in the next clip we will install the Anaconda Python distribution.

Demo: Installing Ananconda Distribution

In this module, we will install the Anaconda distribution on a local Windows machine, but steps are more or less similar for other operating systems as well. To install the Anaconda distribution on your local machine, you can download the installer from anaconda. com. So you can go to the Download section. Here you can find installers for different operating systems such as Windows,

Mac, and Linux. In this clip, we'll be installing the Windows version, but you can follow the simple instructions to install Anaconda distribution based on your own operating system. So on the Windows tab, you will find installers for both Python 2 and Python 3. We will be installing the Python 2 version in this course. So let's click on the download button to download the installer, Save File. So it may take a while to download the installer. I have already downloaded the installer, so let's use this installer to install the anaconda distribution. Simply double-click on the installer, click Next to start the installation process, agree with the license agreement. We can install only for the current user or for all users, so I'll set it up for all users. Let's give the privilege through user account control. Here you can set the installation location. I will let it as the default location, so let's click Next. Here, on the Advanced Options screen we want to add Anaconda to the path so that we can access the Anaconda distribution from any terminal in our command prompt, so let's check this box. If you will not add anaconda to the system PATH environment variable, then you will have to use Anaconda terminal to access Anaconda. However, in this course we will be using our own terminal, and that's why we have added Anaconda to the system PATH environment variable. Now let's click Install to install the Anaconda distribution. It may take a while to complete the installation process, so I'll skip to the end. So now Anaconda is completely installed. Let's click Next. You can also read about Anaconda Cloud and Anaconda Support. Let's uncheck them for now. Click Finish. Now let's verify if we have successfully installed the Anaconda distribution. We can use any terminal. I am using the Git Bash terminal here, but you can pick any terminal of your choice. So on the terminal let's see the Python version using the python version command. So here you can see the default Python on your machine is the Anaconda distribution. The version may change depending upon your installer. Well, Anaconda distribution, as I told you, comes with several pre-installed Python packages, so let's check out the installed Python packages using the pip list command. Here we can see we have several Python packages that come with Anaconda distribution. If you are using Anaconda distribution, then you can also use the conda list command. This will also list out different Python packages in your Anaconda distribution. So now we have set up our machine, in the next clip we will start exploring the Jupyter notebooks, which will be our working environment for the rest of the course.

Jupyter Notebook

Jupyter Notebook has become a go-to tool and environment for most of the data scientists these days. So what is Jupyter Notebook, and what makes it so special? Jupyter Notebook, commonly known as IPython Notebook, has become an integral part of data science projects due to its

ability to combine code blocks along with human-friendly text that can be formatted using markdowns. You can also view the images and videos right in your notebook. This makes Jupyter Notebook as a very interactive document, and you can work on it using your favorite web browsers, and we will see that in a demo in just a moment. Well, the Jupyter Notebook not only supports Python dernels, means not only you can run Python code in it, you can also run codes in other languages as well, such as R or Julia or Scala, right in the single notebook, so you can get all of this in a single development environment. That's really cool, isn't it? At least I find it really awesome. Not only this, if you want to share the notebook, then others can also view your notebook using nbviewer right in the browser, and they don't have to have the whole Python setup at their own end. In fact, GitHub has also started supporting the rendering of Jupyter notebooks in the browser. But if you want to share the document in some other format, then also you can export or convert your Jupyter notebook in other formats such as PDF or LaTeX. So, as you can see, the Jupyter Notebook is a very versatile tool. So now let's learn to set it up on the local machine in the next clip.

Demo: Setting up Jupyter Notebook on Local Machine

In this module, we will set up the Jupyter Notebook on the local machine. To run the Jupyter Notebook on your local machine we can use our terminal or command prompt if you are in the Windows machine. In our working directory, let's create a folder named as module2. Let's navigate to this folder. So inside this folder we can use the following command, jupyter notebook. You can also specify the port number using the port flag. If you do not specify the port number, the notebook will try to use the default port number of 8888. To know more about the notebook server options, you can use the help flag. So, as you can see, there are several options here. One common option that is no-browser option. You can use this option, especially on systems that don't have any graphical user interface. So this will start the notebook server without opening a web browser. This is a very common scenario where we want to host the Jupyter notebooks on cloud, so that you can run the notebook server without any browser and then use your own local machine to access the notebooks. Anyways, let's use the default jupyter notebook to launch the notebook server on your local machine. So here we have the Jupyter Notebook in our default web browser. Now you're all set to go. In the next clip, we will start exploring the Jupyter Notebook and we'll learn some basics.

Demo: Jupyter Notebook - Basics

In this module, you will learn about the basics of Jupyter notebooks. So in order to create a new notebook, you can click on this New drop-down menu, and select the Python kernel. If you have other kernels installed, then the other kernels will also be listed out here. Let's pick the Python 2 as of now. So here is the new notebook. Let's give it a name. To set the name, you can click on this Untitled area. Now you can type the name of your notebook. Let's set it as My First Notebook. Well, a Jupyter notebook is made up of cells. By default, you will get one cell, and you can create additional ones as well. These cells can contain different types of items. By default, whenever you write in these cells they will be treated as code, which in our case is a Python code. If you would have selected different kernels such as R or any other language, then the text will be treated as code in that particular language. Now let's write the hello world Python code here. Now to execute the cell, you can go to Cell and click Run, or you can use the keyboard shortcut CTRL+Enter. We will use this keyboard shortcut to execute the cell. And here you can see the output just below the cell. This is really great, because now you can write your code and view the results side by side. Another thing you might have noticed about this square bracket after the letters In. So if you see the square brackets, this means the cell contains some code, and each time you will execute the cell, the square bracket will show the execution number of the code cells. Another important thing is that when you are using the Ctrl+Enter then it will execute the cell, and the focus will remain on that particular cell only. However, if you use another very helpful keyboard shortcut, that is Shift+Enter, then not only it will execute the cell, it will also create a cell just below it. Let's try that. So as you can see, a new cell is created, so let's use this new cell to enter some markdowns. You might be familiar with markdowns. Markdowns are used to create formatted text on web browsers. To convert the cell type to markdown, you can go to Cell, go to Cell Type, and then select Markdown. However, we will learn another keyboard shortcut for doing this. The keyboard shortcut is Esc+M, m for markdown. Keyboard shortcuts are very helpful, and once you are accustomed to these you will find it really easy and fast. So here we have used the keyboard shortcut Esc+M to convert the cell to a markdown cell. Also, as soon as the cell is converted to a markdown cell, the square bracket from the left of the cell has been removed. Now let's see some basic markdowns that we will use throughout this course. So just like that, you want to add six tags in HTML. You can use hash symbols. So single hash is similar to h1 tag. Let's write something using single hash. So here you can see the size is quite large, and font weight is bold. Typically, you want to use single hash for the main title of the notebook. If you want smaller font, then you can use more hashes. Let's add more hashes. If I add more hash symbols, the font size is decreased. So this is just like h1 h2, and h3 kind of tags. In order to execute the cell, again, you can use the Ctrl+Etner or Shift+Enter if you want to create a new cell after the execution. So now you are comfortable with code and markdown cells, let's do some more hands-on. Here I

have added some bunch of cells, and we will go through each of these cells to learn more about Jupyter notebooks. So let's learn a very simple code. Just type 2 + 2, and you can get the output by executing the cell. Then we have created a variable, x, now let's print out the variable content in the next cell using the Python print command. And here we have the output tree. So you can think one notebook as your working environment, and any global variable that you create in any cell can be used anywhere in other cells. Moving on to the next cell, here we have created a Python Function definition. Let's execute the cell. So the function is now created, now we can call this function and pass some parameter values. And here we have the output. Now let's learn some more keyboard shortcuts. Suppose you have a long code block, then for such scenarios you may also want to add line numbers for your reference. So in order to get the line numbers, you can use the keyboard shortcut Esc+L, and here we have the line numbers. It can be very handy, especially if you encounter some error, and the error shows the issue related to a specific line number. If you want to know more keyboard shortcuts, you can go to help and click keyboard Shortcuts, and here you will find all keyboard shortcuts that you can use. I'll again recommend that you use some of the keyboard shortcuts because it will make your life easier. Moving on to another very useful tip, which I find really handy, and it is the simple exclamation mark. This will help you to run any shell commands in the notebook itself. Let's use it to check the version of the Python installed on your machine. Let's execute the cell, and voila. We can get the cell command output right in our notebook. Isn't it great? So now you are comfortable with the Jupyter Notebook and its environment. In the next clip we'll talk about another very fascinating feature of Jupyter notebooks that are magic functions. But before moving to the next clip, let's save this notebook. Here you can notice the notebook gets autosaved after some time interval, but if you want to explicitly save it, then you can go to File, and then click on Save and Checkpoint. The keyboard shortcut is Command+S on Mac, and Ctrl+S on Windows machine. Now let's move to the magic function demo in the next clip.

Demo: Jupyter Notebook - Magic Functions

In this module, we will talk about the magic functions that you can use in the Jupyter Notebook. So far, we have learned about some basic Jupyter Notebook tips. Now let's learn some magic tricks in this clip. Hmm, I'm in Jupyter Notebook Magic Functions. Jupyter provides these magic functions to extend the development environment to a whole new level, and that's why they're called magic functions. Let's learn some common magic functions that will make your development life very easy. I have already created a notebook for you, and I'll go through each cell one by one. First thing to remember about magic functions that they start with the

percentage or the modulo sign. Also, remember that if you have a single-line expression with a magic function, then you use the single percentage or modulo symbol, however, if you have multi-line expression you need to use double modulo symbols. Now we will start with a magic function that is very common in any data science project. That is matplotlib magic function. Matplotlib is a very commonly use data visualization package in Python, and we will make extensive use of it in subsequent modules. So if you use matplotlib inline, then it will allow to show the visualizations created using the matplotlib package inside the Jupyter Notebook itself. So let's execute this cell. I'm using the Shift+Enter keyboard shortcut here. That not only executes the cell, but also shifts the focus to the next cell. Now let's create a very basic plot to verify if we are getting the visualization in the notebook or not. Don't worry, if you can't understand this code, we will learn about it later. I just want to show you the power of Jupyter Notebook. So let's execute this cell, and there you have a visualization inside the Jupyter Notebook itself. This is a really great thing about Jupyter notebooks, as it helps to have codes, comments, and and visualizations all in a single place. Now let's see some more magic. Time is another very helpful magic function. You can use it to measure execution times. Let's use this magic function to find out the execution time to get a range of 10000 integers. And here we have the wall time. If you will run this cell multiple times, you may get different wall times. But this magic function can be very useful if you want to track how much time your data processing or machine learning algorithms are taking. Another variant is the timeit magic function. Here we are using the double modulo symbols because we have a multi-line expression here. Timeit will perform the time evaluation multiple times, and will return back the average time taken. This will help you to get the overall picture about the time consumption of any function execution. Another useful magic function that we will use in this course is the writefile magic function. So if you want to create a file and write something in it, then you can do that using this magic function. Here we are writing to a file, text. txt. If this file is not available in the current working directory, then a new file will be created; however, if a file with the same name already exists in the current working directory, then it will be overwritten. So after the magic function line, you can write whatever content you want to write to the file. So let's execute this cell. So the file is written. This can be very handy if you want to write some sample test data to a file, or want to write some Python code to a file. Now, to test whether the file is created or not, we can use another magic function, that is the ls magic function. So let's execute the next cell, and here you can see they have a file named as test. txt in the current folder. Well, a Jupyter notebook can also contain different types of items such as HTML or LaTeX. So let's quickly look at the examples of these. So in order to embed the HTML in the Jupyter notebook itself, you can use the HTML magic function. Well, here we are embedding only the image in the notebook, but actually you can embed YouTube videos as well inside the same Jupyter notebook.

Isn't it really cool? Well, if you are working on any algorithm and want to add equations in the notebook, then you can use the latex magic function, and you will have the equation in your notebook itself. Now let's look at another magic function, that is load_ext. This magic function can be used to load other extensions. Let's use it to plug the SQL code right in this notebook. But before that, you need to install the extension for SQL. We can use the pip install to install other packages. To execute the shell command, we can use the trick that we learned in the previous clip, in fact, this is also a magic function. However, you might wonder that why we don't have modulo symbol here. Well, there is a modulo kind of version as well to run the shell command. So here, with the help of double modulo symbols, you can use the magic function for running multiple shell commands. For now, let's use it to install the ipython-sql package. So now the package is installed, now you can load the extension using the load_ext magic function. So here, after the load_ext, we are writing the module name of the extension, which in this case is sql. so let's execute this cell. So now SQL is available in the notebook. Let's use it to write some SQL code. Well you can use this extension to connect to different types of databases. Here we are connecting to a local empty sqlite database, but you can connect to any of the database that is supported by SQLAlchemy. Now we can write some SQL commands to create a table and insert a few rows. Again, recall, if you want to use multi-line expressions, you have to use a double modulo operator. So let's execute this cell now. So now your table has been created with a couple of rows. Now let's create this table using the select operation, and we have the output of the SQL code right in the notebook. Here in this example, we have loaded a SQL extension, but there are several other extensions available that you can include in your notebook. As a data scientist, you may have to use different languages for different kinds of activities, and Jupyter notebook allows you to have everything in a single place. So in this module, we learned about several magic functions, but there are other magic functions as well, and we will learn a few of them in upcoming modules. But if you want to see the available magic functions, you can, again, use a magic function, lsmagic. This will list out all of the available magic functions. So now you have mastered the Jupyter notebooks, we will jump into the data science project template in the next clip. That will establish a common framework to work on any data science project.

Data Science Project Template

In this clip, we'll talk about data science project template. You might be familiar with project templates if you're coming from any software developing background. Well, if you look into a typical data science project, it has several components. So you can have raw data, processed data, you will have some notebooks, some models, a few visualizations, some reports, it can have

documentations, and references as well. So as you can see, even for a small project there can be so many pieces, and for moderate scale and large-scale data science project, things can quickly go out of hand. Therefore, it is very important that we organize our data science project in a manner that other team members can also follow. So you can think a data science project template as structuring of different pieces or artifacts of your data science project in a consistent fashion. So what are the benefits of having a template? A project template will lead to a more consistent project structure, and it will standardize the way people in your team are expected to work. It also makes collaboration easy as different team members can work in their specified _____, and others will know where to look if they require any specific piece of information. And if there is any new team member joining your project team, then he or she can understand the project structure and dive in right away without going through the extensive documentation. Reusability is another very important aspect in our data science project. You should be able to produce a similar set of results or analysis from the raw data in a reproducible manner. This will also ensure better code quality, and you will be more confident about your work and results. Following a structured approach will ensure the reusability and code quality right from the beginning. So the key takeaway here is that if you are following any standardized approach, your life will be easy. So the next, and very obvious question is what is an appropriate project structure or template in a data science world. Well, there is no definite answer to this question. Your team might already be having a project template, and you can follow that. But whatever project template you pick, my advice is to use it from the word go. Even if you are doing for your individual projects, then also following a standardized template will help you in the long run. In this course, we will follow up with a template that is specifically designed for data science projects, and it is cookiecutter data science project template from Driven Data. It is available on GitHub, so you can check out this link as well to know more. I personally found it as one of the most appropriate data science project templates so far, but you are free to follow any other project template if you wish. In the next clip, we will set up cookiecutter project template, and then we will explore its components.

Demo: Setting up Cookiecutter Data Science Project Template

In this clip, we will set up the cookiecutter data science project template. To set up any project using the cookiecutter template, you'll first have to install it. So if you are using pip, then you can go to your terminal and type pip install cookiecutter. But if you have installed the Anaconda distribution, then you can also use the conda install. So let's use it. Let's proceed. Once you have installed the package, you can use it to create a new project. So here we are in our working

directory. Let's clear the screen first. Now we can type cookiecutter followed by the cookiecutter GitHub repo link. Press Enter. This will clone the project structure from GitHub to your local machine. It will then ask you about some basic information about your project, such as the project name, so let's give it a name. And as we'll be working on the Titanic disaster dataset, let's give it a name titanic. You can add the repo_name as well if you want to link this to a GitHub. You can add the author_name, some description. You can select the license type. You can also provide s3_bucket details if you want to sync data between s3 and your local project. Let's skip it for now. You can also provide your AWS credentials. Then you can select your python_interpreter. As we'll be working with Python 2, let's select the first option. So now you are done. Now let's do an ls to the current directory. So now we have a folder corresponding to the project name, that is titanic. Let's get into this directory. Now let's see the structure of this folder using the tree command. Here I have set the level to be 3. As you can see, this template has several components. We will not be using all of the components, but it's good to know about different placeholders and how a typical data science project is structured. So from the top, you have a LICENSE file, then you have a Makefile. These are the typical makefiles that are used to run different scripts. Then inside the data folder we have different subfolders. External will contain the data from third-party sources, interim will contain the intermediate data that has been transformed, processed folder will contain the final data that will be used for modeling activities, and raw folder will contain the original data. And you don't want to modify the raw data in place. So this is an immutable thing. Then you have docs for the documentation, models folder will contain different models or model summaries, notebooks folder will contain all of the Jupyter notebooks. Then the references folder will contain manuals for any other material that you want to include in your project. Reports folder can contain the final analysis in different formats, such as HTML or PDF or LaTeX. Figures subfolder can contain different graphics or figures that you may want to include in your reports. Requirements. txt file can be used to reproduce the analysis environment so that the others can install the packages that you have used to generate the results. Src folder will contain the source code for your project. It also contains an init file so that it can be converted into a Python module. Then all of the scripts to download and generate the data can reside in the data subfolder. Features subfolder can contain scripts to convert the raw data into features that will be used for modeling. Model subfolder can contain scripts to train the model and use the model for prediction. Then you have the visualizations folder that can contain scripts to create visualizations that are part of your project. The cookiecutter template also contains files for setting up test environment and tox. ini file if you want to use tox for automated testing. Although in this course we will not be using all portions of this template, as we'll be mostly concentrating on notebooks, but such a standardized approach can greatly help you in the long run. Another important aspect

of a data science project is version control, especially if you are working in a collaborative fashion. So in the next clip, we will talk about the importance of versioning in data science projects.

Versioning for Data Science Projects

Versioning is a very important aspect for any kind of team or collaborative work. The same is applied to data science projects as well. Versioning system can provide a better collaboration among the team by providing a common repository on which different team members can work together without messing things up. You can also track your changes in a better fashion, and in case you have made some major errors, you can always go back to your previous commits. All the benefits we have discussed apply even if it is an individual project. Take my words, you will have a sense of relief if you are using some kind of versioning system right from the beginning, and I would highly recommend that you start using the versioning system for all of your data science projects. So which versioning system should you use for your data science project? Well, you can pick any one of your favorite. However, the most famous versioning system that is out there is Git. It is simple to use, and is very powerful, and I think you should definitely spend some time learning it. It will pay off in the long run. And if you want to share your work with your team members or publicly, then you can also take benefits off various service provides such as GitHub or Bitbucket. GitHub has become a very common choice for data scientists these days to showcase their work and to contribute to the open source projects. But I'll reiterate once again, whatever versioning system you pick, try to have it as a part of your project right from the inception phase. In the next clip, we will use Git to add versioning to our titanic project.

Demo: Add Project to Git

In this clip, we will add our project to a versioning system, that is Git. In this course, we'll be using the Git versioning system, but you are free to use any versioning system of your choice. If you want to know more about Git, then Pluralsight has some great resources on git. However, we will not be using advanced Git features in this course, so the commands used in this clip will be more than sufficient for the purpose of this course. So let's set up the Git on the titanic project folder. Let's first make sure that we are on the titanic project root folder. So we are in the titanic project folder. Now, in order to initialize a Git repository, we can use the git init command. This will initialize an empty Git repository. Next, in order to add files to the Git versioning system you can use the git add command. We have also specified dot. That means take all the files into this folder and add to the versioning system, press Enter. Now let's commit our changes using the git

commit command. This will tell Git that we are done with our changes, and now you can create a checkpoint. Also, you should always provide some commit message so that later on you can reference it, what changes you have done. And because this is an initial commit, we are adding a message 'initial commit' using the -m flag. Press Enter. Now the changes have been committed to our local Git repository. Now, in order to check the log, you can use the git log command. You can also specify another flag. One line, that will only give you the git commit message along with the commit id. And here we can see our get commit message. So now our project has a versioning system in place, and we are all set to work on this titanic project in the upcoming modules.

Summary

So in this module, we mainly focused on setting up the working environment. We learned about different Python distributions and its benefits. We looked at different options such as Anaconda and Enthought Canopy. We also discussed the pros and cons of using Python 2 or Python 3. We then installed Anaconda distribution on our local machine, and then set up the Jupyter Notebook. We learned some basics of Jupyter notebooks, and how it can be used to have code, comments, and visualizations all at a single place. We then talked about some best practices for a data science project, such as following a standardized project template and using a versioning system. We also set up a cookiecutter project template for our project, and then we added it to Git version control system. So in this module, we created a common ground on which we will work in subsequent modules. Well, for any data science project, the first step is to acquire data, so in the next module we will talk about extracting data from different types of sources, So, see you in the next module.

Extracting Data

Introduction

Hi, this is Abhishek Kumar, and welcome to the third module of the course on doing data science with Python. Well, the journey of any data science project starts with gathering or extracting data, so this module will be focused towards the data extraction phase. However, the data can come from different types of sources. It can be available to files in different formats that you can download manually or programmatically. Sometimes, data may also have to be scraped from website or web pages, but it can be available to API calls. Other data may already be available

directly in your company's database that you can hook into. But sometimes you may have to collect data manually using forms or surveys. But if the situation demands, you may have to extract data directly through some device using available ports. So as you can see, depending upon your problem, you may have to acquire data from different types of sources. Even though the types of data sources can be a huge list, we will look at some most common types of data acquisition tasks in this module that you can typically face in your data science journey.

Overview

From conceptual point of view, in this module you will learn to extract data from some of the very common data sources such as databases and APIs. You will also learn a very useful technique that is web scraping that you can use to scrape data directly from the website by understanding the structure of the underlying HTML document. We will then extract the titanic dataset for our titanic data science project that we will be using in subsequent modules. And towards the end of the module, I will also provide some links and resources to get datasets for your practice. You can use these datasets for your future work as well. From the tools perspective, you will be learning two very commonly used Python libraries. One is Requests and the other is BeautifulSoup. Mastering these two libraries will surely help you in the long run. Other than these two libraries, you will also learn to use some of the Python libraries to interact with different types of databases, such as SQLite, MySQL and SQL Server. So now we have set up our expectations, let's jump into the data extraction from the databases in the next clip.

Extracting Data from Databases

In this clip, you will learn about Python packages that can be used to extract data from various databases. In the market, there are so many databases available, so it's impossible to cover all in one course, so we'll focus on three very popular databases in this course. We will first take the SQLite database that is a self-contained and cross platform SQL database engine. Then we will see how we can extract data from the open source MySQL database, which is an enterprise level database. We will also cover the SQL Server database, which is a proprietary database from Microsoft. But before we jump into the actual demo, I want to tell you a general outline that we will follow to connect to any database and extract data from it. This will be applicable for most of the Python packages that we will be using in the next clip. So first we will import a package that will be used to extract the data from the respective database, and if that package is not available, then you will have to install it first. After importing the package, we will make a connection to the

database using the connection string. The format of the connection string may vary from database to database. Then we will create the cursor that will be used to read data from the database. Then we will execute the query, and then we can fetch the query results and iterate over it. And finally, the connection will be closed. So now you have understood the general code outline, let's see it in action in the next clip.

Demo: Extracting Data from Databases

In this clip, we will learn to extract data stored in different types of relational databases using Python. Here, I have already created a Jupyter notebook for you, and I will go through each cell one by one. I have also added comments that can be helpful to you if you want to refer to it at a later stage. So we will first see how you can extract data from a SQLite database. So you need to first import the library that you can use to interact with the SQLite database. If you have installed Anaconda distribution, then it comes with a sqlite3 package that you can use. So, we are first importing the library, let's execute the cell, then to make a connection to a SQLite database you can use the connect method. In the connect method you can pass the database name. Here the sqlite3 will try to find the database in a local folder, and if it is not available there, then it will create a new database in the local folder. And if you want to close the connection, then you can use the close method on the connection object. So let's see if we are able to connect to the database. Execute the cell, and it is successful. Now let's see how we can create a SQLite database table. But sadly, we do not have any table, we have just created the database. So, let's create the table first, and insert a bunch of rows in the table so that we can later retrieve them. So steps are very straightforward to accomplish these goals. First we create a connection, then we create a cursor, then you execute a SQL statement using the cursor object. Here we are using the SQL CREATE TABLE statement to create the table. Here in this example we are creating a table named as classroom that contains student_id, student name, gender, and marks of them in three subjects. And after the execute statement, we are also using the commit function to commit all the changes. And finally, we are closing the connection. So let's execute the cell. So now we have a database with a table. Now let's fill up this table with 3 rows so that we can extract them later. So in the next cell, we are creating a list of tuples, where each tuple contains values for each row of the table. Next we are using our standard approach creating a creation first, then creating a cursor object. Then we are iterating over all the elements of the list using a for loop, and creating the insert statements for each row. Then once the insert statement is ready, we are executing that statement. And then after the for loop we are committing all the changes, and finally we are closing the connection. So let's execute this cell. So now all the rows are inserted in the database

table. So now we have a database table with some values in it. So we can extract some data from that. So let's see how we can do that. Well, again, the process is quite straightforward. Create the connection, create cursor, create a query string, and then execute the query. Here, we are using a very simple query to select all rows from the classroom table. Next, in order to fetch all rows from the cursor object, we are using the fetchall command. And once we have the results with us, we are printing the results one row at a time with the help of the for loop. And finally, we are closing our connection to the database. So let's execute the cell to see if we are able to get rows from the database. And here we have our results. So as you can see, it is very easy to extract data from a SQLite database. If you already have some database, then all you have to do is to write a query statement and then use it to fetch the results. So now you have learned how to get data from a SQLite database, let's see how you can use the exact same approach to stack data from other relational databases. And we'll start with the MySQL database. To work on the MySQL database, you can use the pymysql package. Well, by default this package will not be available in the Anaconda distribution, but you can install it using the pip installer, or using the conda install if you have Anaconda distribution. Here we are using the conda install command. So we can run the shell command, conda install pymysql, as shown here. We have also used two switch options, -y to accept the installation, and -q to do the installation without spitting out too many intermediate outputs. So let's run it. So now we have installed the package, so let's import this library. Now let's connect to the MySQL database. So in order to demonstrate the extraction part from a MySQL database, I have created a MySQL database instance on Amazon AWS. Cloud services like Amazon AWS or Microsoft Azure makes it really easy to set up these databases. And you can set a full-fledged database just using a few clicks. So here is the endpoint for the MySQL database instance, and we'll be using this endpoint to connect to this database. So coming back to the Jupyter notebook, here you can see we have used the endpoint as the host. We have also added the username and the password, and the database name that we created in the MySQL database instance is classroomDB. So we have put all this information into a dictionary object, and we have given it a name cnx. So we can use the information contained in this dictionary to connect to the database. And here is the line to connect to the database. I will not go through each line here, because it is almost the same code as before, only the way we are connecting to the database has been changed. So this block here contains the code to connect to a MySQL database. Let's test the connection. So we have successfully connected to the database. So let's follow the standard process that we had followed for the sqlite case. First we are creating the table, then we are inserting a few rows, and finally we are extracting data from the MySQL database. Similarly for the Microsoft SQL Server, you can use another Python package that is the pymssql. Again, we are installing it using the conda install, so let's install it first. After installing the package, let's

import it. Again, in order to demonstrate the connection to a SQL Server database, we have created a SQL Server database instance in Amazon AWS, and here's the endpoint. And we have placed the endpoint here in the host field. This time, however, instead of creating a new database, we're using the default tempDB database of the SQL Server instance. Let's see if we are able to connect to the database. So connection is successful. So let's follow the standard approach. Creating the table first, then inserting a bunch of rows, and finally extracting them from the SQL Server database. So as you can see, this is pretty straightforward to extract the data. So depending upon your requirement, you can write your own query. So in terms of the databases, if you have some other database then you can look for some Python packages that are already available to connect to those kind of databases. So now you have understood how to extract data from traditional databases, let's see another very common source of data that is the API or application programming interface.

Extracting Data Through APIs

APIs, or application programming interfaces are one of the most convenient ways to expose the data to the outside world. So we will turn our attention towards how you can use Python to get data from the APIs. So you can think of APIs as endpoints that clients can hook into to make requests without worrying about the underlying details of how to process the request. so, in a nutshell, the client makes a request, and the server returns a response. And sometimes the client also sends some additional information such as query parameters or custom headers to get specific results back from the server. APIs have grown over the years, and you might have seen its usage all over the place, and one very common type of API is the REST API. REST APIs have become the de facto standard for companies to expose their data to the outside world due to its scalability and usability. You might be having some previous experience with REST APIs, but don't worry if you have not worked with the REST APIs before. For the purpose of this course, it is sufficient to know that the REST APIs work on the HTTP protocol, and you can use certain HTTP verbs to get or post the data. So on a broad level, the client makes HTTP requests to the server, and the server can respond with an HTTP response. So if you want to simply fetch some data from the server, then you can use the GET HTTP verb. On the other hand, if you want to send some data to the server POST verb will be used. But as we'll only be retrieving data in this course, we will mostly focus on the GET method. In the Python world, you can use several libraries to make HTTP requests. In the next clip, we will see how we can use one of the most popular library, Requests, to greatly simplify this whole process. But before we get into the actual demo, let's see the typical code outline for making a get request using the Requests library. So first you import

the package, then you can use the get command to make the get request. And once you receive a response back from the server, you can access different properties of the response object such as its status code, the headers, and the actual response text. You can also look at other attributes of the response, such as its encoding. Now let's see all this in action.

Demo: Extracting Data Through APIs

In this clip, we will learn to extract data from API using the Python Requests package. To demonstrate the use of the Requests library to fetch the data from an API, we are going to use the U. S. Government open data platform. You can visit the data. gov to get more information. This is a great platform to find open data for research purposes. And you can browse through your areas of interest, such as agriculture or climate or education. I've been downloading some data from the Education sector, so let's click on Education. Then click on Data to see the available datasets. You can also use filters on the left to narrow your search. And if you will scroll a little bit, then you can find the different formats for available resources. As we will be using the APIs, let's click on the API. So let's take this first dataset, that is the College Scorecard dataset, let's click on this, and then let's click on the College Scorecard API. Here you will find some brief information about this API and the dataset. Let's click on this URL for the documentation part. So here is the documentation, and you can go through it to know more about this dataset. On the right-hand side, you can see the GET API link. We also have your sample API calls here. For example, here is an API to get all the data for the Boston College. So let's use this URL only; however, to use this API you will need one more piece of information. You will need an API key. Well, you can get an API key by visiting the api. data. gov link. Here you can click on this Get an API Key link, and then you can sign up by filling your name and email address. You will get your API key on your registered email id. So once you have the API key with you, you are all set to go. So here we are in our Jupyter notebook, I have created a new Jupyter notebook for you. So in order to use the Requests library, you need to first import it, so let's import this library. Next, we are setting up the URL. Here's the URL that we have already seen. The only thing I have changed here is that I have appended the API key at the end. So you can use the ampersand sign to add extra parameter in the URL. So let's execute it first. So once the URL is set, you can use the get command on the Requests library to make the get request. So let's execute this cell. So once we have received the result, we can inspect the response object. Let's use the status_code attribute to check the status code of the response. And here we have got the status code of 200 that represents a successful completion. You can also inspect other attributes as well. Let's inspect the headers. Here you can find the different information stored in the headers. If you want to get the response in the plain

text, then you can use the text property, and here is the result. It's a long one, let's just scroll. So here is the dataset for you in the text format, but if you want it in just one format, you can use the json, and here we have the result in the json format. So, as you can see, the Requests library makes the whole API thing quite a breeze, but many times you won't find a direct API to access the data you need. You may have to do some manual things, and may have to scrape the required data from the website itself. So we'll talk about the web scraping in the next clip.

Extracting Data Using Web Scraping

Web scraping is the process of extracting required information or data from the web pages by understanding its document structure. It is a very common task that you may have to perform if the data is not directly available to you through regular means such as databases or APIs. almost all of the web scraping techniques work by understanding the structure of the web page that is normally referred to as the document object model, or DOM, in short. So if you will look at any typical HTML document, you will find tags and attributes, and they are arranged in a tree-like structure. On the right-hand side of the slide, you can see the typical HTML structure. The first line is simply a declaration that this document is an HTML document. Then we have several tags that are also known as the elements. The root element is the HTML element, and then we have the head element that typically contains the metadata information and the links to style sheets and JavaScript files. The visible part of the HTML document resides in the body part. Inside the body, you can have different tags. Tags can have attributes as well. For example, here we have the class attribute associated here with the div tag. Likewise, you can have several tags in your HTML document, and the information that you are looking for can be inside one, or more than one tags, and all you have to do is to scrape the information from those particular tags using the document object model. Well, here we have a very well-structured HTML document, but sometimes in the real world it can be quite messy, and you may have to invest more time to understand the structure. Also, just a word of caution. Before you scrape the data from the website, just make sure you have the right permission to do so. You don't want to put yourself in some kind of legal issues, so it always recommended that you talk to the website administrator first before you perform the scraping, or at least make sure that you will not get into any kind of legal issue. So now let's see how you can perform web scraping using Python.

Demo: Web Scraping Using Requests and BeautifulSoup

In this clip, you will learn to perform web scraping with the help of two very popular Python libraries, Requests and BeautifulSoup. We have already seen how to use Requests library in the previous clip where we used it to make the API calls; however, the same Requests library can be used to fetch any web page as well. So here we are in our new Jupyter notebook. So first we are importing the libraries, so we are importing the Requests library, and for the BeautifulSoup, it is available in the bs4 package, and you will already have it installed if you have installed Anaconda distribution. So let's import these two libraries. Next, we want to scrape some data from a website, so instead of going into the actual web page, I have added an html_string here in the Jupyter notebook itself so that we don't have to bother about any legal stuff. Here is the typical HTML document that you will find across the web. So in this example, I have added some detail about this code in an HTML format. Well, let's see how this HTML document will look like. Well, you can do it in the Jupyter notebook itself, but before that let's execute this cell. Well, to display the HTML document, you can use the display function available in the IPython core display library. So we are first converting the html_string into the html_object using the HTML function, and then we are passing the HTML document to the display function. Let's execute this cell. And here we have the rendered HTML document right in the Jupyter notebook. Isn't it cool? So in this web page you have the title of the course, course author name, some brief description, and a table containing the module names and the duration. Suppose you want to extract the module name and the duration of this page for some kind of analysis. So how can we proceed? Well, let's see in a step-by-step process. So you can first pass the html_string to BeautifulSoup, and BeautifulSoup will convert the html_string into a structured object that you can query later. We are also storing the dessert in a variable named as ps, so let's execute this cell. Here you'll see a few warnings. That is basically for the HTML parser. We can skip it as of now. Next, let's print the ps object itself. So as you can see, we have the complete HTML document structure with us; however, this time you can create this document structure. So suppose you want to extract only the body portion. Then what you can do is to pass the body as the value of the name attribute in the find function on the soup object ps, as shown here. Find function can be used to return the first matching child, and because we have only one body tag in the structure, we will get our required information. Let's execute this cell. Now let's print the content of the body to confirm the result. So as you can see, we have only the body portion with us. So now suppose you want to extract the title of the code that is inside this h1 tag. So essentially, we want to extract the text content inside the h1 tag. So you can use the text property. So here we are first selecting the first matching h1 tag, and then we are extracting the text content. So let's execute this cell, and here we have the title of our course. Coming back to the structure, let's apply the find function on the p elements. So we have two p elements here. Let's apply on the p element. So we have got our first matching element.

But, as you already seen that we have two p tags, and if you want to get all the p tags in one go, then you can use the findAll function instead of the find function. Let's execute this cell. So here we have got the list of elements in the result, and because you have a list, you can also iterate through the results, as shown here in the next set. Here we are using a simple for loop to iterate through all the p elements. Other than the tag names, you can also use other tag attributes as well to make the selection. Here you can see we have two p tags, so let's extract the first one with the id attribute value of author. So you can pass the attributes that you want to use for selection as a dictionary to the attrs parameter in the find function. In the first example, we are passing the id attribute value as author. Let's execute this cell. And here we have our p element. Let's use a similar approach to find a second p element where the id attribute value is description. And here is the p element. So now we have made our foundation. Now let's use it to scrape the module details from the HTML document. But before that, let's look at the document structure once again. So we have a table here inside the body with the id of module that contains the desired information. Then inside the table we have a bunch of rows. The first row contains the header information, and we want to skip that, so we need to look through the rest of the rows, and inside each row the first element is the module title, and the second element is the duration in minutes. So now let's see how we can set up the whole web scraping process. So first we are extracting the body, and then inside the body we are extracting the table element that has the id of module. And then inside the module table we are iterating through all rows of the table by using the findAll method, and tr as the value of the name parameter. And also you can see we are skipping the first row, because the first row has the header information. Inside the for loop, we are extracting the text content of the first column that is our module title, and the text content of the second column that is our model duration. And we are also converting the duration string to integer. Finally, we are printing both the module title and the duration. But you could have used these values as you like. I just wanted to showcase the extraction of particular portion from the web page. Let's execute this cell. And here we have our module title and the duration. So, as you can see, we have scraped the web page for the desired information and extracted only those bits. Well, in this example we had a very simple case, but in the real world you may encounter more complicated document structure. However, you can use similar techniques to perform the web scarping as per your need. Well, so far we have seen some of the most common ways to extract the data. Now we will put all of our learning to use, and we'll extract the titanic disaster dataset from the Kaggle website. That will be used in the rest of the course. So let's cover the titanic dataset extraction next.

Demo: Getting Titanic Dataset Using Requests : Part 1 - Initial Preparation

In this clip, you will learn to use Python to extract titanic disaster data from Kaggle website. So in order to download the dataset, you need to first visit the Kaggle website. If you already have an account, then you can log in. Otherwise, you can also sign up easily. I already have my account set up, so I will log in to my Kaggle account. Next, you can go to the titanic disaster data challenge web page. So you can search for titanic in the search box and click on the competition link. Then you can go to Data. So here on the left-hand side you can see there are several files; however, we need to bother about only two particular files. One is the train file, which is the raw training dataset, and the second one is the test data, on which we need predict the survival for each passenger. One more thing you need to do, if you are downloading the dataset for the first time, you may encounter a dialog box for terms and agreements. But if you are able to download a file by taking on these dataset links, then you are good to go. Let's cancel it for now. So basically we need to download the train and test dataset, and we can do it manually by simply clicking on the links. But we will do it in an automated fashion, and why would you need that? Let me tell you one possible case where an automated script will help you. Well, in the titanic disaster challenge, the file size is pretty small, only a few KBs. But if you are working on any problem where you have a large dataset, and you want to use cloud infrastructure to run your algorithms, then you would first have to download the file on your local machine, and then you will have to upload the raw dataset in the cluster before you can do anything with it. And if the dataset is too bulky, then it may take a while depending upon your internet speed at your end. And moreover, if you have a very large dataset that is beyond the capacity of your own local machine, then it also makes sense to directly download the dataset on the cloud machine. And on the cloud also, you may be working on terminal-based machine without any graphical user interface, so it's always good to have some automated script to get everything in place from the command line. Not only this, as a data scientist you should always strive for automation. So now let's see how you can come up with an automated script that will automatically download the data for you, and I promise you that you will learn some new tricks along the way that can help you in the long run. Now, enough of the background, let's see how you can download the dataset in an automated fashion using Python.

Demo: Getting Titanic Dataset Using Requests : Part 2 - Downloading Data

But before we get into the actual code, remember, you can download the dataset only after logging into the Kaggle system. So you need your username and password. So let me share one good practice to tackle such cases where you need to use your credentials in the notebook. Well, why do you bother about this? Well, you may be sharing your notebooks on GitHub or any other public repositories, and you don't want to push your credentials in the public. So now let's see another best practice where you can use your credentials without exposing it. So we are in the titanic project directory. Let's check the present working directory. Now let's open this folder using your favorite editor. I'm using the Atom editor, but you can use any other. And here is our project. As you can see, the cookiecutter project template has also provided one more file, that is. env file, let's click on this. So in this. env file, we will store our credentials. So we'll be creating two variables here. So here I have added two variables, KAGGLE_USERNAME and KAGGLE_PASSWORD, where we will store our credentials. You can put your own username and password here. Let's save this file. Well, don't worry, this file will not be added to the version control. Why? Because if you go to the getignore file, so inside the gitignore file, you can place all those files which you don't want to get dragged. And here you can see there is an entry for. env file as well, so you can rest assured that this file will not be pushed anywhere. So now we are all set. Let's launch the Jupyter notebook in the titanic folder, navigate to notebooks folder, and I have already created a notebook here for you for the data extraction process, so let's open it. So in order to use the credential information stored in the. env file, we will be using a utility Python package, python-dotenv. With the help of this library, you can use the. env file very easily. Here, first we are installing the python_dotenv package using the pip install. Once the package is installed, you can use it. So in the next cell, we are importing a couple of functions, load_dotenv and find_dotenv from the dotenv package. So let's execute this cell. Then in the next cell we are using the find_dotenv to find the dotenv file. This function will find the dotenv file by automatically walking up the directories until it is found, and once the path through the dotenv file is found, you can load the entries in the dotenv file to the environment variables using the load_dotenv function. So let's execute this cell. And once the information is stored in the environment variables, you can leverage the os library to get the value. Here we are extracting the KAGGLE_USERNAME from the environment variables. We are also printing the username. Let's execute this cell. And here we have the KAGGLE_USERNAME that we mentioned in the dotenv file. Well, in the similar lines, you can also get the stored password. So now you have learned how to store and use the information without sharing it, now let's see how you can download the training and the test data from the Kaggle website. We are again using the Requests library here. So essentially what we are doing here is that first we are sending the credentials to log into the Kaggle website, and once we are logged in, then we are downloading the files. We will be using

the session function available in the Requests library to create a login session. We are also creating a dictionary named as payload that contains the username and password. And the action value is login, because we want to login to the system. And the url is the actual url of the training file. You can get this link from the Kaggle website itself. So in the Kaggle data page, you can right-click on the Download button and copy this link location. So coming back to the Jupyter, next we are creating a session, and we are first logging in using the login url and the payload information. And once we have logged in, then we are downloading the file contained in the url variable. We are also printing the response here just to make sure this process is working. So let's first load the libraries, and execute the cell. And here we have our titanic dataset displayed in a Jupyter notebook. Well, so far we have only printed the training dataset that we downloaded from the Kaggle website, now let's see how you can actually save the response as a csv file. In the next cell, we are creating a method extracting this code data where we are passing the url of the file to be downloaded, and the file path where it will be stored. However, this time we have set the stream property to True so that we get the response as a stream, and then we can write each block of 1024 bytes one by one in the destination file. So let's execute this cell. So now our function is ready. Next, we are using the extract_data method to get both the train and the test data. So this time we have two urls, one for the train data and another for the test data. We are also storing the downloaded file in the raw subfolder inside the data folder. Again, as a good practice, we are not using hard-coded paths, rather, we are first getting the parent directory, and then using the os. path. join to join the path of raw folder to the parent folder. This ensures that our code can work on both Windows and Linux-based machines without any issue. Once the raw folder path is set, we are creating the file paths for training and test file, again, using the path join technique. And finally, we are using the extract_data method a couple of times to download the train and test file. So let's execute this cell. So now both of the files should be in the raw folder. Let's check out using the ls command. Well, now we have successfully downloaded both the desired files, and we can stop here itself. But, again, I want to push you a little further towards building a full reproducible script that you can execute from the command line without opening the Jupyter notebook and executing cell by cell, so that on a new machine, all you have to do is to execute the script and you will have the raw data. So in the next clip, we'll put everything that we have done in the Jupyter notebook into a single script.

Demo: Creating Reproducible Script for Getting Titanic Data

In this clip, you will learn to create a reproducible script that you can use to download the titanic disaster dataset directly from the Kaggle website. So here we are in the same Jupyter notebook

from the previous clip. So before we create the script, let's decide where we want to put that. And we will be storing all of our scripts in the src folder. And as because this script will be used to get raw data, we have created our data subfolder inside the src folder. So first we are creating a path to the script file, again, using the os. path. join trick, and then we are writing to this file using the Jupyter notebook writefile magic function. So now let's see the contents of this script. Well, we have already seen all parts of this script, so I'll quickly go through it. So first we are importing our libraries. So as you can see here, we are importing one more library that we have not seen before, that is the logging library. This will be used to log intermediate messages so that users can know about the intermediate steps that have been completed. Logging is also a very good practice, and you should definitely use it in your work. And the logging library is very useful, and you can use it to log the intermediate messages. And moreover, it is very easy to set up, and we'll see how you can do it in a short while. So coming back to the script, after the import lines we have our payload dictionary. Then we have the extract_data method that will download the file from the Kaggle website, and it will write to the file. Then we have created a new method named as main. Here, inside this method, first we are getting our logger instance using the get logger function. So once you have the instance, you can use the logger to log any informative message using the info command. Next, we have put our urls for the training and the test file, and then we are creating the file paths, and finally using the extract data method to download the files. The important part in this function is the project_dir argument. That corresponds to the main project directory. We are getting the main project directory, again, with the help of os. path. join trick. So here inside this main block we are getting the path of the script file first. So here you can see we are using the os. path. join to get the project directory, and we are passing three parameters here. First, we are getting the path of the script file itself, then we are appending the parent directory a couple of times. This will help to move two levels up. We are doing this because we know that the script file is two levels down from the root folder of titanic. So once the project root directory is set, we can use all paths related to the project root directory. We are passing the project root directory path to the main function. We are also doing a couple of things in between. First, we are setting up the logger, along with the log format, and second we are using the. env trick to load the credentials stored in the. env file. We have already seen how this works. So coming back to the log, here you can see we have first set the log format so we have included the time, the name of the function, the level, as well as the message, and then we are using the log format to set up the logging. So now we have seen the different parts of this script, let's execute this cell to create the actual script. But before we do that, let's execute the previous cell. So now the path is set, so now we updated the script inside the src folder. So in the next cell, we are executing the script in the Jupyter notebook itself. Great, so finally we have successfully downloaded our dataset using

the script file. So now we have the raw data, and we will be using this raw data in upcoming modules, and we'll build the predictive model to predict the survival from the titanic disaster. But you can use the learnings of this course to other datasets as well. So before we close this module, let's see some of the resources from where you can find some interesting datasets that you can use as a playground and hone your skills.

Public Datasets

Well, in this course we will be working on the titanic disaster dataset, and you will learn to perform various data science interactively, such as data exploration, data munging, data visualization, and predictive modeling. But you can use the learnings of this course to tackle other data science problems as well, and I would highly recommend that you do your hands dirty on different types of datasets and problems, because each dataset will throw a new type of challenge to you, and it will help you to learn new things. But from where you can get publicly-available datasets? Well, let me share some of the links and websites that you can go through to get very interesting datasets that you can work upon. We have already seen one such data source, that is the U. S. government open data initiative. If you want to get data for any other country, then you can also check if similar resources are already available. And if you are looking for large datasets, then you can also look into the Amazon AWS public dataset collection. It has datasets from different domains. In most of California, _____ also hosts some very popular datasets, especially from the machine learning perspective. You can find many research papers and articles on the internet using these datasets. I would also like to point you to a very useful GitHub repository. This repository is a compilation of some amazing datasets that are open and available for public use. You can go to your respective domain of interest and then find relevant datasets to work upon. Don't worry, you can find all the links that I have discussed here in this this clip in the presentation file. But, again, I want to re-emphasize that the more you will practice on different types of problems, the more you will learn, and I hope that you have a great data science journey ahead.

Committing Changes to Git

However, before we end this module, let's commit all of our work to git versioning system as good practice. So here I have opened the git bash on the titanic root level folder. Let's check the git log first. I'm also using the oneline flag here. So if you recall from the previous module, we made an initial commit when we had created this project. Now let's use the get status command

to check what files we have changed or added since then. Now let's add these two files. We've also got some warning that we can safely ignore. So let's commit these changes. So now let's check our git log again, and here we have all of our commits listed out. If you have synced up your code to any cloud repository as well, such as GitHub, then you can push your changes to GitHub also, but for now, we are all set for the next step in our adventurous data science journey.

Summary

So in this module, you learned about several Python tools and techniques to extract data from a variety of sources. We started with some common and very popular databases, then we shifted gears and learned to use the Requests library to get the data through APIs. We even extracted a dataset from the U. S. Government open data platform by using the git method on the REST API. Then we learned some tricks to scrape data from the website directly, by understanding and exploiting its hierarchal structure. We also used the BeautifulSoup library to work on a very simple dry example. Then finally, we extracted the titanic disaster dataset from the Kaggle website. We even built a reproducible script to automate everything. Then we went through a few resources for public datasets that you can use to work on different types of problems. So overall the agenda of this module was to get you accustomed with the different types of data sources and data extraction processes. Now, from the next module onwards, we will focus on only one dataset, that is the titanic disaster dataset, and we will start with a data exploration and the processing phase. So, see you in the next module.

Exploring and Processing Data - Part 1

Introduction

Hi, this is Abhishek Kumar, and welcome to the fourth module of the course on Doing Data Science with Python. This module is the first part in a three-part series of modules on exploring and processing data. In the previous module, you learned how to extract raw data from a variety of sources, including the titanic dataset that we extracted from the Kaggle website. Now, we will take our raw data and start exploring it to get some valuable insights. And moreover, rarely you can use raw data directly further down the line in the data science lifecycle to build predictive

models. Therefore, you need to organize or process the raw data in some form or another. Well, you might be wondering why we have three modules dedicated on organize step, and why don't we go directly on the fancy predictive modeling and machine learning steps. In order to answer this question, let's look at some numbers to understand where real-world data scientists spend their overall time in a typical data science project. So here is a simple pie chart that demonstrates time spent by data scientists on different data science activities. And straightaway if you look at the biggest portion of this pie chart, it is dedicated to data cleaning and processing. That is almost 60%. And the second largest portion is data collection. That accounts for 19% of the total time spent. So collectively, data extraction and organization step can take around 80% of the time. Well, we have already worked on the data collection in the previous module, now our attention has shifted to data cleaning and organization. That is 60% of the total time spent in a typical data science project cycle. But why this whole organization step takes so much time, and what are the key issues we would like to solve during this step? Well, that's what we'll discuss in detail in this module, and over the course of the next couple of modules. So if you look at the path from raw data to processed data, then you will find that this path is in itself iterative in nature. After getting raw data, you first explore the data using some basic summary statistics and basic visualizations. this step is called as exploratory data analysis or EDA in short. Through this step, we can also learn if the given dataset has some discrepancies or not, such as if there are missing values in the dataset, or the dataset has outliers that need to be treated before moving ahead. Then, in the next step we take care of these issues. That means we tackle the outlier issues, missing value issues, and any other issues that require some kind of processing. In the data science world, we refer to this step as data munging. So in this step, we'll look for potential issues, and then dissolve them. Then, we'll perform some feature engineering to process existing features or to generate more features that will be used in the modeling activities later on. Many times, we also create some advanced visualizations in order to gain more insights about data, and that in turn can help us to perform some more feature engineering. In fact, visualizations are very important as they become the part of the presentation step towards the end of the data science lifecycle. And not only this, we can repeat this whole cycle multiple times until we reach a desired level. Let's drill down in the EDA or exploratory data analysis phase itself, that we'll be covering in two modules. In this module, we'll be focusing on very basic EDA techniques. So we'll start with the basic structure of the dataset itself. We will also look at the key summary statistics for different features, whether it's a numerical feature or a categorical feature. So by the end of this module, we will have some basic idea about our titanic dataset. Then, in the next module we will switch our gears, and we'll learn some more advanced EDA techniques. So we will look at the distribution of different features. Then you will also learn how to get properties for a group of

observations using grouping and aggregations. And towards the end of the next module, we will also look at EDA techniques that are based on reshaping the existing dataset in some way or another. So in this direction, we will also cover crosstabs and pivot tables. So hopefully, these two modules will equip you with enough tools and techniques that you can apply to any dataset to explore it and to understand it in a better fashion.

Overview

From the conceptual point of view, this module will start with importing data to the working environment, and you will learn to accomplish this task using Python. And once we have the data with us, then we will move towards the EDA phase. As discussed in the previous clip, as a part of the EDA phase, we'll be learning about the basic structure and summary statistics in this module. We will cover the rest of the EDA techniques such as distributions, grouping, crosstabs, and pivots, in the next module. From the tools perspective, we will focus on a couple of Python libraries in this module. The first one is NumPy, and the second one is pandas. These two tools have become very popular workhorses for performing various data science activities in the Python world. Therefore, mastering these tools can be a very important step towards making of a great data scientist. Therefore, before we jump into our actual data exploration and processing phase, let's briefly talk about these couple of tools.

Introduction to NumPy and Pandas

NumPy is a very popular Python package, and is often considered as a fundamental tool for performing scientific computing with Python. NumPy helps to perform very efficient array operations, unlike the native Python lists that can lead to performance issues when dealing with large arrays. Moreover, NumPy can also be used with higher dimensional arrays and matrices. Other than that, NumPy also provides a bunch of high level mathematical functions that you can use for your scientific work. While NumPy is great in dealing with homogenous data types, often a real-world dataset contains heterogeneous data types. That means in your dataset you can have both numerical features, as well as the categorical features. so in order to deal with such scenarios, packages like pandas can be very helpful. Pandas is, in fact, built on top of NumPy, so all the goodness of NumPy, it also provides structure and operations to work on tabular data. A typical real-world dataset looks like a spreadsheet, having a tabular structure with a bunch of rows and columns. So you can think rows as observations, while each column represents a feature or attribute. And moreover, each column or feature can be of a different data type. Pandas can

help you to deal with such situations very easily and efficiently. In pandas, such tabular structure is referred as pandas DataFrame, and pandas provides functionalities for data manipulation and other common operations such as joins or groupings on these dataframes. Pandas also provides high-level functions built on the top of another very popular Python library, that is matplotlib, for creating various visualizations. So in this module, you will learn how to use pandas for performing various data science related activities. We will also use NumPy for some of the data manipulation operations. So now you have a brief overview of the Python tools that we'll be using in this module. Let's jump into the actual exploratory data analysis phase, where we will use these tools to solve real-world problems.

EDA: Basic Structure

Exploratory data analysis, also referred to as EDA, is basically the first stop you take after acquiring and importing the data. This step helps you to gain insights into the dataset, and uncover the underlying structure. So we will start with the basic structure of the dataset using pandas, but before we jump into the demo, let's see what do we mean by basic structure. Well, once you have imported the dataset in your working environment, you first want to know how many rows or observations you have in your dataset, or how many columns or features are available to you, and what are their data types? You may also want to look at a few rows of the dataset, maybe the head or the tail of the dataset, to just have the feel of how your dataset actually looks like. So now let's jump into the demo to explore the basic structure of the titanic dataset.

Demo: Investigating Basic Structure

In this clip, you will learn the use of pandas to import the dataset to the Jupyter notebook, and then use various pandas functions to investigate the basic structure of the dataset. So here we are in the Jupyter environment. By now you must be very familiar with this environment. So if you recall, we have placed all of our notebooks in the notebooks folder, so let's navigate to this folder. Here I have already created a new Jupyter notebook for you, and we have given it a name using the convention that we have discussed in the previous modules. Let's open this notebook. I will go through the different cells of this notebook one by one. I have also added comments in this notebook so that you can refer them later if you want to. I would also suggest that you work along with me so that you can get maximum benefit out of it. So as the first step, we are importing a bunch of Python libraries such as pandas, NumPy, and os, and we will use these

libraries in this notebook. Also, as a good practice, we have provided alias names for pandas and NumPy libraries so that we can use their functions using the shortcut notations instead of typing the whole word of pandas and NumPy each time. In fact, this is a very well-accepted common norm for pandas and NumPy users. So let's execute this cell to import the libraries. Next, in order to import data, we have first set our path. As discussed in the previous module, we have stored our raw training and test files in the raw subfolder in the data folder. Again, as a good practice, we have used the related part from the parent directory instead of the hard-coded path. This ensures that our code can work on both Windows and Linux-based machine without any code change. Let's execute this cell. So now our paths are set. We can import the dataset using the pandas read_csv method. In the read_csv method, we need to specify the path of the file as the first parameter. You can specify other attributes as well, if required. I would also recommend that you look into the pandas documentation to see all of the available options, because you may require them for different datasets. Well, here in our case we have only set one attribute, that is index_col. This attribute is used to choose the column as the index column of the pandas DataFrame. You can think index as the unique field to identify each row. And because in the titanic dataset each operation is seated with one passenger only, we have used the passenger id as the index column. So here in this cell, we are creating two dataframes, one containing the training data and the other containing the test data. Let's execute this cell. So our dataframes have been created. So if you use the type function on any of these dataframe objects such as train_df, then you should get a pandas DataFrame. Let's execute the cell to confirm this. And we do have the pandas DataFrame as a return value of the type function. So now we have successfully imported our data to the Jupyter environment, we will look at some very useful functions to explore the basic structure of the dataset. And the first function that we will look at is the info function. The info function can be used to get some basic information about the dataframe. Let's execute this cell to get the information for the train dataframe. So the info function spits out a bunch of useful information for us. So it says we have 891 entries in the training data. It also provides information about different columns and their default data types, as inferred by pandas. To know more about these columns and what they actually mean, you can refer to the Kaggle site as well. However, let's quickly look at these columns, because it will help us to understand the dataset. So we have already seen what the PassengerId means. The Survived column provides information if the passenger has survived the Titanic disaster or not. It is included as an integer, where 1 means that the passenger has survived, and 0 means that the passenger could not make it. The Pclass column refers to the passenger class. This column is also encoded as an integer. So 1 means that the passenger was travelling in the 1st class, 2 means the 2nd class, and the 3 means the 3rd class. Then we have the Name, Gender, and the Age column.

So then we have a column SibSp, which refers to the number of siblings and spouses aboard in the trip. The Parch column refers to the number of parents or children on the trip. We also have a Ticket column that contains the ticket number. Fare column contains the passenger fare. We also have the information about the cabin of the passenger, and the Embarked column suggests the point of embarkment, or the boarding point for the respective passenger. We have three embarkment points encoded as C, Q, and S in the dataset. That represents three locations. So coming back to the Jupyter notebook, you now have an idea about different columns of the dataset and what they actually mean. Well, other than simply the column names, the info function also provides information about the number of non-null entries for each column. For example, here in the Age column we don't have age information for many passengers. Similarly, we have cabin information for only 204 passengers in the training dataset. Also, for a couple of passengers in the dataset, we don't have information regarding their embarkment point. Info method also provides the memory usage of the dataframe. This can be very useful, especially when you have large datasets. Because you will get an idea about the memory taken by the dataframe object. So now you have some information about the training data, let's use the same info method on the test dataframe. Let's execute this cell. So in the test dataframe we have 418 entries, and again, we have some missing values in the test data as well. Don't worry, we will tackle these issues in the subsequent clips. But one thing that you can notice here is that you don't have Survived column in the test data set. In fact, that is the challenge given to us, as we have to predict the survival for passengers in the test data set. Well, we'll talk about all the partition steps in the last module. As of now, let's turn our attention on the exploration of the entire dataset. So in order to work on the entire dataset, we will first concatenate both the training and test data into a single dataframe. But before we do that, let's add a column named as Survived to the test dataframe, just like we have in our training dataframe, so that both the dataframes have similar structure. So in order to achieve this, we can simply use a single square bracket with the column name to create a new column in the dataframe, and we have set the default value as some arbitrary value such as -888, as shown here. You can pick up any other value. Let's execute this cell. So now both the training and the test dataframes have similar column structure. We can concatenate them using the pandas concat function. In the concat function, we have passed the dataframes as a tuple. You can also specify the axis parameter in the concat function. We have not shown here, because we are using the default value of 0, but you can explicitly mention the axis parameter, so let's do that. So what exactly this axis parameter is? Well, this parameter defines how do you want to concatenate two dataframes. If the axis value is 0, then pandas will concatenate row-wise, means values from one dataframe will be stacked over another. However, if you would have set the axis value to 1, then concatenation happens column-wise, means you put values from one dataframe

to the side of the values from another dataframe. Well, for this case we want to use the axis value to be 0. So let's execute this cell. So now we have one single dataframe containing both the training data, as well as the test data. Now let's again use the info function on the full dataframe. So in the full dataframe, we have total 1309 rows. Next, let's investigate a few rows from the dataset to get the feel of the actual data in the dataframe. So you can use the head function on the dataframe object to get top n rows, and by default it will return top 5 rows. Let's execute this cell. And here we have top 5 rows of our full dataframe df. If you want to see more rows, you can specify the count in the head function. In the next cell, we are passing a value of 10 to get top 10 rows of the dataframe. Let's execute this cell. And here we have top 10 rows. Similar to head, you can also look into the tail of the dataset using the tail command. Again, by default, the tail command will return last 5 rows. Let's execute this cell. And here we have the last 5 rows. So now you have explored the basic structure of the dataset, next we will move further in our data exploration journey. However, in your journey, you may have to select one or more rows or columns to get some fine-grained information. So before we move further in our data exploration journey, let's learn about the basic pandas selection, indexing, and filtering in the next clip.

Demo: Selection, Indexing, and Filtering

In this clip, you will learn about basic pandas selection, indexing, and filtering techniques. This will be very helpful to you in your journey ahead. So here we are in the Jupyter notebook, and we'll start at the pandas selection. So if you want to select any particular column from a pandas DataFrame, then you can do it in a couple of ways. The first technique is using the dot operator. So after the dataframe object, you can use the dot symbol, and then use the column name. Here in this example, we are extracting the passenger names using the Name column, Let's execute the cell. And here we have the passenger names. Alternatively, you can also perform the column selection by passing the column name as a string inside the square bracket. Let's execute this cell. And here we have the same result. Sometimes you may want to work on more than one column. In that case, we can simply pass list of column names inside the square bracket. Here in this example, we are extracting both Name and Age of passengers, so let's execute this cell. And here we have both Name and Age. So far, we have discussed about column selections, but you may want to select particular rows as well. So we can provide index of rows or columns that we want to select and extract from the dataframe. Pandas supports different types of indexing. We will talk about a couple of indexing methods that will be very useful for you in the long run. The first one is the label-based indexing. So you can use actual labels for indexing. So we can use loc indexer, where inside the square bracket we can specify indexes for rows and columns. So before

the comma, we need to specify indexes for rows, and after the comma we need to specify indexes for columns. In this first example, we have specified only the row labels, and here we want to get rows for passengers with passenger ID labels ranging from 5-10, and because we want to get all columns, we have not specified any indexes for columns. Let's execute this cell. And here we have all columns for passengers with passenger ID from 5-10. But if you want to extract only selected columns for these passengers, then you can specify column indexes as well. In the next cell, we are using the column range. So this line means that give me all columns from Age to passenger class for passengers with passenger IDs from 5-10. Let's execute this cell. And here we have the desired output. But if you want to specify discrete columns instead of column range, then you can do it using a list of column names. So here we are getting only the Survived, Fare, and Embarked column information for passengers with ID from 5-10. Let's execute this cell. And here we have the required information. So this was about label-based indexing. Pandas also supports position position-based indexing, where you can use positional indexes for rows or columns. However, remember that indexes in Python are 0-based. So here we are using iloc to specify position-based indexes for rows and columns, and here we have specified positional indexes for rows from 5-10 and for columns from 3-8. So let's execute this cell. And here we have our output displayed in the notebook. So we have seen both the label-based and position-based indexing, and depending upon your requirement, we can use either of them. In fact, you can mix and match both label and position-based indexing. However, I don't prefer such indexing, because that can be confusing at times, so my suggestion would be to stick with one indexing philosophy for one activity. Another very common requirement could be to filter your rows based on certain conditions. Suppose you want to know how many male passengers are there in the whole dataset. Then, you can do that using a very simple Boolean expression. Here we are comparing the sex attribute with the male string to get male passengers, and then we are assigning the result or filter operation to a variable male_passengers. And finally, we are printing out the number of male passengers using the len function on the male_passengers variable. Let's execute this cell. So we have 843 male passengers in the dataframe. You can have multiple conditions as well, and you can use operators like 'and' or 'or' to build more complex conditions. Here in the next cell, we are extracting rows for male passengers travelling in the first class. So we have added one more condition and used the and symbol to couple together both the conditions. Again, we are storing the result in a variable, and we are also printing out how many rows we have in the filtered dataframe. So let's execute this cell. So out of 843 male passengers, 179 were travelling in the first class. So now you have mastered the art of selection, indexing, and filtering, we can move further in our data exploration journey, and we will talk about getting summary statistics using pandas in the next clip.

EDA: Summary Statistics

Summary statistics is a very useful technique to get some interesting insights about the data as it provides numbers that can help you to summarize your data in an overall sense. So what does summary statistics include? Well, the type of the summary statistic that you want to get will depend upon the type of the feature. So if you have a numerical feature or numerical column, means if you have real-world continuous values, then you can look into its centrality measure and the dispersion measure. Typical centrality measures are mean and median, and for the spread or dispersion, you may want to look into the values such as range, percentiles, variance, and standard deviation. For categorical feature where you have categories, you cannot directly calculate centrality and dispersion measure, but you can use other metrics such as total count. You can also find out how many unique values are there in that categorical feature. You may also want to look into the category break up such as per category counts and proportions. Not only this, you can also look into the summary statistics for other numerical features, when you group the observations using the categorical feature in hand. We will go into the details later when we discuss the grouping and aggregations. So now let's discuss these summary statistics starting with the centrality measure of a numeric feature next.

Centrality Measure

Centrality measure provides you a number that you can use to represent the entire set of values for a certain feature. This number will be central to the data, and that's why we call it central tendency. Now let's look at some common centrality measures.

Centrality Measure: Mean

First, and in fact the very common centrality measure is mean or average. So you can calculate a single value for a numerical feature, and this figure will tell you about an average behavior for any numerical feature. Let's take an example. Suppose we have 10 passengers, and we have 8 values for these 10 passengers. So we can calculate mean age by adding up all the ages and then dividing up by the number of values. So in this case, we get a mean value of 10. So overall, by using the single summary statistic of 10, we can say that we have a bunch of young kids here. Well, even though mean is a very useful statistic, it has its own problems. The main problem with mean value is that it is easily affected by extreme values. Let's suppose we add one more individual to this group with an age of 98, and suddenly the mean value will jump to 18. So if you were given a single value of 18, then you would have thought that you had a bunch of adults here,

while the truth is that you have some young kids, and one very old person, so be cautious whenever you are using mean to infer some information about the data. Another centrality measure that can help us to get rid of this issue of extreme values is called as median.

Centrality Measure: Median

Median represents the middle value in the sorted list of values. Let's again take the age example. So we have 10 individuals, and we want to calculate the median value. So we can first sort the age values, and once we have the sorted list, then we can find the middle value. And as we have 10 values here, which is an even number, therefore we will have two middle values. That means that 50% of values will be above these middle values, while 50% of values will be below these middle values. Therefore, to calculate the median we can take the average of these two middle values so our median will be 10. Now let's introduce an old person in the list with the age of 98, just like we did in the mean case. So now we have 11 individuals in the list, and because 11 is an odd number, we only have one middle value, that is 10. Therefore, the median value is 10. So, as you can see, with the extreme value in the list, the median value has not been affected. Therefore, median is a much safer figure if you want to talk about the centrality measure and getting some insights from data. So we talked about a couple of centrality measures, mean and median, but these are not sufficient alone, because a few values in the data could be near the central value, while a few could be far away. That's why you may also want to look into the spread of data to get some more insights. So we will discuss spread or dispersion measure next.

Spread Measure

Well, while the centrality measure gives you a central value, the spread or the dispersion measure helps you to understand how the values are spread out from the central value. So the data spread will give you an idea about the variability in the data, means how similar or dissimilar values are there in the given data set. If all values in the data are similar, then all will be around the central value only, but if the values in the given data are very dissimilar, then they will be spread out from the central value. So now we have made some background about the dispersion measure, let's talk about a few commonly used dispersion measures.

Spread Measure: Range

Range is possibly the simplest of all, and it's simply the difference between the maximum and the minimum value. So if your range is very low, that means that the values are packed together. and

if the range is very high, then it could be possible that the values in the data are spread out. Let's again take our passengers age example. So in this example, we can get the range by getting the difference between the maximum age and the minimum age. So, in this case, it would be 12-8, that is 4. So by looking at this number, 4, you can say that it is a quite a narrow range, so the values should be very closely packed. It means the values in the list are quite similar to each other. This is great, because by having just one number we can get some information about the variability in the data. But the problem with the range is quite similar to that of mean, that is, it is easily affected by extreme values. So just like the previous example, let's suppose we have one more individual in our list with an age value of 98, and now if we calculate the range it bumps to 90 from 4. So if you were given just one value of 90 as a range, you may be tempted to infer that the values in the data are very dissimilar, as the range is very high, but that's not true, because 10 values are very closely packed, and only one value is on the extreme side. So bear such issues in mind when you are using the range metric. So what are the other dispersion measures?

Spread Measure: Percentiles and Boxplot

Well, instead of the range, you can also use percentiles as a measure of dispersion. So what is percentile? Well, if for a certain set of values x percentile is y, then it means that the x% of values in the data will be below y. So let's suppose the 50th percentile for a certain set of values is 10, then it means that the 50 percent of values will be less than 10. So typically you use a bunch of percentiles such as 25th, 50th, and 75th to get an idea about the spread of the data. These numbers are very special, because what essentially we are doing here is dividing the whole dataset in four buckets, each with 25% of values. So these special percentiles are also referred as quartiles, because they help to spread the data in four zones, and then observe the data spread. Quartiles are very useful numbers. In fact, there is a visual representation of these values that is known as the box and whisker plot. Box and whisker plot, also known as box plot, is simply visualization that shows the three percentiles that we have discussed. So I have depicted these three special percentiles using three lines here. Let's add a couple of lines more, one for minimum value and one for maximum value. Remember, we pick up these minimum and maximum after excluding extreme values in the data. I will shortly discuss how to identify extreme values. But basically, we have four buckets that we have discussed previously. Next, create a box that ranges from 25th percentile to 75th percentile. So that is our box. And for whiskers, we simply extend the line from minimum to 25th percentile, and one line from 75th percentile to maximum, and our box and whisker plot is ready. So once you know how to read a box and whisker plot, you have all the important information with you. Also, in case you might have noticed, 50th percentile is nothing

but the median value, because 50% values will be below the 50th percentile value, and 50% of value will be higher. Another very useful number is interquartile range, or IQR, which is nothing but the difference between the 75th percentile and 25the percentile. So IQR is basically the height of the box. Sometimes you will also see a few dots after the whiskers in the box plot. These dots represent extreme values also known as outliers. So anything beyond 1. 5 times the interquartile range is considered as extreme or outlier. So, as you can see, a single visualization has several interesting components in it. You will also see how we can create a box plot using pandas. But as of now, let's take a couple of more very commonly used dispersion measures.

Spread Measure: Variance and Standard Deviation

Variance is a very important measure of data spread or variability. Variance shows how far each value in the list is from the mean value. So if the variance is small, then it means that the spread is less in your data, while high variance represents large spread. If you are interested in the actual calculation, then it is very straightforward. You take the difference of each value from the mean, and then you square the difference and then sum for all the values in the list, and finally, we divide the whole sum by the number of elements in the list. So, as you can see, we have the mean in the calculation, which in itself gets affected by extreme values. That's why variance can also be affected by extreme values. But there is one more issue with variance, and that is regarding the unit of the variance value. So as you can see in the formula, we are taking the square of the difference, and then summing it up. That's why the unit of variance is basically the square of the unit of measurement. So if we have the passenger age values, then the variance will be in terms of square h, which is not very clear and intuitive. So in order to solve this issue, we take help of another dispersion measure, that is standard deviation. In fact, standard deviation is nothing but the square root value of variance. So by taking the square root, the unit of standard deviation becomes the same as the unit of the feature. Other than the unit of the standard deviation, the rest of the concepts are very similar to that of variance, means low standard deviation is a signal of less spread in the data, and high standard deviation is a signal of large spread in the data. So now you have a basic understanding of different summary statistics for numerical features, let's put all of the concepts that we have learned so far in the practical use using the titanic dataset.

Demo: Getting Summary Statistics for Numerical Features

In this clip, you will learn to use pandas and NumPy to get summary statistics on the titanic dataset. Here, we are in the Jupyter notebook. The first pandas method that we will talk about

here is the describe function. This is a very handy function to get summary statistics very quickly. So, to get various summary statistics, we can apply the describe function on the dataframe object. By default, it will return different summary statistics for all the numerical features. So let's execute the cell to see the results. So we have different types of summary statistics for all of the numerical features in the dataset. So we have mean value here, which is the centrality measure. So we can see the mean Age is roughly 30, and the mean Fare is roughly 33. If you look at the mean Survived value, it is some negative value, which doesn't make sense, but we are getting the negative value because we had saved the Survived value for the test dataset as -888, if you recall, and that's why we are getting some weird number here, so don't worry about it. Other than the mean, we also have maximum and minimum values with us. So the maximum age in the titanic dataset is 80 years, while the minimum is 0. 17. Well, probably the passenger is an infant. We also have important percentile values here. But as you can see, we have got a few NaN, or not a number here, for Age and Fare features. This is because these features have missing values. Later we can see how we can rectify this issue. Well, other than percentiles, we also have standard deviation here, that is a measure of data dispersion or spread. But overall you can see the describe function is a very handy function to get various summary statistics in one go. Now let's see how you can get different summary statistics manually. Let's start with a numerical feature, that is passenger fare. To get the centrality measures, such as mean or median, you can directly apply the mean or median function available in pandas. So first we extract the passenger fare using df. Fare, and then apply the function. Remember, by default pandas will skip missing values while calculating these numbers. So let's execute this cell now, and here we have Mean fare and Median fare. Now let's see a few dispersion measures, starting from range. So for range, we can simply take the difference between the maximum and minimum values, and that you can get using the max and min function of pandas. For percentiles you can use the quantile function available in pandas, and then you can pass the percentile point. So if you pass 0. 25, it will give you 25 percentile value. Similarly, you can get 50 percentile and 75 percentile. However, remember, if you have missing values in your data, then pandas will return nan. And as we know that we have missing values in the passenger fare, so you should get nan. Don't worry, later we will fix this issue. For other dispersion measures such as variance and standard deviation, you can use the var and std functions of pandas. Now let's run this cell to get results of different dispersion measures. So here we have the result. So fare range is 512 percentiles are nan, as expected, variance in the fare is 2678, and standard deviation is 51, which is quite high, so as you can see, we have large spread in the passenger fare. Now let's create a box plot to understand the spread of data visually. So in order to create a box plot, you can use the plot function available in pandas. So in the plot function, we can set the type as box in order to get the box plot. So here

we are trying to create the box plot on the passenger fare, so let's execute this cell. However, you cannot see the visualization here. This is because we have not told this Jupyter notebook to render the visualization in the Jupyter notebook itself. So to fix this issue, we can include one magic function that we have discussed previously, that is matplotlib inline, so let's include that. So I have created one cell, and I have used the matplotlib magic function. So now we should see the box plot here. Let's execute it once again. So here we have our box plot. So this middle line represents the 50th percentile, or the median value, and the box is extended from 25th percentile to 75th percentile. We also have quite a few outliers, or extreme passenger fare values. Well, so far we have discussed about numerical features. Let's turn our attention towards categorical features.

Counts and Proportions

Counts and proportions are two very commonly used summary statistics for categorical data. Let's again take the example of passenger data, however, this time we will focus on the gender of each passenger instead of their ages. Well, gender is a categorical feature. So, in our list, we have either male passengers or female passengers. So, the total count is 10, as we have 10 general values, and as we have either male or female as gender values, therefore, the unique count is 2. So for such categorical data, we are also interested in category-wise break up. So let's take the count breakup. So we have four male passengers and 6 female passengers here, but sometimes you are more interested in the proportions instead of the actual count. So to get the proportion, we can simply divide each category count by the total number of passengers. So for males, the proportion would be 0. 4, means there are 40% male passengers in the whole list, while we have 60% female passengers. So straightaway we have gained some insights about the data, that is, we have more female passengers on the list than the male passengers. So let's see how we can come up with such summary statistics using pandas in the next clip.

Demo: Summary Statistics for Categorical Feature

In this clip, we will use pandas and NumPy to get summary statistics for categorical features. So here we are again in the Jupyter notebook, where we have already seen the usage of describe function to get summary statistics for numerical features. But you can use the describe function on categorical features as well. All you have to do is to set the include parameter of the describe function to all, and you are done. Let's execute this cell. And as you can see, we have some information for categorical features as well. So we have total count and unique count. Like for the

Embarked column, there are three unique points where people got into the titanic ship, and the point from which most people embarked was S, shorthand for Southampton. When in this output we don't have category-wise counts or proportions, so for that you can use another useful function, that is value_counts. Here we are using the value_counts on the gender or sex attribute. Let's execute this cell. And as we can see, we have 843 males and 466 females. If you want to have category-wise proportion instead of count, then you can set the normalize property to True in the value_counts function. Let's execute this cell. So here we have the proportion with us. So roughly 64% of passengers are male and the rest, 36%, are female. So as you can see, value_counts function is a very handy function. Now let's apply this function on other columns. In the next cell, we are applying the value_counts function on the Survived column. Well, even though we have encoded the Survived column as integer, but actually it is a categorical feature, means a passenger is either survived or not survived. So here we are extracting all those passengers for which the survived value is not -888. So if you recall, we have set this default value for Survived attribute of test data, for which we don't have information of survival. So basically we are extracting the training dataset only from the dataframe, and then we are trying to find how many passengers survived in that training dataset. So let's execute the cell. And as we can see, 342 passengers survived means for 342 passengers, the value of Survived attribute is 1. Now let's apply the same on the passenger class to see how many passengers were there in each passenger class. And here we have the breakup. And as we can see, most of the passengers were in third class. Other than in numbers, you can also visualize the counts of proportions. So on the output of the value_counts, we can apply the plot function, and here we have set the plot type as bar, because we want to make a bar plot. Let's execute this cell. And here we have a really nice bar plot where each bar represents the count of passengers in the corresponding passenger class. However, there are certain issue with this plot. Well, if you can see, the labels on the x axis are not rendered properly, Also, it would be good if we can have some title of the plot. Also, you may not like the color of the bars, at least I don't. Not only this, you might have noticed some matplotlib output here, which is a bit distracting. So let's try to create this plot using various parameters of the plot function. So in the next cell, we have set the rot property to 0 so that the x axis labels become horizontal. Then we have et the title of the plot using the title attribute. And if you want to set the color of the bar, you can use the color property. Here we have set it to c, which is a shortcut notation for c on color. Also, to suppress the matplotlib output, we can use a semi-colon at the end. So now let's execute the cell. And here we have our desired output, a much better-looking plot.

Summary

So in this module, we mainly focused on some basic exploratory data analysis techniques. First, we discussed how we can leverage the pandas library to import data in the Jupyter notebook environment. Then we performed some very basic EDA tasks. We learned about various pandas functions that you can utilize to understand the basic structure of the dataset, such as number of rows, number of columns, and their data types. We also discussed various concepts related to summary statistics for both numerical, as well as categorical features. And we used various pandas functions to extract the summary statistics, and to get some very interesting properties about the dataset. Well, overall in this module we built that foundation of EDA, and in the next module, we will further build upon this foundation, and we'll learn some more advanced EDA techniques. So, see you in the next module.

Exploring and Processing Data - Part 2

Introduction

Hi, this is Abhishek Kumar, and welcome to the fifth module of the course on Doing Data Science with Python. This module is the second part of the Exploring and Processing Data modules. Just a quick recap, from the data science project cycle view, we have already covered our data extraction phase and have extracted our titanic disaster dataset. And in the previous module, that was the first part of the exploring and processing data modules, we discussed about the Organize tab, and had some basic idea about our titanic dataset, such as the basic structure and summary statistics. Well, we are still at the organize step in the data science project cycle. And if we go deeper in the organize step, we are still at the exploratory data analysis phase, and the focus of this module will remain on the EDA phase only. We will start from where we have left in the previous module, and we'll continue our data exploration journey. And along the way, we will learn some very interesting EDA techniques. Also, just to reiterate, we'll be covering the data munging, feature engineering, and advanced visualization in the next module.

Overview

From the conceptual point of view, in this module you will learn about some advanced techniques of exploratory data analysis. We will dig down in the dataset to understand the distribution of various features that will help us to get some very interesting insights. We will also talk about basic grouping and aggregations in this module. And towards the end of this module, we will focus on a couple of very interesting EDA techniques, that are crosstabs and pivots. From the tools perspective, we'll be using the same two Python libraries, that is NumPy and pandas. However, in this module, you will learn some more pandas and NumPy functions that can be very useful for you in the long run. So now we have set up our expectations for this module, let's continue our EDA journey, and look into the distribution of different features of our titanic dataset.

EDA: Distributions

Distribution is a very important concept in the EDA phase. There are quite a few visual tools that will help you to understand the distribution of data. We have already seen one such visual tool, that is a box plot, that gives you some useful insights by providing the distribution of data using quartiles. We will look at a few more visual tools next. So if you are interested in univariate distribution, means you want to visualize the distribution of only one feature at one time, then you can use the histograms to do so. It is a very popular visual tool. Kernel Density Estimation, or KDE, is another visual tool for univariate distribution, but if you want to visualize distribution of two features in one go, then you can use the scatter plot. Well, there are ways to visualize more than two features as well, which is beyond the scope of this course. So let's talk about these distributions, starting with the January distributions in the next clip.

Univariate Distribution: Histogram and KDE Plot

Histogram is basically the feature representation of distribution of values in different bins. Let's take the passenger example once again to understand this concept. So we have 10 passenger age values. Next, let's create a few buckets or bins. I have created 4 of them, and leveled them from 1 to 4. So the range of first bin is 6-8, means greater than equal to 6 and less than 8. And then we'll see if we have any age value that belongs to this bin. And, well, we don't have any. Remember, we want to consider values less than 8, and not equal to 8 for this bin. Now let's move to the second bin, which is from 8-10, and we have 3 age values that belong to this bin. Similarly, for bin 3 we have 5 age values, and finally, for the fourth bin we have remaining 2 age values. So we have a frequency table with those, and histogram is nothing but the visual representation of this

frequency table. So if you place the bins on the x axis and the frequency on the y axis, then you will get this type of plot. This plot is popularly known as histogram. Histogram is a really neat way to look into the distribution of the data. Another univariate distribution plot that is widely used in the data science world is the KDE plot. Well, kernel density estimation is a pretty advanced topic, but I'm going to give you just a glimpse of it without going into the rabbit hole. Well, we have already discussed a histogram, right, where we have looked into the frequency of values in the different bins. Well, when we are talking about the KDE plot, we are talking about the probability instead of frequency, and as you know, frequency is directly related to probability, means if a value has a higher frequency, then the chances of occurrence of such value is also high, means the probability will also be high. So when we talk in terms of probabilities, we get a smoother curve type of plot instead of the bar chart type of plot. So, as you can see, we have one peak in the KDE plot. This is a sign of unimodal distribution. You might encounter a KDE plot with more than one distinct peak as well. Again, I don't want to get into the nitty-gritties here, but it is sufficient to know that you can use a KDE plot to look into the distribution of the data, and in pandas it is very easy to create such plots without getting worried about the underlying mathematics. So we have looked at a couple of plots for visualizing univariate distribution. But once a histogram or a KDE plot is given to us, what exactly are we looking for? Well, one very important aspect that you may want to notice in the distribution is skewness. But before we talk about the skewness, let's talk about a very common distribution that you will encounter throughout your data science journey, and that is the normal distribution. So if you will look at various histograms or KDE plots in the real world, then you will encounter similar type of distributions, and among these, the most common distribution is a normal distribution even though you will rarely find a perfect normal distribution. But a perfect normal distribution can act as a standard due to its various interesting properties. Well, first of all, a perfect normal distribution will look like a bell curve, and it is very symmetric across the central line. It means that 50% of values will be below the central line, and 50% of values will be higher than the central line. So you can think of the central line as median, and in fact, in a perfectly normal distribution, the mean line is also same as the median line. It means that we have no skewness in the data. But what happens when you don't have a perfect normal distribution? As we have already seen, for a perfectly normal distribution the mean and median line is same, but if the mean line is greater than the median line, then we get a distribution that typically looks like this. It is skewed towards the right, means it has a right long tail. In such cases, the skews value will be positive, or greater than 0. On the other hand, if the mean line is smaller than the median line, then you will get a left-skewed distribution, and the skewness property will be less than 0. So now we have worked

around the concepts related to univariate distribution, let's apply these concepts on the titanic dataset.

Demo: Creating Univariate Distribution Plots

In this clip, you will learn about various pandas functions to create several univariate distribution plots. So here we are in the Jupyter notebook. Let's start with histogram. Let's create a histogram for the age feature. So in order to create a histogram, we can again use the plot function on the age values, however, this time we have set the kind property of the plot function as hist. We are also setting the title of the plot and the color of the plot. Let's execute the cell. And here we have the histogram with us. Remember, while creating the histogram, pandas neglects all the missing values. And moreover, pandas has, by default, created a few bins here. But if you want to set how many bins you want, you can do that using the bins property. Let's have more bins. Let's set it to 20. Now let's execute this cell. And here we have our histogram. Well, if you look at this histogram, it is kind of a normal distribution, but it is positively skewed. So this was about histogram, Now let's create the KDE plot for the age variable. And it's very easy to create a KDE plot in pandas. All you have to do is to set the kind property of the plot function to kde. Now let's execute this cell. And here is our KDE plot, And as you can see, it is a much smoother plot. This curve also suggests that we do have some positive skewness. Let's create a few more histograms. Let's create a histogram for passenger fare. By looking at this histogram, we can see that we do have a very skewed distribution. Let's compare the skewness of age and fare attributes. Well, to get the skewness value, you can apply the skew function on the values. So here we are applying the skew function on both age and the fare. Let's execute the cell. And the skewness for age is 0. 41, which is slightly higher than 0, however, for passenger fare the skewness is very much higher, it is almost 4. 3. So this was about univariate distribution. Now let's see how you can look at bivariate distribution using scatter plot.

Bivariate Distribution: Scatter Plot

Scatter plot is a very effective way of visualizing the bivariate distribution. Let's take our passenger example once again. Suppose, along with the age values of all the 10 passengers we also have the height information for them. So here the unit of height is an inch. So now we have two numeric features with us. One is age, and anther is height. So to visualize both of them together we can create a scatter plot. It is very simple. So all you have to do is to take one feature on the x axis and another on the y axis. So here we are using the x axis to represent the age, and

y axis to represent height, and each dot corresponds to one passenger. So, for example, this dot here represents one 80-year-old passenger with a height of 48 inches. So, as you can see, by plotting all the points we get a general sense of the distribution of both the features. You also get some information about the correlation between the two features. We can see a trend here that the passengers with higher age values tend to be taller. So now you have understood what a scatter plot is, let's create some scatter plots on the titanic dataset.

Demo: Creating Scatter Plots

In this clip, we will use pandas to create scatter plots. Suppose you want to check if passenger fares in the titanic voyage are indicative of their age, means if there exists some kind of interesting pattern between passenger age and passenger fare. So for such exploration work where you want to look at two variables in one go, scatter plot could be your go-to tool. You can very easily create a scatter plot using pandas. Here we are using the scatter function on the pandas dataframe to create the scatter plot. You need to specify the feature you want to place on the x axis and the feature you want to place on the y axis. Here we are putting age on the x axis and passenger fare on the y axis. We are also specifying the value of the color attribute, so it will be the color of the dots of the scatter plot. We have also specified a title for the plot. Let's execute this cell to create the visualization. And here we have our scatter plot. So we have a bunch of dots, but we do not see any interesting pattern here, because as the age value is increasing, fares are not changing much. But anyway, it was worth a try. Another thing that you might have noticed here that we have so many dots cluttered together that they overlap each other. So for such scenarios, you can use the alpha value trick. So in the next cell, we have set the alpha value, which is used to set the opacity of the dot. So here we have set the value to 0. 1, means 10%. Alpha ranges from 0 to 1, where 0 means completely transparent and 1 means completely opaque. Let's execute this cell. And if you look at this visualization, the most crowded zone in the plot is getting highlighted. But, again, there is no significant correlation between passenger age and their fares. But don't worry, it was worth a try. Now let's try to see if passenger class has an impact on passenger fare. So in the next cell, we have set the passenger class on the x axis and passenger fare on the y axis. We have set the alpha property to 0. 15. Now let's execute the cell. Hmm, some interesting patterns here. You can see three vertical lines, because the passenger class has value of 1, 2, and 3 only, so basically passenger class was a categorical feature that we are treating as numerical feature here. But we can get some useful information from this plot. First of all, the first-class passenger fares are high, as expected, but even in the first class there are passengers who had to pay low fare. Maybe they might have booked their ticket a long

time back, or maybe they got some kind of discount. Also, for passengers in second and third class, the fare ranges between 0 and 100. So, as you can see, scatter plots can give some great insights from the data. Well, so far we have discussed various ways to explore our titanic dataset; however, we have not touched upon a very common and useful EDA technique, that is grouping, or aggregations. But before we get into the actual demo, I want to introduce you to the basics of grouping, so let's learn about grouping in the next clip.

EDA: Grouping

Grouping, or aggregation, is a very useful technique in the exploratory data analysis phase. And in your data science journey, you will be using it day in/day out to extract some key information from the data. Grouping means group together rows or observations based on your variable of choice. Let's take our passengers example once again. So we have age information for 10 passengers, so we know in this list we have a few male passengers and a few female passengers. Suppose you want to know the average age of male passengers and the average age of female passengers, so what will you do? You will simply group the passengers based on their gender information. Let's name this group as female group and male group, and once you have these two groups you can calculate the average age for each of the groups. So we are doing a couple of things here. First of all, we are grouping our observations based on some variable, which in this case is the gender information. And once the groups are created, we are summarizing the groups based on our requirement, which is the mean age in this case. You could have performed other summarization as well, such as median age per group, but it could be simply the number of observations or count per group. Also, here we have made our groups based on one single variable, that is the gender information, but you can make groups using more than one variable as well. So, for example, suppose other than the age value you also have the information about the passenger class in which these ten passengers are travelling, and you may want to group together not only based on the gender, but also based on the passenger class. Then you can do that as well. So you can have six groups, because you have two general categories, and three passenger class categories, and then you can extract the desired summary information for each of the groups. So now we have understood what exactly a grouping or aggregation is, let's see it in action in the next clip.

Demo: Grouping and Aggregation

In this clip, you will learn to perform grouping and aggregation using pandas. Here we are in the Jupyter notebook. In order to perform grouping, you can use the groupby function available in the pandas. So inside the groupby function, you can pass the variable that you want to use for grouping. Well, if you have only one variable to crate groups, you can simply pass it as a string. Otherwise, you can use a list of column names. Here in this simple example, we want to group together our titanic dataset using the sex or gender information, and once the groups are created you can apply any summary function on any attribute of the group. Here, we are extracting the median value for Age attribute for each group. So let's execute the cell. And here we have median age for male and female passengers on the Titanic ship. So by looking at these numbers, we can see that median age for both males and females are similar. Now let's look at a few more examples. In the next line, we are creating groups based on passenger class, and then getting median passenger fare for each group. Let's execute this cell. So it is evident from the result here that first class passengers have paid more fare then second and the third class passengers. Let's try to find median age for each of the passenger class. So this time we have applied the median function on the age attribute. Well, if you want to have summary statistics such as median on multiple columns in one go, then you can do it, as shown here in the next cell. Here we are applying the median function on both Fare and Age attribute. Let's execute this cell. And we have previous two aggregation results in one table. Btu what if, if you want to extract different types of summary statistics or aggregations in one goal? Well, then you can use other very handy pandas function, that is agg, shorthand for aggregation. So here we have used the aggregation function on the groupby output, and inside the aggregation function we have passed a dictionary. So here, after grouping by the passenger class, we want to calculate the mean for fare, but median for age. Let's execute this cell, and here is our desired result. Aggregation can be a very handy function, but in pandas you can perform even more complicated aggregations. Let's see one such example. Here we are creating a nested dictionary variable named as aggregations. You can follow the similar structure if required. Let's look at the first block, and for the Fare column we want to find mean field, median field, maximum fare, and minimum fare. Also, I have shown the use of a NumPy function to demonstrate that you can use the NumPy function as well in the aggregations. The next block is for age attribute, so for age we want to calculate median age, minimum age, maximum age, and range of age. Here I have used a Lambda function to demonstrate the things you can do. The Lambda function here simply finds a difference between the maximum age and the minimum age. Don't worry if you are finding this whole thing very complicated. I would highly recommend that you take this example and make some changes maybe using some other variable for grouping, or using some other summary statistics. So coming back to the problem at hand, let's create our dictionary by running the cell. Now we can

simply pass this dictionary to the agg function as shown here in the next cell. Remember, here we are making the groups based on the passenger class. Now let's execute this cell, and here we have our result in a nicely-formatted table. Pandas is a very powerful thing, and you can do simple, as well as very complicated things quite easily here. Well, if you notice, so far we are making our groups based on only single variable, but it is very much possible to perform the grouping based on more than one variable. In the next cell, we have used both passenger class and Embarked column to get median fare for each of the combination of passenger class and Embarked column. And here we have used a list of column names in the groupby function. Let's execute the cell, and here we have our result. So for each of the passenger class, and for each of the Embarked column for the respective passenger class, we have median passenger fare. Grouping can be a very handy tool to get some crucial information from data. In the next clip, we will talk about another very useful EDA technique, that is crosstab.

Crosstab

Crosstab is a very handy technique if you are dealing with categorical features. So basically you can create cross tabulations using the crosstab. Well, what does this all mean? Let's understand this using our passenger example once again, where we have the gender and the passenger class information for 10 passengers. So we have two categorical features with us. So what is a cross tabulation? Well, you can think it like a table. So we have put together gender in rows, and passenger class in columns. Now we can fill each cell of this table. So let's try to find out how many males travelling in the first class. So we have two such passengers. Similarly, you can fill out other cells as well. So in this table, we are simply putting the number of observations for each gender and class combination. Such crosstabs can be very handy to get some quick insights from the data.

Demo: Crosstab

In this clip, you will learn to create crosstabs using pandas. So here we are in the Jupyter notebook. Suppose we want to create a crosstab on the passenger class and gender attributes. So you can use the pandas crosstab function to achieve the desired result. In the crosstab function, you can pass the passenger gender values and passenger class values. Let's execute this cell. And here we have our crosstab results. So we have 144 female passengers travelling in the first class, as opposed to 179 male passengers. Also we can see here that in the third class, the majority of the population is male. There's no doubt this table has some great information that

you can use to gain some insight from data. And crosstab is a really handy tool, and we will use it throughout this course. In fact, if you want to create some visualization on the crosstab results, you can do that by applying the plot function on the crosstab output. Here in this case we're creating a bar plot using the plot function. Let's execute this cell. And here we have our bar chart. So now you have understood how to create crosstabs in pandas, let's learn another very handy EDA tool, that is pivot table, which you can think as natural extension of the crosstab.

Pivot Table

Pivot table is a very useful EDA tool, but it can be confusing at the first glance. So let's break it down using our familiar passengers example. Suppose along with the gender and the passenger class, we also have age information for 10 passengers. So just like the crosstab, we will first create the table. So now let's try to fill the cells of this table. So if you recall, earlier in the crosstab example, we filled the cells with the count of each combination of gender and passenger class. However, this time let's fill each of the cells with average age of passengers for each of the combination. So, for example, for many passengers in the first class, we have two age values, so we can calculate the average age and then fill the first cell. So it is 9. 0. Similarly, we can fill all the other cells. And as you can notice, we have a NaN value here. This is because we have no male passengers in the third class, therefore it doesn't make sense to calculate the average age when we do not have any passengers in that bracket. And once you have filled up all the cells, your pivot table is ready. So, as you can see, in order to create a pivot table we have used four crucial information here. First, which attribute will be our row. So here the gender values are arranged row-wise. Then the second piece of information is the column information, so we have the passenger class as the column information. Third information is the attribute that is used to fill up the cells. And here we have used the age information for filling up the cells. And the fourth information is the statistic that you want to apply. And here we are calculating the mean or the average value. So now you have understood what a pivot table is, let's see how we can create pivot tables in pandas in the next clip.

Demo: Pivot Table

In this clip, you will learn to create pivot table using pandas. Here we are in the Jupyter notebook. Suppose we want to find mean or average age for male and female passengers in each passenger class using the titanic dataset. So we can create a pivot table for this. To create a pivot table, you can use pivot_table function, and just like the example that we have seen in the previous clip, you

need to specify four pieces of information. First, the row attribute, which is the index parameter here. Here we have used the gender information in the mixed parameter. Next is the column attribute, which is the passenger class. And third is the attribute that you want to use to fill the cells, which in this case is age. And the fourth attribute is the aggregate function that we want to apply on the values for each row and column combination. Here we are using the mean as aggregate function. Let's execute this cell. And here we have our result. So, as you can see, by looking at this pivot table we can see that on an average, male passengers' age is higher than female passengers' age. Well, pivot table can be very useful when you want to work on any numerical feature for different combinations of two categorical features. But remember, in pandas you can achieve the same result in multiple ways. You can get the same information using a groupby function as well, as shown here in the next cell. And because we wanted to look for different combination of gender and passenger class, we can apply the groupby function on two columns, and then extract age for each group, and finally calculating the mean value. Let's execute this cell. And as we can see, we have similar results, but not exactly in the tabular format like the pivot table, rather the values in the result are stacked over each other. But you can use one trick to get the results in the tabular format as well. In the next cell, we are applying the unstack function at the end. This will unstack the results, and will present in a tabular form. Let's execute the cell. And here we have exactly the same result, but in a tabular form. In fact, unstack is a very handy function that you may use in your work. But as I mentioned earlier, you will always find more than one way to achieve your goals in pandas, so it's good to know a few tricks. Well, that wraps up our exploratory data analysis phase.

Summary

So in this module, we finished our exploratory data analysis phase, and used several techniques to get some very interesting insights about our titanic dataset. We started with exploring distributions for different features, and checked their different properties, such as their skewness. We then talked about the very important concept of grouping and aggregations. Then, towards the end, we also learned two useful pandas functions to create crosstabs and pivots. So now we have done our initial exploration of data. Next, we will further build upon on this foundation, and will perform other activities, such as data munging and feature engineering. You will also learn to create some cool visualizations in the next module. So, see you in the next module.

Exploring and Processing Data - Part 3

Introduction

Hi, this is Abhishek Kumar, and welcome to the sixth module of the course on Doing Data Science with Python. This module is the third and the final part of the Exploring and Processing Data modules. Just a quick recap, in terms of the data science project cycle, we are still at the organize step, and in this module, we will wrap up our organize phase. Inside the organize step in the previous couple of modules, we looked at various techniques for exploratory data analysis and extracted some very interesting insights from the Titanic disaster dataset. In this module, we will further work on our Titanic dataset. We will be taking a deep dive in another very important aspect of data science activity that is data munging where we will try to fix potential issues in the dataset so that it can be used further down the line for extracting useful insights and building predictive models. So if you can recall from the previous module, we discussed that we do have lots of missing values in the Titanic dataset. So as a part of the data munging step, we will fix the missing value issues before moving ahead. We will also see if there are other issues with the dataset or not, and if there are, we will fix them as well, if required. After data munging, we will go through the feature engineering step and will create some additional features that we will use in our modeling activities. Towards the end, I will also talk about some advanced visualizations that can help to get some interesting insights into the data. These visualizations can also be used as a part of the presentation step towards the end of the project cycle, especially if you're trying to showcase your work to the various stakeholders. So overall, this module is fully action packed with lots of cool stuff, so let's get started.

Overview

From the conceptual point of view, in this module, you will learn some of the techniques of data munging, such as missing value treatment and working with outliers. Then, we will go into the art of feature engineering and will create some interesting features. You will also learn about a very important topic that is feature encoding that you will use extensively if you have categorical features in your dataset. You will also learn to create script to reproduce all of the steps to create processed data from raw data. Towards the end, you will be taken to a quick tour to a Python visualization library that will help you to create and customize compelling visualizations. From the

tools perspective, we'll further move on in our NumPy and pandas journey, and you will learn some more useful methods and techniques. Other than NumPy and pandas, you will also be exposed to another very useful Python package, Matplotlib, that can be used to create very interesting visualizations. So now we have set up our expectations for this module, let's jump into the data munging phase.

Data Munging

Data munging is a common lingo used by data scientists that involves activities such as looking into potential issues in the data and solving them using appropriate techniques. But before we delve into the solution phase, let's briefly look at some of the most common data quality related issues that you may encounter in your data science journey. Well, in the real world, there could be several types of data quality related issues, but the most common issue you will encounter is the problem of missing values where you don't have information for one or more attributes corresponding to one or more observations. The second most common issue is the problem of extreme values, also known as outliers. Outliers can significantly impact your analysis. We have already seen in the previous modules how statistics like mean, or range, or variants can be impacted by the presence of extreme values. Not only this, many times you can have erroneous or mislabeled values as well in your dataset, and it can be very tricky to detect erroneous values unless they are very extreme in nature. Well, in the Titanic dataset that we are working upon, we are assuming that the creators of the dataset have done due diligence and the dataset doesn't have erroneous values. Therefore, we will focus our attention on the other two aspects of data quality issues that are missing values and outliers. So let's start with the missing values in the next clip.

Missing Value: Issues and Solution

What is a missing value? Well, in simple terms, a missing value is a situation where the value is not known for one or more features for one or more observations. In the real-world situation, it is a very common scenario. Missing value problems can arise due to several reasons. It may be possible that the data is not available at the first place, or if there is a manual data entry process to gather data, then there could be issues with the manual process itself. Or if the data is getting logged using some automated device or equipment, then equipment error may also result into missing values. But why do you bother if you have missing values? Well, first of all, if you are doing any analysis on a dataset which has missing values and you are not doing anything about it,

then it may lead to inaccuracies in your decision making. You may even make wrong judgements that can be detrimental for your business. Also, if you are making any predictive modules using the dataset having missing values, then you may encounter issues because most of the predictive algorithms are not built to accommodate missing values. Therefore, you should try to resolve the issue of missing values to the maximum extent possible. So what are the ways to tackle the missing value issues? Let's take the simplest solution that is to simply remove all of those observations for which you don't have information. You can use this solution if you have only a few observations with missing value issues. But in all other cases, you don't want to throw away data because throwing away data means discarding information that could be useful or important for the analysis. Or you can take the alternate route to tackle the missing value issue. That is obviously much harder. This part is known as imputation. Imputation is nothing but replacing a missing value with some plausible value. So with the help of imputation, you will have more data to perform the analysis and build models because you are not discarding observations anymore. But how do you perform missing value imputation? Let's look at some of the commonly used techniques in the next clip.

Missing Value Imputation Techniques

One of the most common techniques for missing value imputation is the mean imputation. So in simple terms, for mean imputation, you replace the missing value or a certain feature by its mean value. Let's take our passengers example once again to understand this concept. So suppose you have 10 passengers, but you have age values for only 9 passengers. Then, one very simple way to impute the missing value is to first calculate the mean of all available passenger age values, which in this case is 10, and then you replace the missing value with the mean value of 10. As you can see, this is a very straightforward process, but if you recall from previous models, mean is not a very reliable figure because it can easily be affected by extreme values. So instead of mean imputation, we can use median imputation because median is less impacted by the presence of extreme values. So in the median imputation, instead of mean, we use the median value to replace the missing value. So if you want to be on the safe side, you should go for median imputation. In this particular passengers example, median is also 10, so we will use 10 as the replacement of the missing value. However, as we know that we are using median, we don't have to worry too much about the presence of extreme values in the data. If you are dealing with categorical variables, then you can go for mode imputation where you can use the mode value to replace the missing value. Taking the passenger example once again, suppose you have the problem where one of the passenger's class information is missing. Then, in this case, you can take the mode that is simply

the highest frequency class, that is 1, and use the mode value of 1 to replace the missing value. Well, the examples we have seen so far are very straightforward, and we used either mean, median, or mode to replace the missing values. But sometimes you can go for more complicated missing value treatment techniques that is beyond the scope of this course. However, just to give you an idea, there is one approach known as forward or backward fill that is normally used if you have some time series or sequential data. So for forward fill, you take previous available values to replace the next missing value, and for backward fill, you do it the other way around. You use the next available value to replace the previous missing value. There are more advanced techniques as well for missing value imputation where you can use some predictive models based on available data to predict the plausible value that can replace the missing values. Well in this course, we'll focus on simple data imputation only, but I would highly recommend that you look into other options as well. So now you have understood what is a missing value and what kind of problems it can create, let's see if we have such issues in our Titanic dataset and how can we resolve them.

Demo: Treating Missing Values Using Pandas - Part 1

In this clip, we will learn to use pandas to treat missing values in your data. So here we are in our Jupyter Notebook. Let's first see if we have a missing value problem or not in our Titanic dataset. We have already seen the use of info command on the pandas DataFrame. Let's use it once again. So here is the output of the info command. There are a total of 1309 rows, and we do have a couple of missing values in the Embarked column, as we have 1307 entries out of 1309 entries. We also have missing values in the Age and Cabin columns. So now we have predicted the issues, let's try to resolve it one by one. We will start with the Embarked column where we have only a couple of missing values. But before we work on the missing Embarked feature, let's extract those rows that contain the missing values for the Embarked column. In order to get those rows, you can use the. isnull function on the Embarked column. We have already seen such kind of filtering in the previous module. So let's execute the cell. And here we have two rows that contain null values for the Embarked column. So now we want to input these two missing values of the Embarked column. Well, how to do that? We can take several approaches here. Let's talk about the first approach. Let's try to find out what is the most common embarkment point. Well, to get the count for embarkment point, you can use the. value_counts function on the Embarked column. So let's execute the cell. And as we can see at the Southampton location, the most number of passengers have boarded the ship. So we can use Southampton to fill the missing value. Moreover, you can also observe from the table that both of these passengers survived the

disaster, so it would be an interesting thing to know that which embarkment point has the most number of survivors. So to get that, you can use another really useful pandas function that is. crosstab that we learned in the last module. We want to see among the passengers for which we have information about the survival, which embarkment point has the more number of survivors. So as you can see, we are filtering out those rows for which the survivor value is -888 as these rows are from the test dataset for which we don't have survivor information. We are then passing the Survived feature and the Embarked feature in the. crosstab function. Let's execute the cell. And here we have a nicely formatted tabular data. And here you can see for each of the embarkment points how many passengers survived and how many couldn't. It is clear from this table that Southampton has the higher number of survivors. So now we can replace the missing Embarked value with Southampton. In order to replace the missing value, you can first extract the rows containing the null value for the Embarked column, and then extract the Embarked feature, and finally assign it a value of S that is the notation for Southampton. Alternatively, you can also use a very handy pandas function meant for the purpose of missing value imputation only. This is. fillna function. So you can use the. fillna function on the Embarked column and then pass the value that will be used to replace the missing value. Here, we are setting the inplace property to True. This way the existing DataFrame will be changed. If you would have set the inplace property to False, then pandas will create a copy of the DataFrame and then replace the missing value. So for the missing value imputation, we can execute either of these two lines. But we will not be executing either of these two lines for now. In fact, we have commented out these two lines. Well, let's take a step back and try to be Sherlock Holmes here. Let's utilize one more piece of information to solve the mystery of missing Embarked column data. Let's observe a few more things about the rows containing the missing value of the Embarked column. As you can see for both of these passengers, fare value is 80. 0, and both of them were in the first class. So let's see what is a fare for the first class passengers for different embarkment points to check out which embarkment point has an overall fare near to 80. 0. So what we are doing here is essentially grouping all passenger observations by both passenger class and embarkment point. And then for each group, we are trying to calculate the median fare. Remember, as we discussed previously, median is used instead of mean to avoid the impact of extreme values. Let's execute the cell. Here we have our output. For each passenger class and embarkment combination, we have the median fare value. So if you look closely, the median fare for the first class passengers who boarded at the point C, the shorthand notation for Cherbourg, is 76, which is very close to 80. So we can say that it is very much possible that the two passengers for which we don't have embarked data might have boarded at the point C. So now let's fill the missing Embarked data using the. fillna as we have seen earlier. Let's execute the cell. And now as you will try to find rows

containing the missing Embarked feature, you should get an empty result. Let's confirm it by executing the cell. And here we have an empty result. So now we have solved the missing value issue for the Embarked column, let's use the. info function once again to see what are the other missing value issues. So now we have to fix Age, Cabin, and passenger Fare attribute. Let's tackle the Fare attribute next as it only has one missing value.

Demo: Treating Missing Values Using Pandas - Part 2

In this clip, we will try to resolve the missing value issue in the Fare attribute. Here we are in the Jupyter Notebook. Just like the previous case of the Embarked column, let's first explore the rows for which we have missing fare information. Let's execute this cell. So this passenger with ID of 1044 was traveling in the Pclass 3 and boarded at Southampton. So to replace the missing fare, we can use the median fare of passengers traveling in third class and who boarded at Southampton. So here in the next line, we are first trying to figure out passengers with class 3 and Embarked value as S. Here we have used the and operator to include both the conditions. This is a very useful trick when you have multiple conditions to filter the rows. Next, we are only extracting the Fare attribute of all such filtered rows, and finally we are calculating the median value. Let's print the median value as well. So the median fare is 8. 05. Next, we are replacing the missing value with the median fare, again using the. fillna method. So now we have replaced the missing value for passenger fare. Let's check the. info command to see if we have other missing value issues. And as we can see for the Age column, we have lots of missing values, so let's tackle it in the next clip.

Demo: Treating Missing Values Using Pandas - Part 3

In this clip, we'll try to resolve the missing value issue in the Age attribute. Here we are in the Jupyter Notebook. Again, before we do anything with the missing Age values, let's have a look at the rows containing the missing Age values using the. isnull function. Let's execute the cell. And as you can see, we have lots of rows containing the missing Age values. To be precise, 263 rows. So sometimes having so many rows displaying in the Notebook can create a clutter. Pandas does provide a neat way to handle such situation. So here we are setting the. max_row property in the pandas, and we have set it as 15. Let's execute the cell. Now let's read on the next line. And here we have only 15 rows displayed in the notebook. This is a very neat trick if you are dealing with large outputs. So coming back to the original problem of missing Age values, what can we do about it? Well, as the most basic option, we can simply replace all missing Age values with mean

age. But before we do that, let's explore the distribution of Age using the histogram. We have already seen how to create a histogram. Here we are creating a histogram on the Age attribute. Let's execute the cell. So the age is distributed from 0-80 with most of the passengers near 20s and 30s. Let's check the mean age using the. mean function on the Age attribute of the DataFrame. So the mean age is roughly 30. So if you want to replace all missing Age values with a mean value, you can do it using the. fillna function as discussed previously, but if you look closely at this histogram, we can see that we have a few extreme values in the 70s and 80s, and we know that mean can easily be affected by extreme values. So let's take some other part. It is possible that the male and female passengers have a different age distribution. So, maybe we can use the median age of males and females to replace the missing values. So here, we are first grouping together passengers by their gender information, and then we are calculating the median value on the Age attribute. So, we can see that the median age is almost similar in both the cases. Let's also try to look at the distribution of age for male and female passengers to see how they compare to each other. So in the next cell, we are creating a boxplot by grouping the age for male and female passengers. Also, you can see that we are extracting rows with only non-null age values. And here we have our boxplot. And as we can see, we have almost a similar distribution for male and female passengers. So it seems like gender may not be a very good choice to replace the missing Age values. And we will look at some other options very shortly. But before we do that, if you'd have liked to replace the missing Age with median age of male or female passengers, then you could have done that using these two lines. First, group the observations with the variable of your choice, which is Sex in this case, then get the Age value, and then use the. transform function. And inside the. transform function, pass the function to be executed. That is median in our case. So this line will return rows containing median age depending upon the gender information for each observation. Here, we are also storing the result in a variable, and then we are using the variable in the. fillna function next. Well, we are not executing these lines for now because we want to explore a better choice to replace missing Age values. So let's see the next option where we are trying to explore median age as per different passenger's class. Let's create a boxplot just like we did previously. And here we can see we have a somewhat different distribution for different passenger class. And it seems to be a better option for replacing the missing Age values. So again, if you want to use the median age for passenger class, you can do it with the help of. transform and. fillna function. And it is not a bad choice. However, let's wear our Sherlock hat once again and explore something else hidden in the data for a better missing value treatment strategy. Typically, the title in the name can give a hint about the age of the person. For example, the title of Master is used for a small kid, while the Sir title is used for an old veteran. So let's see if we can use such information. Let's explore the Name attribute to get the feel of it. So

here are some of the passenger names. And if you can see, we have a pattern in the names. First we have the family name, then we have a comma, and then a title followed by the first name. So let's try to write a function that can extract this title information. Here we have written our function GetTitle. So first we have split the name using the comma character and then picked up the second part. Then in the second part, we again used the split on the dot character and extracted the first portion. Then we are stripping out all of the white spaces and finally converting the title into lowercase. Let's execute the cell to create the GetTitle function. Next, we want to use this function on the Name attribute of the pandas DataFrame. To do this, you can again use a very handy pandas function that is a. map function. You can use. map function on the Name attribute. Inside the. map function, we are using a lambda function. So here the x will represent the Name attribute. So we are passing the name to the GetTitle function. The lambda function is a very handy mechanism for such activities, and we will use it later in the course as well. So now let's execute the cell to see what kind of output we are getting. So we are getting the titles successfully. So what are the unique titles that we have extracted? Let's apply the. unique function on the output of the. map function. And as we can see, we have lots of titles here. We have mr and mrs, miss, master, and some interesting ones too, such as countess and dona. As we have so many titles, let's clump together several titles to create a smaller list. So we have slightly modified our GetTitle function. Here we have introduced a dictionary and mapped each title to a bucket. For example, we are treating don, rev, and sir as a single Sir title. So in the return statement of the GetTitle function, we are using the dictionary to return our custom titles. Let's execute the cell. Now let's use this new GetTitle function to get the Title. This time we are also storing the titles in the pandas DataFrame by creating a new column, a new feature named as Title. So this is how you create a new or additional feature in the pandas DataFrame. So let's execute the cell. Now let's use the head command on the DataFrame to see some of the rows. And here you can find the Title column as well. Now let's again create the boxplot to see the age variation for different titles like we did previously. And here we are getting some interesting results. As expected, median age for Master title is lower than others. Similarly, for observations with title of Miss has a slightly lower age than Lady or Mrs. So it seems like the Title is a very good candidate to replace the missing Age values. So let's finally fill out the missing Age values that we have delayed for so long. We have already seen how to do it. So first use the. transform function to get the median age for observations grouped by Title, and then finally using the. fillna function to actually replace the missing values. Let's execute the cell. So now we are done with the missing Age values. Let's use the. info command once again to see how far we have reached in our missing value treatment journey. So here we can see that we have solved the missing value issues for all columns except the Cabin. It seems like we have Cabin information for only a few of the

passengers, so we'll not use it directly in our predictive models. However, when we go into the feature engineering section, we will see how we can utilize this Cabin information to some extent. But as of now, our missing value imputation journey is over. In the next clip, we will look at the second most common type of data quality issues. That is outliers.

Outliers: Detection and Treatment

Outliers, as the name suggests, is something that is significantly different from the normal behavior. There could be multiple reasons for outliers to appear in your dataset. It may be possible that someone has entered some extreme values while making a data entry or while measuring any attribute. Outliers can also be a result of your data processing logic. Many a times, by nature, some values can be extreme. But the question is why should you bother about outliers? Well, if you are performing some kind of analysis on data with outliers, then it may be possible that your analysis becomes biased due to the extreme values. We've already seen how mean, range variance, and standard deviation can be impacted by extreme values. And moreover, if you're creating predictive models, then outliers can impact your model and model outputs. So in summary, outliers can be problematic for you. But how do you detect outliers after all? Well, there are methods of outlier detection ranging from very simple ones to very complicated and sophisticated algorithms. Let's look at very commonly used visual tools to detect outliers. So if you are working on only one variable, then you can use histograms to have a general sense of data distribution and use it to detect if you have extreme values in your data. Here you can see a couple of bars are far from most of the crowd, so you may have some outliers here. Other than histograms, you can also use boxplots for univariate outlier detection. We have already discussed that you may find dots beyond the whiskers that represent outliers. So the points beyond 1. 5 times the interquartile range will be treated as outliers as per the boxplot. You might want to refer to the boxplot discussion in the previous module in case you have missed that. A scatter plot is another very useful visual tool to detect outliers, especially for a bivariate distribution. Here you can see the two points are outside the typical crowd of points near the origin. So overall, these visual tools can be very handy for outlier detection. You can also read about more sophisticated outlier detection algorithms, but it is beyond the scope of this course. Well, you know how to detect outliers. The next question is how to resolve the outlier issues. Well, as a simple solution, you can simply remove observations containing extreme values. But if you are toying with the data, then obviously you are removing some important information. Sometimes you can apply transformation on features to treat outliers. For example, you can take a log of a feature or take a square root to reduce the impact of outliers on your analysis. Binning is another very useful

technique to treat outliers. So in binning you create bins, and then you put your values in the bins. And maybe you can put your extreme values in a separate bin. Imputation is also used to treat outliers. And just like the missing values, you can impute or replace extreme values with more reasonable values. Sometimes, it can also happen that you may not want to treat the outliers, but then also you should definitely explore them. Outliers often carry some very interesting information that can be of great help. So now we have made some ground about outliers, let's see if we have some outliers in the Titanic data.

Demo: Detecting and Treating Outliers Using Pandas and NumPy

In this clip, we will use pandas and NumPy to detect and treat outliers in the Titanic dataset. Here we are in the Jupyter Notebook. Let's look at the Age attribute first to see if we have outliers in this particular attribute. Here we are creating a histogram to check the distribution of age. We have also set the bins to 20. Let's execute the cell. Well, we have already seen this histogram before, but now let's see this from the outliers perspective. By looking at this histogram, we can say that while most of the age values are around 20 and 30, a few age values are in the 70s and 80s. Let's try to explore those rows where the age is more than 70. Well, we have 6 rows in the result with one male passenger with the age of 80. Now let's explore another attribute. That is passenger Fare. In the next line, we are creating a histogram for passenger fare. And in fact, there are a few passengers who have paid an exceptionally high fare than the rest of others. Let's also create a boxplot for passengers' fare. In the boxplot, cross marks outside the whiskers boundary suggests that we have outliers. This extreme outlier is in fact significantly far from others. Let's try to investigate what's going on here and if we can find something interesting. In the next cell, we are exploring rows where passenger fare is equal to maximum fare. Let's execute the cell. And here we have four passengers in the list. As you can see, all of these 4 passengers have paid a very high fare of 512. Moreover, they have a common ticket number. So it means that all four of them could be from the same family or at least they were together. All of them were also traveling in the first class, so it may be possible that this family or the group of individuals booked their ticket at the last moment and had to pay a very high fare. And also, 3 of them survived the Titanic disaster because we have a value of 1 in the Survived column. And for the fourth one, we don't information because it is in the test dataset. We know it because if you remember, we had manually set the survived value to -888 for the test dataset. So what can you do about these extreme values? Well, we are not removing these entries as of now, but it was still worth exploring the extreme values. In fact, these extreme values often carry some very interesting insights. But also, because we know that we have a very skewed distribution of fare, we can apply

some kind of transformation to make it less skewed. Transformation is a very common way to treat outliers, at least to some extent. As we know that the passengers' fare will never be negative; therefore, we can try the log transformation. In the next cell, we are apply the np. log function on the passengers' fare. Also, if you can see, we are adding 1 to the fare deliberately because a few of the passengers' fares are 0, and log of 0 is not defined. Finally, we are also storing the results of the log in a variable LogFare. Let's execute the cell. Now let's create the histogram once again on the LogFare. So if you look at this histogram, you can find that this is now a less skewed distribution. In fact, such transformations can be really handy at times. Binning, as I told you earlier, is also a very useful technique for outlier treatment. In pandas, you can use the cut or. qcut function for binning. We are using the. qcut function here that performs quantile-based binning. In the. qcut function, we have passed the passenger fare, and then we have specified 4 as the number of bins. So basically, we are spreading the passenger fare in 4 bins where each bin contains almost an equal number of observations. Let's execute the cell to know what is going on. So here, pandas has created four bins, and you can look at the range of each bin as well. Once the bins are created, pandas has assigned each observation into one bin. You can also specify names for each bin using the labels attribute of the. qcut function. Here we have specified names for four bins ranging from very_low to very_high. You can use any names as per your wish. Now let's execute the cell. So now the pandas has assigned each observation one label. So basically, we are converting a numerical feature to a categorical feature where each bin level is one category. For your reference, such techniques are also known as discretization techniques because we are creating discrete categories on continuous numerical features. Now let's create a quick visualization to depict the number of observations in each bin. So as we have converted a numerical feature to a categorical feature, you can again use the. value_counts method to get count per category, and then we are using the. plot function and setting the plot type to be bar to create a bar plot. So here we can see that we have a similar number of observations in each bin. So now that we have seen how binning works, let's create a new feature to store the bins that we have just created. We are storing the passenger fare bins in a new column, and we have given it a name of Fare_Bin. We may also use this newly created feature in our modeling activity later. Let's execute the cell to create this feature. Well, in this clip, we have seen several methods to detect and treat outliers. Now we will move towards our next topic that is feature engineering. Well, even though I have not formally introduced you to the feature engineering term, but we have already done some feature engineering work so far. If you recall, we have already used the Name attribute to create a new Title feature. Then we used the passenger Fare attribute to create the Fare_Bin feature. So next, we will further work on our Titanic dataset, and we'll perform some more feature engineering. So let's explore the feature engineering in a more formal way.

Feature Engineering

Feature engineering is one of the most crucial aspects of a data science project cycle, and you might have heard of this term over and over again if you have ever been exposed to data science projects or competitions. So what is this feature engineering exactly? Well, you can think of feature engineering as a process where you transform raw data in one way or another to have features that are better representative of the problem at hand, and that can in turn be used to create better predictive models. Feature engineering is a very wide area and contains several activities. Feature transformation is one of those activities. Transformation could be a very simple one, such as simply taking a log like we did with the passengers' Fare in the previous clip, or it could be complicated ones where you use more sophisticated algorithms or processing methods. Feature creation is another aspect of feature engineering where based on the domain expertise you can create new features using existing features. Here, I want to highlight the term domain expertise because until or unless you understand the problem at hand, you will not be able to create better features. Feature selection is also considered under the umbrella of feature engineering where you again use your domain expertise or sometimes various feature and selection algorithms to select important features among all of the features. So you can see feature engineering involves lots of things. In fact, feature engineering is often considered as an art where you can take your domain knowledge and utilize your technical expertise to come up with innovative features. So now you have been exposed to the formal definition of feature engineering, let's perform some feature engineering on the Titanic dataset. We have already created a couple of features such as Title and Fare_Bin. Let's create a few more in the upcoming demo.

Demo: Feature Creation Using Pandas and NumPy – Part 1

In this clip, you will learn to use pandas and NumPy to create some interesting features using existing features of the Titanic dataset. So here we are in our Jupyter Notebook. Let's take our Age field and use it to create a feature named as AgeState. So if the age of the passenger is greater than or equal to 18, we will treat the passenger as an adult. Otherwise, the passenger will be considered as a child. We are creating this field because it is very much possible that children might have been given priority to be on the lifeboats, and in turn, have a better survival rate. So the AgeState could be a good indicator of survival. To accomplish this goal, we are using the very handy np. where function. So in the. where function, we need to supply three things. First is the actual comparison, and here we are comparing age values with the value of 18. So if the condition will be true, then the np. where function will return the value of Adult, but if the condition is false

for certain passengers, then it will return the value of Child. We are also storing the result in the AgeState feature, so this line will create a new categorical feature AgeState, and it will contain either the value of Adult or the value of Child depending upon the age of the passenger. Let's execute this line. So now let's quickly see how many adults and children we have in our dataset. Here we are again using the. value_counts method on the newly created AgeState column. So we have 1147 adults as opposed to 162 children in the entire Titanic dataset. Now, let's also do a crosstab to check the survival rate for different AgeStates. So in the next cell, we are passing the Survived column data and AgeState data for passengers for which we have the survival information. Let's execute the cell. So as expected, survival rate is higher among children because 63 children survived the Titanic disaster as opposed to 54 children who could not make it. So in this clip, we created the AgeState feature. Now let's create some more features.

Demo: Feature Creation Using Pandas and NumPy – Part 2

In this clip, you will learn to use pandas and NumPy to create some more features. Here we are in the Jupyter Notebook. Now let's create another feature using the parents and sibling information available in the Titanic dataset. It is very much possible that if there is a small family, and one of them got into the lifeboat, then other family members also got into the lifeboat. On the other hand, if the family size is very high, then there could be a sense of panic to select who will go on the lifeboat. Well, let's try to uncover all of these possibilities using a derived feature of FamilySize. So to create the feature of FamilySize, we are simply adding the parent size with sibling size. We are also adding 1 for the self. So let's execute the cell. Now let's quickly see the distribution of family size to have a feel of it. So as you can see, most of the individuals are either single or part of a very small family size. However, there is a small bunch on the extreme right with a very large family size. As I told you, extreme values often carry some interesting information. Let's try to explore these extreme values. Here, we are extracting rows with a FamilySize equal to maximum of FamilySize. Let's execute the cell. So here is the result. Let's extract only a few columns. So here you can see there are 11 members in the family traveling on the same ticket, and for 7 of them, we already know that they could not survive the disaster. So it is very much possible that the rest of the four who are in the test dataset might also have not survived the unfortunate event. Well overall, you can see that we are creating a very interesting story here. Now let's quickly create a crosstab on the FamilySize and Survived column to see the impact of family size on the survival rate. So let's execute the next cell. So the results show that for a small family size of 2, 3, and 4, more passengers survived. So it could be possible that if one managed to get into the lifeboat, other family members also managed to get into the lifeboat.

However, for a large family size, the survival rate is very poor. Now let's take ourselves back to the last moments before the Titanic ship sank in the North Atlantic Ocean in 1912. When passengers were put into the lifeboats, mothers with babies could have been given priority over others. So let's check if motherhood has some impact on the survival rate or not.

Demo: Feature Creation Using Pandas and NumPy – Part 3

In this clip, we will create another feature for motherhood and check its impact on survival rate. Here we are in the Jupyter Notebook, and we are again using the np. where function to create the feature IsMother. So what essentially we are saying here is that if a passenger is female and if the Parch is greater than 0, means they have at least 1 child on the ship, and the age is greater than 18 and the passenger is married, means not having the title of Miss, then we can treat the passenger as a mother and set the IsMother property to 1. Otherwise, set it to 0. So let's execute the cell. Now let's again use the crosstab to see if motherhood has impact on survival. And here we can see that 39 mothers survived on the ship as opposed to only 16 who could not make it. So definitely motherhood has some impact on the survival. Now let's see if we can create some more features.

Demo: Feature Creation Using Pandas and NumPy – Part 4

In this clip, we'll be working on another feature that is the deck information for each passenger. Decks can provide useful information about the passenger's location on the ship. It can also give a hint towards the socioeconomic status of the passenger and availability of lifeboats. Here we are in the Jupyter Notebook. In order to get the deck information, we can leverage the Cabin attribute. Let's explore the Cabin attribute to get some idea. So here we can see some pattern. We have a letter followed by some numbers. So the letter may be referring to the deck, while the number could be for the room number. You can also see we have so many NaNs as well, so it could be possible that many passengers were not assigned any particular cabins. Let's use the. unique function to get the list of unique cabins. So here you can see we have so many cabins. We have NaNs as well. And then we have one particular cabin that is T that seems to be different from all of those. So let's see passengers in the cabin T. So here we are extracting rows for which the Cabin value is equal to T. Well, we don't want to create one separate deck for one passenger, so let's assume that the Cabin value of T is there by mistake. So we can set it to NaN as shown here in the next cell. Here we are setting the cabin to NaN if the Cabin information is equal to T. Next, let's see the unique cabin list once again. So now we don't have T here. So let's try to find

out the deck information, which is the first character. And for all of the NaNs, we can create a separate deck. Let's name it as deck Z. So in the next cell, we are creating a function to extract the deck from the Cabin attribute. Well, don't get scared by this long line of code. Let's break it down. So we are using the np. where function and comparing Cabin value to null using the pd. notnull function. If it is null, then we are returning the value of Z. Otherwise, we are converting the Cabin to the string first, then extracting the first character of the string. We are also converting the alphabet to uppercase so that we have all the deck information in uppercase. Then we are using our. map function on the Cabin attribute and pass the Cabin value to get the deck information using the get_deck function. Again, we are using the lambda function trick here. Let's execute the cell. So now we have created the Deck feature, let's see how many passengers we have per deck. Again, we are using our familiar value_counts function on the Deck feature. So we have the most passengers in the Z category for which the deck information is not clear. Second most populated deck is deck C. Now let's check the survival rate for different decks using the crosstab. And here we can see that for deck B, D, and E, survival rate is pretty higher. So overall, Deck seems to be a very good indicator for survival. Well, so far we have created some interesting features. Now let's use the. info command once again to see where we are as of now. So we have 17 features as of now, a few of them as integers, a couple of them as float, and then we have a few features as object and category. So these objects or category are basically categorical features. Categorical features are good for understanding, but unfortunately, most of the machine learning algorithms which we will discuss in the next module can work on the numerical values only. They cannot process strings as such. So what can we do about these categorical features? Well, there is a technique to tackle this issue that is known as categorical feature encoding that we will discuss next.

Categorical Feature Encoding

Categorical feature encoding is a process where you take your categorical feature and then convert it into some numerical form. And as we discussed in the previous clip, we perform such conversion because almost all of the machine learning algorithms expect numerical values as input. But how do you achieve this goal of converting categorical features into numerical form? Well, you can perform categorical feature encoding in multiple ways. Let's see a few examples to learn some of the very commonly used feature encoding methods.

Categorical Feature Encoding: Binary Encoding

One of the most common techniques of categorical feature encoding is binary encoding that you can use if you have a categorical feature with two categories or two classes. Let's take our passenger's example once again that we used in the previous module. Suppose you have gender information for passengers. Gender is a categorical feature, and here in this example, gender could be either male or female, so we have a binary categorical feature. So how do you encode it? Well, it is pretty straightforward. You can construct a feature, such as Is_Male, and then if the gender is male, encode it as 1, otherwise 0. You could have encoded as Is_Female as well. So it's totally up to you how you want to encode your binary feature. But whatever part you pick, just remember what is the meaning of your feature when you're making some inferences out of it. But what if you have more than two categories in your feature?

Categorical Feature Encoding: Label Encoding

First, a very commonly used technique for encoding a multi-category feature is label encoding. Let's see an example to understand what exactly this is. Let's say we have passenger fare information for all of the 10 passengers, and we have 3 categories of fares, low, medium, and high. So now we have more than two categories. So in the label encoding, we can simply use integers to encode each level. Here we are using 1, 2, and 3 to represent low, medium, and high passenger fares respectively. So once we have decided our encoding scheme, we can then create our encoded feature by simply replacing the label with the encoded value. Well, we have used 1, 2, and 3 here, but you could have picked any 3 numbers. However, there is one little thing you should bear in mind. When you are using label encoding, then the machine learning algorithm will use the encoded value, so it will take 3 as a larger value than 1. So if you have features where you have some kind of intrinsic ascending or descending order, like the passenger fare in this case, you should use label encoding. Or in other words, for ordered categories, you can use label encoding without any issue. But sometimes categories don't have any such order. Then, in that case, you can use another type of encoding that is one-hot encoding.

Categorical Feature Encoding: One-hot Encoding

One-hot encoding is one of the most popular categorical feature encoding techniques, and you will encounter it very often in your data science journey. Let's see an example to understand this encoding technique. Suppose you have information about embarkment point for all of the 10 passengers. Here we have three embarkment points, A, B, and C. In this case, label encoding may not make much sense because there is no intrinsic order in the embarkment points. Well in the

case of one-hot encoding, you simply create as many features as the number of categories. So for this example where you have three labels, you can create three features, Is_A, Is_B, and Is_C. So if the label is A, you can put 1 in the Is_A column and 0 in the rest of the columns. Similarly, for B, you can use 1 in the Is_B column and 0 in the others, and the same for Is_C. So basically, you are using three bits of information to represent one particular feature. So you can create three features in this case to represent three Embarked categories. If you would have more number of categories, then you will require more number of bits or more number of features. One-hot encoding is a very safe way to encode your categorical features, especially when you have a small number of categories. So now you have understood the concept of categorical feature encoding, let's see how you can perform feature encoding on the real dataset.

Demo: Categorical Feature Encoding Using Pandas

In this clip, you will learn to use pandas and NumPy to encode your categorical feature. Here we are in the Jupyter Notebook. First feature that we want to encode is the gender attribute as this a binary feature. So we can use the np. where function to easily encode this feature. We are creating a feature IsMale by comparing the gender with the string literal male. And if it is true, we are assigning it as a value of 1, otherwise 0. Let's execute the cell. Then in the next cell, we are performing several feature encoding in one go, and we are using the one-hot encoding technique. In pandas, you can use the very handy. get_dummies function to encode your categorical features. In fact, you can pass all of the columns that you want to encode by passing the column names in the columns property of the. get_dummies function. The first parameter is the DataFrame itself. So we are encoding the Deck, Pclass, Title, Fare_Bin, Embarked, and AgeState features. Well, actually I'm doing some kind of cheating here. AgeState is a binary feature because if you recall, it contains only two categories, adult and child. So you could have again used the np. where technique to encode it as a single feature such as Is_Adult or Is_Child, but that is the beauty of the. get_dummies function. It doesn't mind if you pass a feature with only two categories. So you could have done the binary coding as well for AgeState, and I encourage you to try the other parts as well. I have added the AgeState binary feature just to showcase the possibilities. Also, notice that we are assigning the output of the. get_dummies function to the DataFrame itself to override the contents. Let's execute the cell. So now we have encoded a bunch of features, let's use the. info command once again to see the state of our DataFrame. So now we have 39 columns with us. If you see, we still have a few columns not numerically encoded. So we have the Cabin feature here, but we have already used it to create the Deck feature, so we can remove this feature from the DataFrame. Then we have the Name column, which we used to

extract the title, so we don't need this feature either. Then we have the Ticket feature. Well, this is also an interesting feature, and I again encourage you to look into this feature and see if you can derive some useful features from the ticket information, but we will drop this feature as well for now. We have also used the Parch and SibSp column to create the FamilySize feature, so we'll not use these two columns later. Also, we have created the feature of Is_Male using the Sex attribute, so we can remove the Sex column as well from the DataFrame. So let's remove all the extra features, and let's also reorder the columns. I always prefer the column that you want to predict as the first or the last column in the DataFrame. So in the next clip, we will see how you can drop and reorder features from the pandas DataFrame.

Demo: Drop and Reorder Columns Using Pandas

In this clip, you will learn to use pandas to drop and reorder columns. Here we are in the Jupyter Notebook. Well, in order to drop the columns, you can use the pandas. drop function on the DataFrame. You can pass the column names that you want to remove from the DataFrame as a list. Here, we are also specifying the axis property to 1 so that the drop function can work on the columns. Then we have set the inplace property to True as we want to change the DataFrame itself without creating a copy of it. So let's execute the cell. So now we have dropped our columns that we will not require in the future. Let's reorder our columns as well. In the next cell, we are first creating a list of columns using a simple list comprehension and by iterating over all columns of the DataFrame that you can access using df. columns. We are also excluding the Survived column from the list. Then, on the next line, we are adding the Survived column in front of the list. And once we have the order of columns, we are passing the column list in the square bracket. We are also assigning the DataFrame with reordered columns back to the original DataFrame. So let's execute the cell. Now let's use the. info command once again to check the final status of the DataFrame. So we have our Survived column on the top of the list. Then we have all other features which are either integer or float. So finally, we have done all the hard work to convert the raw data to the processed data. Let's save the processed data and write them to files in the next clip.

Demo: Save Dataframe to File Using Pandas

In this clip, you will learn to use pandas to save and write a DataFrame to a file. Here we are in the Jupyter Notebook. First, we are creating the path to save the processed files like we did when we first imported the data to the Jupyter Notebook. Here, we are storing the processed files to the

processed subfolder inside the data folder. We are creating the path for both training and test files. So let's execute the cell. Next, we are writing the DataFrame to the file. We already know that rows where Survived column is -888 belong to the test dataset. So in order to get only the training dataset, we can remove rows for the test dataset. Here we are using a simple filter to accomplish this task. Then we are writing the DataFrame to a file using the pandas. to_csv method. As you can see, it is a very straightforward process. Then we are extracting the rows that should belong to the test dataset that we'll save to the test file. But if you recall, the raw test data file did not have the Survived column. Therefore, we are first creating a column list without the Survived column and then using it to extract all columns except the Survived column. So now let's execute the cell. So now we have saved our processed files, and we are done with the processing part. And in fact, we can stop at this moment, but I want to push you a little further. If you recall in the data extraction module, we created a script to download the data from the Kaggle website and store it in the proper location. We did so because we wanted to have a reproducible script ready that you can simply run and get everything. Similarly, here we will put everything that we have done so far and put that into a single script so that you can run this script, and it will do all the work for you from importing the data to processing the data. Again, this is a very good practice because you don't have to execute each cell one by one. Rather than, you can run a single script from the command prompt, and you're done. So in the next clip, we'll be creating a reproducible script to perform all of the data processing steps in one go. You can find the content of the next clip a bit daunting at the first go; therefore, I have marked the next clip as optional. But I would still encourage you to take the pain because you will be learning some more pandas tools and will learn a standard practice to perform data processing.

Demo: Reproducible Script for Data Processing Using Pandas and NumPy

In this clip, you will learn to use pandas and NumPy to create a reproducible script to perform all of the data processing steps on the raw data. So here we are in the Jupyter Notebook. Just like our previous script that we created in the data extraction module, we are creating a script here, get_procesesd_data. py, that we'll place in the data subfolder inside the src folder. So let's execute the cell. Next, we are using the %%writefile magic function to write the content of the cell to the file. Now let's scroll to the bottom of the cell where we have the main function. But it is all what we have done in the last three modules, so don't worry. I will go through different sections of the script one by one. So the main function has three high-level functions. First is to read the data, second is to process the data, and the third one is to write the processed data to the file. So

let's see each function one by one starting from the read_data. So inside the read_data function we are creating our paths, then importing the files. I will not go into the details because these are the exact same lines that we have seen previously. So after importing the data, we are creating the Survived column in the test_df and giving it a default value. Then, we are creating a full DataFrame by concatenating the training and the test DataFrame. So this was our read_data function. Now let's see our process_data function next. Here again, we could have written all of the steps that we have taken in the Notebook previously, but I want to introduce you to another very useful pandas technique that is known as method chaining. Method chaining is a concept where you can use one method and then pass the results to the next method in the chain. So in the process_data, we are applying a bunch of methods on the DataFrame. The order in which we are processing the raw DataFrame is slightly different than what we have done so far. It is simply because I have set up the functions that way. Don't worry; end result will be exactly the same. So if you recall, we have first done our missing value treatment, but before we go to that step, we are creating the Title feature because we used Title feature to replace the missing value of Age. So for creating the features, you can use the. assign function and pipe it with the DataFrame. In the. assign function, we can create the feature Name and then assign it with the values that we want to put in. Here, we are again using the lambda function to do so. However, this time, x refers to the entire DataFrame. So we are applying the get_title function on the Name attribute, and the get_title function is exactly the same that we have seen earlier. So once the Title feature is created, we are all set for our missing value treatment. Here we have chained another function that is the pandas. pipe function. You can use. pipe function if you want to apply any function on the DataFrame. So here we are passing the processed DataFrame to a function, fill_missing_values, where we will be filling all of the missing values. Inside the fill_missing_values function, we are doing all of those steps that we have done already, first replacing the missing Embarked column with a value of C, then replacing the missing Fare with a median fare of corresponding passengers class, finally replacing the missing Age values with the median age as per the Name title. At the end, we are returning the DataFrame back to the process_data function. So coming back to the process_data function, after the missing_value treatment, we are creating all the rest of the features starting with the Fare_Bin. The code is almost the same, so I will not go into the details again. You can refer to the previous clips in case you have missed that. So after the Fare_Bin, we are creating our other columns, such as AgeState, FamilySize, and IsMother. For Deck feature also, just like we did previously, we are first assigning the Cabin value of 3 to nan, and then we are applying the get_deck function on the Cabin attribute. Get_deck function is also exactly the same. So coming back to the process_data method, after all of the feature creation, we are going to the feature encoding phase where we are creating the IsMale

feature using the. assign function. And then we are using the. pipe to apply the get_dummies function. We are also supplying columns attribute to set the columns that will be used for encoding. Then, we are dropping the columns that we don't need. If you have noticed, we don't need the inplace property here because we are using the method chaining approach. And finally, we are reordering columns. Here we have used a function, reorder_columns. So again, we have already seen the logic for this particular function to reorder columns, so I will not go into the details. So this was about the processing_data, so now let's see the write_data method. Again, nothing new here. We have already seen all of these lines in the previous clip. So this was our script. Let's execute the cell to create the script. So now our script is ready, let's run the Python script to check everything is fine. So everything is working, and we have a completely working script that is reproducible in nature. It means that you can run your script multiple times, and every time it will accomplish the same goal of getting a processed file in the processed folder. So now we are done with our data processing part. But I have not touched upon one important aspect of the exploratory data analysis that I had promised you before; that is creating some advanced visualizations. Even though we have created a bunch of visualizations in the last three modules using pandas, but with pandas alone you will have some limitations in terms of customization and presentation. So in the next clip, I'll take you through a Matplotlib journey. This will also serve as a foundation that you can utilize to create some more visualizations. So let's dive into the Matplotlib demo right away without wasting any time.

Demo: Creating Visualization Using MatPlotlib

In this clip, you will learn to use Matplotlib to create some interesting visualizations. Here we are in the Jupyter Notebook. So in order to use the Matplotlib, you need to first import the library. Here we are importing the pyplot module of the Matplotlib, and we are also creating an alias plt. Again, this is a very common norm followed by Matplotlib users. We are also using a Jupyter magic function, %matplotlib inline so that we can see the visualization in the Notebook itself. So now let's execute the cell. So now we are all set. Well, unlike the pandas high-level plotting functions, in the Matplotlib, you need to create all of the components of the visualization by your own. So let's start with a simple histogram chart on the Age variable. So we can use the. hist function to create the histogram, and we can pass the Age values to the function. So let's execute the cell. And here is your histogram. But other than the histogram, Matplotlib also spits out some information that it created internally to create the plot. To hide such information, you can use a. show method as shown here. We are also setting a few attributes for the histogram, such as number of bins and color. So let's execute the cell. Now we have a much better looking histogram in a nice color. Now

let's add a few more components to the plot. In the next cell, we are adding the title of the plot using the. title function. We are also setting the axis levels. For X axis levels, you can use the. xlabel function, and for the Y axis level, you can use the. ylabel function. So let's execute the cell. So now we have a chart title and labels as well in the chart. Well, another way to create exactly the same plot is to use the axis object explicitly. So in the next cell, we are using the second approach. This approach can be very handy when you want to add more subplots in a single visualization. So first, you extract the figure and axis object using the plt. subplots function. Then, you can create different chart components just like we did earlier. So you can use. set_title to set the title,. set_xlabel to set the X axis label, and. set_ylabel to set the Y axis label. So let's execute the cell now. And here we have exactly the same histogram. So what is the benefit of this approach? Well, as I told you earlier, this can be very handy if you want to include subplots. It means multiple plots into a single visualization. So in the next cell, we are trying to create two subplots placed side by side. So we want one row and two columns for our subplot regions. So we'll have two axes, ax1 and ax2. We can also set the figure size using a tuple. So we want the figure to be 14 inches wide and 3 inches in height. Then for the first subplot, we are using the ax1 axis object and for the second subplot we are using the ax2 axis object. Here we are creating two histograms. The first histogram is on the Fare attribute, and we are using the cyan color for it. And the second histogram is on the Age attribute, and we are setting the color to be tomato. So let's execute the cell. And here we have two beautiful subplots. In fact, Matplotlib is an excellent tool if you want to customize any aspect of your visualization. Now let's switch our gears to create even a more complicated visualization to have five subplots instead of two. So for five subplots, we are creating three rows and two columns. This time we are using ax_arr instead of individual axes. But you can access any axis object using a two-dimensional array indexing. So for the first plot, in the first column and first row, we can access the axis object using the index of 0 and 0. Similarly, you can use the axis objects for the rest of the subplots. So we are creating several plots here, two histograms, one for Fare, one for Age, then we are creating a couple of box plots, and finally, one scatter plot. For a boxplot, we are using the. boxplot function and for the scatter plot, we are using the. scatter function just like what we did previously when we created these individual plots using pandas. So let's execute the cell. Well there seems to be some issue here. We have a few overlaps between the subplots, and we have one empty subplot. Let's fix them one by one. So in order to remove the overlap issue, you can use the plt. tight_layout function. It will add just the subplots with some padding. So let's execute the cell once again. So here you can see we have much better subplots, and we don't have any overlapping. Now let's fix the issue of the empty subplot. So what exactly we did was we created six axis objects by specifying three rows and two columns, but we never worked on the sixth subplot. That is why it

is empty. So to get over of this issue, you can set the axis off as shown here. So now let's execute the cell to check the output. And voila, we have solved the issue of the empty subplot. So as you can see, Matplotlib offers lots of low-level functions that you can use to customize your visualization as per your requirement. So now we have seen how you can use Matplotlib to create some interesting visualization, and hopefully lessons learned in this clip will equip you with basic tools to create some more compelling visualizations. So we have almost reached to the end of the data exploration and processing phase.

Demo: Committing Changes to Git

However, before we end this module, let's commit all of our work to the Git versioning system as a good practice. So we have opened the Git bash terminal on the Titanic root folder. You can use any other terminal as well. Let's make sure we are in the Titanic project folder. Let's also check our git log. And here we have our two commits that we have done in our previous modules. Now let's use the git status to see the new changes. And here are the files that we have changed over the course of the last three modules. So let's add these changes to Git. We can safely ignore this line ending warning. Now let's commit our changes. We have also added a commit message. So our changes have been committed. We'll see the git log once again. And here we have all of our commits that we have checked into the local Git versioning system. And again, if you are pushing your changes to any public repository such as GitHub, feel free to do so. So this wraps up our exploring and processing phase.

Summary

So in this module, we covered lots of ground. We started with data munging and used pandas and NumPy to look into potential issues in the data, such as missing values and outliers. We not only learned to detect the issues, but also learned the techniques to deal with them. Then, we dived into feature engineering and created some interesting features such as AgeState, FamilySize, Deck, and Fare_Bins. We also created a script to convert raw data to processed data in a reproducible fashion. Along the way, we learned several pandas and NumPy methods, and these learnings will surely help you out in the long run. And then towards the end, we took a short journey on Matplotlib and learned to create plots and subplots. So overall, in this module, you went through lots of pain, and I want to congratulate you for your perseverance. Now we are all set for the fancy stuff, that is building the predictive model where you will use the processed

dataset to build a model that can predict who will survive the Titanic disaster and who will not. So, see you in the next module.

Building and Evaluating Predictive Models – Part 1

Introduction

Hi, this is Abhishek Kumar, and welcome to the seventh module of the course on Doing Data Science with Python. This is also the first part in the two part series of modules on building and evaluating predictive models. In the first part, we will build our foundation of machine learning and develop a simple predictive model. Then, in the second part, we will take our model to a whole new level by fine-tuning its performance and even creating an API layer on top of it in order to make real-time predictions. Not only this, along the way, you will also learn about some of the best practices for performing machine learning work in the real world. If you look at the data science project cycle that we have been following since the beginning of this course, so far we have done all the hard work. First, we extracted the Titanic dataset from the Kaggle website. Then, we went through the organize step where we explored and processed the data. Now, we are heading towards the modeling phase. In this phase, you will not only learn to build models, but also learn to analyze the model characteristics based on the problem at hand. We will also cover the presentation phase in the series of modules. If we drill down in the modeling phase, in this model, we will start with the fundamentals of machine learning and some of the core concepts associated with it. Then, we will build our baseline model. After that, we will build our first predictive model and will compare its performance with the baseline model. Then, in the next module, you will learn some more advanced topics related to predictive modeling and will learn several useful techniques to fine-tune your machine learning model. You will also learn to persist your model in the next module. That can be really useful if you want to use your trained model for real-time predictions. Now let's talk about the presentation phase. So far, we have created several visualizations that can be part of the presentation phase, especially if you are presenting your insights to different stakeholders. However, from the Titanic Kaggle competition perspective, the key deliverable is a simple text file containing the predicted survival value for each passenger in the test dataset. So we'll create these submission files as well. Not only this, in the next module, we will go one step ahead, and we'll present our machine learning model itself as a machine

learning API. This will ensure that your model is not only sitting in your laptop, but also can be a part of the full-blown data solution.

Overview

From the conceptual point of view, in this module, you will learn about some basics of machine learning. If you already have exposure to the machine learning field, then it will be a quick brush up for you, and if you're new to this field, then you will learn some fundamentals that should be sufficient for the purpose of this course. So you will learn what exactly machine learning is and what types of tasks you can accomplish using machine learning. Then we will focus on one specific task, that is classification task, as the Titanic disaster challenge is basically a classification problem. So we will see what exactly is a classifier, and then we will look at some of the common metrics to evaluate the performance of a classifier. We will then create a simple baseline model that can be used as our first Kaggle submission. You can think of this model as version 0. This will establish the baseline case. Then we will look at a slightly more complicated predictive model that is a logistic regression model, and we'll build our version 1. We will also compare the performance of version 1 model with the baseline version 0 model. From the tools perspective, we will continue to use the tools from previous modules that are NumPy and pandas. But I will introduce you to a new and very popular Python package for tackling various machine learning related activities that is scikit-learn. This is a great Python library, and it can surely make your machine learning journey quite a breeze. Now we have set up our expectations from this model, let's jump into the exciting world of machine learning.

Machine Learning Basics

In this clip, we will take a step back and will cover some fundamentals about machine learning so that all of us are on the same page when we are using the terms related to machine learning. So what is machine learning after all? Well, as a very short definition, machine learning is learning from data or examples. So you can see we have two crucial components here. First one is learning, and second one is data or examples. So you use data or examples to learn some kind of pattern or intelligence. And once you have the pattern or intelligence with you, you can use it in many ways, and one very common task is to use the pattern to make predictions. Let's see a concrete example to understand all of this.

Machine Learning Basics: Representation and Generalization

All of us use emails, right? And all of us also get lots of spam mails, and these spam mails can be very problematic if you are not attentive. But luckily, our email providers do a great job of detecting the spams automatically. And in almost all of the cases, email providers detect the spam correctly. So how do your email providers detect whether the mail is a spam or not? Well, this is a classical machine learning application, so let's see what's going on under the hood. Suppose you already have mails with you, and some experts have manually labeled them as spam or non-spam. So in this example, green ones are non-spam and red ones are spam. So these are the examples that are already available to you. Therefore, this is our available data where we have certain email features for all of the emails, such as sender URL, mail contents, or sender location. We also have associated labels for each mail, which means whether they are spam or not. So now on the given data, we apply learning mechanisms to understand the pattern. And once we have the pattern with us, then in real time when a new mail arrives, we match the new email features with the pattern, and based on the pattern, we predict whether the new mail is spam or non-spam. Well, even though it is a spam prediction example, but this is how most of machine learning applications work. You take data, you learn the pattern, and then use the pattern to make predictions.

Machine Learning Basics: Spam Classification

The data we use to understand or learn the pattern is known as training data because we are using this data to train or learn. And inside the training data we have two components. First component is the input features. So for the spam prediction problem, the email features that we talked about, such as sender URL, email content, are input features. The second component of the training data is the output feature. And in this example, the output feature is the associated label with each mail, whether it's spam or non-spam. Input features are also referred as predictors because they are used to predict the output. And for output, you may also hear the term response or outcome. So overall, you can think the training data as a spreadsheet where you have columns for input features, and then you have a column for output feature. And each row can be considered as a row that represents one observation, or one sample, or one example. So overall, we are representing each mail with the help of some input features. This process is formally known as representation. So you represent your example with the help of certain attributes or features. Also, if you look at the block diagram once again, then you can see we are using the pattern to predict the spam on the new or unseen email for which we only have input feature information. We don't have the output label. This is what we want to predict based on the learned pattern. If you will have to predict the label of one such email that is already part of the training

data, then you could have directly done that using a simple lookup in the training data, and you don't need any pattern. However, the whole point of learning the pattern is to make the predictions on the unseen cases where you don't have prior information about the output label. This is a very core concept of machine learning and is formally called as generalization. You want to learn the pattern that can make predictions on unseen cases, which means the pattern should generalize well on unseen cases. So you can think the unseen case as a test case. So in the test data, you will only have access to the input data. The output data is not available to you beforehand. In fact, you need to make predictions to get the predicted output.

Machine Learning Basics: Supervised Learning

So, we have seen that in the spam prediction problem we have the training data where we have both the input features and the output feature components. And based on the training data, we learn the pattern, and then in the test data, we have only the input features, and we need to predict output values. So such learning process where we have both input and output features in the training data is formally known as supervised learning. We use the term supervised because we already have answers or outputs in the training data that can guide or supervise the whole learning process. And you will find plenty of supervised learning applications in the real world. In fact, the Titanic disaster problem that we are tackling in this course is also a supervised learning problem because in the training data we have input features, such as Age, Fare, FamilySize, etc., and we also have the output feature that is the Survived feature in the training data, which means for all of the passengers in the training data, we already know the output. However, in the test data, we only have the information about the input features, and we need to predict the value of the Survived feature for passengers in the test dataset. Moreover, the Survived column can have only two possible values. It can be 1, which means the passenger survived, or it can be 0, which means the passenger could not survive the Titanic disaster. So the problems where in the output you can find only discrete labels are known as classification problems. Well, in the Titanic disaster challenge, we have labels of 1 and 0 that represent 2 classes, Survived and Not Survived. So the Titanic disaster challenge that we are trying to solve in this course is a classification problem. But what if the output feature doesn't have discrete labels, rather it contains real values that are continuous in nature? Well, such machine learning problems can be put under the umbrella of regression task. For example, you have a problem where you need to predict car mileage based on various car features such as car height, car width, and number of cylinders. So here in the training data, you have car mileage as the output variable. Car mileage is a continuous feature because it can contain any positive real value. Such kind of problems where the output is a

continuous variable are known as regression problems. So as you can see in the regression task, we have a slightly different problem setup than the classification task. However, one thing is common in both the classification and regression tasks. That is both are supervised learning problems because in both the cases you have inputs, as well as output in the training data.

Machine Learning Basics: Unsupervised Learning

But there are other types of learning problems as well, and a very common type of machine learning problem is to identify the pattern even when you don't have any output features. But still you have to find some kind of pattern among the training data and then apply the pattern on the test data. Well, such machine learning problems where you only have an input portion in the training data are known as unsupervised learning problems. Customer segmentation, or targeted marketing, is one such application where based on the customer information you may want to identify high-valued customers. So in the training data, all you can have is input features that may be some demographic features or previous purchase history of each customer. And based on the input features, you may want to group together similar types of customers so that you can create different clusters of customers and then target your marketing efforts depending upon the cluster, or you can focus only on the high-valued customers. Such a machine learning task is very common in the machine learning world and is popularly known as clustering where you identify patterns or clusters by grouping together similar types of observations. Well, this was a very quick introduction to the machine learning world, and hopefully, by now, you have a brief idea about what machine learning is and what it can do for you. Now let's turn our attention back to our problem at hand that is the Titanic disaster problem.

Titanic Disaster Data Challenge

Well, we have already seen in the previous modules that in the Titanic disaster training data we have both input features and output feature. So we can say that we have a supervised learning problem at hand. Moreover, because the output feature is a categorical feature and has discrete labels, it is a classification task. And because we have only 2 output categories, 0 and 1 in the output feature, therefore, this classification task is actually a binary classification task. There can be classification problems with more than two classes as well, but we'll not be talking about those in this course. However, the concepts of this course can be applied to a multiclass classification problem as well. Now, let's turn our attention back to the binary classification problem at hand. So let's again revisit the Titanic disaster challenge. So in the training dataset, we have passengers

for which we have input features, such as age, fare, embarkment point, and others. We also have associated labels with each passenger. And labels are 1 and 0, which means survived or not survived. And we are trying to use the training data and apply some learning mechanism to learn the pattern. And once the pattern is available to us, then for any new passenger in the test data, we use the new passenger's input features, and we'll try to predict whether the passenger will survive or not. In the classification problem settings, the pattern is also known as a classifier or a classifier model. So what exactly is a classifier? Well, let's talk about classifier next.

Classifier

In order to understand what is a classifier, let's take a very simple example. Suppose in the training dataset we only have two features, Age of the passenger and the Family size. And now, let's put our passengers in the training data to this two-dimensional plot. So the green ones are those who survived the Titanic disaster, and the red ones are those who couldn't. So when we say that based on the training data we are trying to make a classifier model, we essentially mean that we are trying to find a decision boundary that can separate both the classes. So in the simple example, our classifier is nothing but this simple straight line. And once we have established our classifier, then for a new passenger, we can simply check on which side of the line the new passenger is. And in this case, we can easily say that the new passenger should belong to the survived, or the green, category. So as you can see, the single separating line is our pattern, or classifier. But you might argue that why I have drawn the line like this only? You can have an infinite number of such lines that separate both the classes. Well, the process of getting the best separating line is known as the learning process, and there are many learning algorithms that you can use to find the best possible decision boundary. Well, in this example, I have shown you a very straightforward case where you have a simple line as a decision boundary, but you can have more complex decision boundaries as well. So you can have a classifier like a curve or nonlinear line as well. In reality, decision boundaries can be very complex depending upon your data and your problem. And moreover, we are only considering two input features here, but you can have more than two features as well. Let's say you have three features, such as Age, Family size, and Fare. Then, the decision boundary can be a plane instead of a line. In fact, if you go beyond three features or three dimensions, then you will have a high-dimensional decision boundary that is normally known as a separating hyperplane. Coming back to the two feature problem once again, as you can see, even for this simple case, we can have so many classifiers, some straight-line boundaries and some curved ones. You can have more complicated decision boundaries as well. But not only this, to make things more complicated, there could be different types of classifiers as

well. There are several classifiers out there in the real world. And in fact, it is a very active area of research. So a classifier can be a very simple one like a logistic regression model, or it can be more complex and fancy. Just to name a few, there is a support vector machine classifier, very popular neural network classifier, or random forest classifier. There are many others as well. In fact, there could be an entire course or set of courses on different types of classifiers only. But this is not the intention of this course. We'll be focusing on the very simple logistic regression model in this course, and we'll use this simple model to learn end-to-end machine learning model development process. But don't worry. Even though logistic regression is a simpler model, it is very powerful, and it is still used in many real-world applications. However, one very important point to ponder. If there are so many classifiers out there, how can one judge the performance of any classifier? Well, to judge the performance, first you need to have some sort of performance metric that you can use to compare classifiers. Without the predefined performance metric, you will have no base to compare the performance. So let's talk about the model performance metrics next.

Performance Metrics

The famous management guru, Peter Drucker, has once said that if you can't measure it, then you can't improve it. This code highlights the importance of performance metrics that should be decided at the first place in order to judge the performance. And Peter's statement can be applied in the machine learning world as well. Well, there are several performance metrics in the machine learning world, but we'll be focusing some of the most commonly used performance metrics. And moreover, only those performance metrics that are applicable for a binary classification problem as the Titanic disaster data challenge that we are dealing with in this course is a binary classification problem. We will first look at one of the most commonly used performance metrics for a classification problem. That is accuracy. In fact, for the Titanic disaster Kaggle challenge, accuracy is the performance metric used to judge the performance of each Kaggle submission. Other than accuracy, we will also look at a couple of more performance metrics that are precision and recall. So let's start with the accuracy metric.

Performance Metrics: Accuracy

Accuracy is possibly the most commonly used performance metric for a classifier. It is very easy to understand as well. Let's take a quick example. Suppose on your test data you have made some predictions for the output label. And if you have the actual output already available to you,

then you can compare your predictions with the actual output to judge how good was the prediction of your model. Accuracy is nothing but the total number of instances where your prediction matches with the actual results divided by the total number of instances. Here in this example, out of 10 rows in the test data, for 6 rows our prediction matches with the actual output. Therefore, our model accuracy is 6/10. That is 0. 6. So we can say that our model is 60% accurate. Well in the ideal case, you want your classifier's accuracy to be as high as possible.

Performance Metrics: Precision and Recall

Precision is another commonly used metric to judge the performance of a classifier. This particular performance metric is most suited for binary classification problems. So what do we mean by precision? Well, suppose you have two classes, negative and positive. So you can create a table or a 2x2 matrix where row labels represent true or actual count and column labels represent predicted count. Now let's fill each cell of this table, or matrix. So suppose for observations in the test data actual classes are negative, and your classifier also predicted them as negative. Then, the count of these observations will go into the true negative bucket that we have denoted as TN. Similarly, we can have TP, or true positive, where your model predicted positive and actual values were also positive. Next scenario could be where actual true values are negative, but your model predicted them as positive. So it is false positive count. Opposite of this will be false negative. So your aim is to come up with this table. In fact, there is a particular name to this table. It is called as confusion matrix because this table gives you an idea where the model got confused and made errors in the prediction. Now let's use this confusion matrix to answer a few questions. And the first question is what fraction of positive predictions are correct? So the focus is only on the positive predictions, and the fraction is nothing but the proportion of true positives in the total positive predictions, and the total positive predictions is nothing but the sum of true positive and false positive. In fact, this particular fraction is known as precision. For example, let's say we have the following confusion matrix for a model. So for this case, true positive count is 60, and the total positive predictions is 40+60. That is 100. So the precision would be 60/100, which means 0. 6. Now let's take another question that is what fraction of actual positive cases were you able to catch or predict correctly? So this time, focus is only on the actual positive cases, so it will be true positive divided by total positive cases. And the total positive cases is nothing but the sum of true positive and false negative. In fact, this fraction is known as recall. Let's take the same confusion matrix that we looked into the previous slide. So for this case, the recall would be 60 divided 60+30. That is roughly 0. 67. So ideally, you'd like to have high precision and high recall for your classifier. Well, so far we have understood what is a

classifier and what are the performance metrics to assess the performance of a classifier. But how will you measure the performance metrics in the real world? So let's look at the overall model evaluation process in the next clip.

Classifier Evaluation

A classifier can be evaluated in multiple ways. However, one very common technique of evaluation is known as train test split. Let's take an example to understand this concept. So if you recall, in our Titanic challenge, we have two datasets. One is the training dataset where we have both input and output, and the second one is the final test data for which we don't have any output information. In fact, we need to predict the output by learning the pattern using the training data. But until you know the actual answer, how will you assess how your model is performing? One simple way is to make predictions on the final test data and then make a submission in the Kaggle platform because only in the Kaggle platform the actual output of the final test data is available. But if you want to assess the performance of your model without submitting your predictions, what can you do? Well, you can follow a few steps in that case. So you take the original training data and split it into two portions, one we call as training data, and the second we'll call as test data. Well this is not the final test data, rather a portion of actual training data that we will use to test the model performance. Let's say you have used an 80/20 split, which means 80% is the training data and 20% is the test data. You could have picked any other proportions as well, though 80/20 is a common choice to split. However, the key point is that in this setup, now we have input and output in both training and test data. So now you can use your training data to build a model, and once the model is trained, you pass the input features of the test data to the trained model to get the predictions. And as you already have the actual output in the test data, you can compare the actual output and predicted output to get the final score of your model. Remember, the score will be based on the performance metric that you have selected to compare the actual and predicted results. So once you feel that your model's score has reached your desired target, you can pick the model and use it to make predictions on the final dataset and make submission on the Kaggle platform. In fact, this is a common flow for most of the machine learning projects. Well, now we have made our foundation in terms of classifiers, various performance metrics, and the process to evaluate the performance metrics, let's build our baseline model and evaluate its performance in the next clip.

Baseline Model

Baseline model should be the first step in your machine learning modeling journey. Typically, people don't create the baseline model and directly jump into using machine learning to build models. However, as a best practice, you should always start with a baseline model that doesn't use any machine learning at all. Yes, you heard it right; build a model without any machine learning. And also, remember, don't skip the baseline model because with the help of baseline model, you will assist the performance of any subsequent model that you will create. This will also help you to keep track on whether your model is performing better than the baseline or not. So what is a baseline model? Well in a classification problem setting, baseline model always gives the output of the majority class. Let's say you have a binary classification problem with two classes, 0 and 1, and 1 is the majority class with 60 observations, so your baseline model will always return class 1 as the output. Now let's say if your performance metric is accuracy, then in this example, model performance of our baseline model will be 60 divided by the total number of observations. That is 60 divided by 100, which means 0. 6, or 60%. So our baseline model accuracy is 60%. So now if you go ahead and start building machine learning models, then it should be better than the baseline model in terms of model performance; otherwise, it does not make sense to use machine learning model, right? So now you know that what is a baseline model, let's create our baseline model for the Titanic disaster challenge and assess its performance. But before we can create our model, we need to prepare our data so that it can be fed into the model. So in the next clip, we'll prepare our data, and we'll also perform train test split that we had discussed in the previous clip.

Demo: Preparing Data for Machine Learning Model

In this demo, you will learn to prepare your data for the modeling purpose. Here we are in a new Jupyter Notebook. So in order to prepare the data, first we need to import our processed data into the Jupyter environment. So let's do that. So first we are importing the required libraries. Then, we are importing our processed train and test file that are placed in the processed subfolder inside the data folder. So let's set the path first and import the files using the pd. read_csv method. So now we have imported our files, now let's quickly look into the train and test DataFrame. So in the train DataFrame, we have 891 rows and 33 features. Out of these 33 features, Survived is the output label while the rest of the 32 features will be used to build the model. And in the test DataFrame we have 418 rows, and we need to predict survival for these passengers. So before we build our predictive model, let's prepare our data. Remember, most of the machine learning algorithms expect numerical arrays. So here we are creating two arrays, one for input and another for output. So in order to create the input array X, we are taking all rows and all columns except the Survived column. Here we are saying that extract all columns of the

train DataFrame from the Age column onwards. And if you look into the DataFrame, the second column is Age, and we want to take all columns starting from Age as model input features. So coming back to our input X, once we have extracted the required rows and columns, we are converting the DataFrame to a matrix and then converting the data type of each element of the matrix to float. So now X will be our matrix where each value is a real float value. For the output labels, we are creating the array y using the Survived column. Here we are also using a NumPy. ravel function that creates a flattened one-dimensional array. So let's execute the cell. Now let's quickly look at the shape of these arrays that we have created. And also, as X and y are NumPy arrays, we can use the. shape attribute to get the shape. So let's execute the cell. So the shape of X is 891x32, which means 891 rows and 32 columns. Each column here represents one feature, and for y, we have a one- dimensional array with 891 values. Also, if you might have noticed, I have used uppercase for the variable X and lowercase for the variable y. While it is not mandatory, but as a good practice, you may want to use an uppercase symbol for matrix and a lowercase symbol for one-dimensional array that is also called as vector. Now let's split the array X into two parts, one for the training the model and another for evaluating our model performance. We have already talked about the train test split in the previous clip, so I will not go into the details here. Well, the scikit-learn provides a train_test_split function to accomplish this task. So first we are importing the function from the scikit-learn, or sklearn package. Then, inside the train_test_split function, we are passing our arrays X and y. We are also specifying the test_size to be 0. 2. So essentially, it means that 20% of the data will be used for model evaluation, while the rest, 80%, will be used for model training. We are also setting the random_state. You can think this as a seed value or randomization. This is also a good practice to specify the random state so that every time you run this line you get the same output. So from the array X, we have two portions, X_train and X_test, and from array y, we have y_train and y_test. We are also printing the shape of each of these newly created arrays. So let's execute the cell. So we have 712 rows in the train portion, and the rest, 179, in the test portion. Sometimes it is also a good practice to check how many positive class values are there in the train and test data. So in the next cell, we are using the np. mean function to get the proportion of positive classes. Here, the positive class means the value of 1 if the passenger has survived. So let's execute the cell. So there are a couple of observations from the output here. So the first observation is that we have a similar proportion of positive class in both train and test dataset, both containing roughly 39% of the positive cases. So ideally, you want positive cases to be evenly distributed in the train and test data. The second observation is that only 39% of data has positive cases, while the rest 61% are negative classes. So we have some kind of imbalance between the positive and negative class. Well, we'll not pay too much attention to the problem of imbalanced class here in this course because 38% is still fine. But you

may encounter problems where you have a highly imbalanced dataset. For example, in marketing conversion problems, the conversion percentage may be too small. It could be 2 or 3% or even less than that. For such kind of problems, you need to follow a slightly different approach for building and evaluating models. We will not cover the imbalanced class problem in this course, but I wanted to give you an idea that you should always check for imbalance. So coming to the problem at hand, we have now prepared our data that we can feed into the model. So let's build our baseline model in the next clip.

Demo: Building and Evaluating Baseline Model

In this demo, you will learn to build and evaluate the baseline model. So here we are in the Jupyter Notebook. So in order to build a baseline model, we'll be using a new scikit-learn function, DummyClassifier. However, this classifier is available in v0. 19 onwards. So let's check the version of our scikit-learn. So first, import the sklearn package. Then use the version property to check the version. If it is not 0. 19 or above, then you need to update your sklearn package. If you are using the Anaconda distribution, then you can simply use the #! conda update command to update the scikit-learn package. Also, remember if you are updating your scikit-learn package, then you'll have to restart your kernel. So you can go to Kernel and click Restart. And obviously, you'll have to again rerun all of the previous cells if you are restarting your kernel. And if you already have the v0. 19 or above, then you can proceed further. So in order to build a baseline classification model, we can first import the DummyClassifier function available in the sklearn. dummy. Then we can create our model object. Here we have specified the strategy as most_frequent because the baseline model we want to create will always output the majority class. That is not survived or 0 in our case. Again, as a good practice, we are setting up the random_state. Once the model object is created, we can train the model using the. fit function on the model object. In the. fit function, we need to provide two things. One is the input data and second is the output data. So we are using the X_train and y_train to train our model. So let's execute the cell. Then, to evaluate the model performance on the test data, you can simply use the. score method. Here we are passing the test data to evaluate the performance. So model will first predict the output on X_test, and then it will compare the predicted output with the actual output. That is y_test. Also, remember for a classification model the default score represents the model accuracy. And we have already seen what is accuracy for a classifier. So let's execute the cell. And the model score is 0. 61, or 61%. This is a very important number. This is our baseline accuracy. So what we are saying here, even without using any kind of machine learning algorithm, if you will always predict the value of 0, or non-survived, then you will get the 61% accuracy. While

the. score function will give you accuracy by default for a classification model, you can also explicitly calculate accuracy score along with other performance metrics. So in the next cell, we are importing functions for a bunch of performance metrics, including accuracy, confusion_matrix, precision, and recall. The method signature for all of these metrics are also the same. The first parameter is the actual output, and the second parameter is the predicted output. So to get the predicted output, you can use the. predict function on the input data. So now let's get the value of each of these metrics. So the accuracy score is 0. 61. This is the same as the output of previously used. score function. Here is the confusion matrix, and it is clear from the confusion matrix we are always predicting the value of 0 or the negative class. So our precision and recall should also be 0 because true positive count is 0. Let's confirm it. And yes, precision and recall values are also 0. We are also getting some warning that we can safely ignore. Don't worry about these poor numbers. After all, we have only built a baseline model. But we are all set to make our first Kaggle submission. So in the next clip, we'll submit our final test results using the baseline model that we have just created.

Demo: Making the First Kaggle Submission

In this demo, you will make your first Kaggle submission using the baseline model. So here we are in the Jupyter Notebook. First, we are preparing our final test data on which we need to make the predictions. Again, we are using the. as_matrix method to convert the test DataFrame to a two-dimensional matrix, and then we are converting the values to floating type. Remember, the test_df is the DataFrame for which we don't have the actual answers. Only the Kaggle platform has the actual answers. So we will submit our predictions on the Kaggle platform, and the Kaggle platform will perform the evaluation by comparing our predictions with actual output. So let's first create this matrix, test_X. Then, in order to get the predictions, we can use the. predict method. So we are saving all of the predictions to the predictions variable. So now we have the predictions, but we need to attach the predictions with the PassengerIds. In the submission file that we have to upload on the Kaggle platform, we need to give a Survived value for each PassengerId. So in order to get this type of submission file, we are preparing a pandas DataFrame with two columns. We can get the PassengerIds from the index value of the test_df Dataframe. Remember, we have used the PassengerId as the index column when we had imported the processed data. Then we are creating a column, Survived, where we have placed our predictions. So let's execute the cell to create this DataFrame. So now we have our DataFrame. Let's look at top few rows using the. head command. So we have the PassengerId and predicted Survived values. Now, we can simply write this DataFrame to the disk. We are putting all of our

submissions in the external subfolder inside the data folder. Inside the external folder, we will create the submission file 01_dummy. Let's execute the cell. Next, we can use the. to_csv method to write the DataFrame to the disk. Also, we are explicitly setting the index property to False. Otherwise, one more column will be added in the final output file with indexes, which we don't want. So let's execute the cell to create the file. So now our file has been created. Well, let's also put all of the steps of creating the submission file into one single function because we will be following the same steps for our future models as well. So here we are creating a function, get_submission_file, and the steps are exactly the same that we have seen already. That is converting the test DataFrame to a matrix, making predictions, and then creating a DataFrame, and finally, writing the DataFrame to the disk using the. to_csv method. Here we are using the model and filename as function parameters so that we can change them based on the model and submission filename. So let's execute the cell to create this function. Now let's invoke this function on the baseline model. So now our first submission file is all set. Let's submit our prediction file to the Kaggle platform. You need to log in to make the submissions. I have already logged in. So now we get to the Titanic competition, then click on the Join Competition. Here you can upload your submission file. So the submission file is there in the external folder. Here's the file. So it may take some time to upload the file. So the file is uploaded. You can also mention some comments. Now click Make Submission. So now your submission is being evaluated. Now let's see where we are in the leaderboard. And viola, your first submission is done, and you got some rank. Don't worry! We have only made our first submission through using the baseline model. So now we have made our baseline model, let's build a machine learning model and see whether it works better than the baseline model or not. The classification model that we will build in this course is a logistic regression model. But before we build the logistic regression model for the Titanic disaster survival prediction, let's develop some intuition for logistic regression in order to understand how it works. However, in order to understand how logistic regression works, let's take a step back and quickly see how linear regression works. This will help you to have a better intuition for logistic regression as well. So let's quickly learn about linear regression in the next clip.

Linear Regression Model

Linear regression is a very important machine learning model and is a base for several machine learning algorithms. So understanding linear regression will surely pay off in the long run. Let's take an example to understand how a linear regression model works. Suppose you want to predict an individual's income based on his or her age, and we have a bunch of data points where

we already have age and income values with us. So each green dot here represents one individual for which the age and income combination is known. Now, our aim is to learn the pattern and then use the pattern to predict income based on a given age value. Let's reiterate the things that we have learned so far about machine learning in this model. So in this problem, in the given training data that are represented by green dots here, we have both input and output feature. Age is the input, and income is output. And therefore, first of all, this is a supervised learning problem, and moreover, since the output feature that is income in this case is a continuous feature, the problem at hand is a regression task. Well, for a linear regression model, what essentially we do is to find a line that fits the data points in a best possible manner. Suppose here is the best fit line, so this fit line is our model. And once we have the line with us, we don't need our training points anymore. So let's forget them for the time being. So here is our model with us, and we can use it to make predictions. So suppose we have an age value for which we want to predict the income. We can simply create a vertical line passing through this point, and the point at which this line cuts on the model line, that will be our predicted value. Also, from the high school mathematics, you know that the equation of a line can be represented as intercept term plus the slope multiplied by the X variable. Here also, we can write the equation of the fitted line as income equals to b0, that is our intercept, plus b1, that is the slope, multiplied by the age. These b0 and b1 are also known as coefficients of the model. So if you have b0 and b1, then you can simply plug the age value and predict the income. So hopefully, you now have the basic understanding of how a linear regression model works. Let's extend this intuition further to understand logistic regression.

Logistic Regression Model

Logistic regression is again one of the most used machine learning algorithms for a classification task, so let's try to develop some intuition about how it works. Again, let's take one example. Suppose you want to predict the loan grant based on the income value, and we have a bunch of data points where we already have the income and their loan grant status. Now, our aim to learn the pattern and then use the pattern to predict chances of loan grant based on the given income value. Again, in this problem, we have both input and output features in the training data. So income is input, and loan grant is output; therefore, first of all, this is a supervised learning problem. And since the output feature, that is loan granted in this case, is a categorical feature, the problem at hand is a classification task. And since we have two classes in the output, that is 0 and 1, it is a binary classification problem. Now let's try to apply the concepts that we have learned in the linear regression in the previous clip. Let's try to fit a line like we did for the linear

regression. And once we have the fitted line, then for a given income we'll again follow the same steps, get the vertical line and check where it cuts the model line. Let's say it cuts the model line at 0. 8. Now, the issue is that the loan granted can either be 0 or 1; therefore, it does not make sense that we get the value of 0. 8. So what do we do here? Well, let's make a slight change in our thinking. Let's treat the Y axis as the probability of the chance of getting the loan. So now you can think 0. 8 as 80% chance of getting the loan. So now for any income that is in this range, we can get the predicted probability, and it will be between 0 and 1. But what if we go beyond the range? Well, the probability will be either less than 0, or it will be more than 1 because the predicted line goes beyond the 0-1 limit. And we know that probability cannot be less than 0 or greater than 1. So let's make a small change in our model. Let's clip the predicted line for points outside the range. Well it seems to be a fair choice; however, in the actual logistic regression model, we have a slightly different model. So instead of these wide straight lines, the shape of the model is a curved one, similar to what you can see here. Apparently, there is a name to this curve. This is known as sigmoidal curve, or the sigmoid function curve. Now let's also draw one more horizontal line at 0. 5. This we'll call as threshold line. So if the predicted probability of the output value is higher than this threshold value, which means greater than 0. 5, then we will treat the output as the class 1, else we'll consider it as class 0. So for an unknown point, we can get where it crosses the curve. Let's say the value is 0. 85, and as it is more than 0. 5 threshold value, we can put this new point into class 1. So this is how logistic regression works on a very high level. Let's look at some background mathematics as well. This may be somewhat heavy, but I will try to make it as simple as possible. So let's start from the creation of the line that we showed in the previous clip, but we already know that this line has certain issues, and we need to think the output in terms of probability. And moreover, we need the line to be in the 0-1 range, so for that, we use the sigmoid function. From the equation perspective, it is nothing but 1 upon 1 plus e to the power minus Granted. So for the unknown point, you first calculate the granted value using the equation 1 and then pass it to the sigmoid function to get the probability using the equation 2. And if the probability is greater than the threshold, it will be class 1, else it will be class 0. So hopefully, now you have developed some good intuition about logistic regression and its associated concepts. Now let's see how you can make a logistic regression model by your own in the next clip.

Demo: Building Logistic Regression Using Scikit-Learn

In this demo, you will learn to create to a logistic regression model using the scikit-learn library. Here we are in the Jupyter Notebook. So in order to create the logistic regression model, you first

need to import the LogisticRegression function that is available in the sklearn. linear_model. So let's import this function. Next, we have exactly the same steps like the baseline model. So I will not go into the details this time. So first, you create the model object. Then you train your model on the training data. Next, we can get the model score on the test data. So here you can see our model performance has improved to 0. 83 from the baseline value of 0. 61. And because we have the baseline performance with us, we can say that our predictive model is better than the baseline. Now let's quickly get some performance metrics like we did for our baseline model. So the accuracy is 0. 83. Then we have our confusion_matrix as well. And the precision and recall have also improved to 0. 78 from the baseline value of 0. And moreover, the whole purpose of training a logistic regression model is to find out the optimal weight of coefficients. So if you recall from the previous clip, we have the model coefficients b0 and b1 that we wanted to find out. So in order to get these coefficients for the Titanic disaster challenge, you can use the. coef property on the model object as shown here. And here we have the model coefficients for each of the input features. These coefficients have different names. You can also call them model weights or even model parameters. So in summary, model parameters are internally optimized during the training process. Well, for the Titanic Kaggle challenge, we don't care too much about the model parameters as the focus is on the predictions. But if you want to know the relative impact of different input features on the output, then you can leverage these coefficients to get some useful inferences. So hopefully, now you have a better understanding of the logistic regression model and how to build this model using sklearn. Well, since now we have an improved model, let's do our second submission on Kaggle in the next clip to check if we are getting some jump in the ranking table.

Demo: Making Second Kaggle Submission

In this demo, you will make your second Kaggle submission. So here we are in the Jupyter Notebook, and here we are using the get_submission_file function that we had created previously in this module. However, this time we are passing the new model object to the function, and we are also passing the new filename that will contain the second submission. So let's execute the cell. So now we have our second submission file ready, let's submit it on the Kaggle platform. So here we are on the Kaggle platform. Let's upload the submission file. And here is the second file that we created. Let it upload. So now the file is uploaded to the Kaggle platform. Let's add some comments. Click on the Make Submission button. And viola, we have our second submission, and our rank in the leaderboard also has improved significantly. Well, we'll further improve our model in the next module, but the whole point of this course is not to achieve the number one rank in

the Kaggle competition, rather to understand the complete data science flow and to do things in a standardized approach. Well, we have now reached to the end of this module.

Summary

So in this module, we did lots of cool stuff. We started our journey by building a solid foundation for machine learning. We looked into the fundamentals of machine learning, the types of tasks that you can accomplish using machine learning, and then we delved into classifiers and learned how to train and evaluate a classification model. We then went ahead and made our first baseline model. We also made our first submission on Kaggle using the output of the baseline model. Then, we looked into logistic regression model and went through the core concepts behind logistic regression. Then we also built our logistic regression model using scikit-learn Python library and made our second Kaggle submission. By using the logistic regression model, we also got significant improvement over the baseline model. Now in the next module, we will build on the foundation, and we'll go through some of the advanced topics in the machine learning world. That will take your data science journey to a whole new level. You will learn about some very interesting concepts related to model tuning, model persistence, and API development. So overall, plenty of fascinating stuff to cover, so see you in the next module.

Building and Evaluating Predictive Models – Part 2

Introduction

Hi, this is Abhishek Kumar, and welcome to the eighth and the final module of the course on Doing Data Science with Python. This is also the second part in the two-part series of modules on building and evaluating predictive models. In the first part, we built our machine learning foundation and learned to build predictive models. In this module, you will learn some more advanced topics related to predictive modeling, and we'll fine-tune our predictive model that we built in the last module. Just to reiterate, if you look at the data science project cycle, we are in the modeling and present phase, and we will wrap up our journey in this module. If you drill down in the modeling phase, in this module, we will proceed from the point where we have left our predictive modeling journey in the previous module. So we'll fine-tune the predictive model that

we had built in the previous module. We will also learn about model persistence to save your train model for future use. In the presentation front, we will create our next version of the submission file for the Kaggle Titanic disaster challenge after fine-tuning our machine learning model. Towards the end, you will also learn to use Python to create an API through which you can expose your machine learning models to external systems. This will enable your machine learning model to be a part of any full-fledged data solution.

Overview

From the conceptual point of view, under the umbrella of model tuning, we will start with a couple of code machine learning concepts. First one is the discussion around model underfitting and overfitting, and the second one is regularization that helps to tackle the overfitting scenario. Then, we will talk about a couple of techniques that I use to fine- tune the machine learning models. First one is the hyperparameter tuning, and the second one is cross-validation. We will then take another feature engineering technique that is commonly used in the real-world machine learning projects. That is feature normalization. Towards the end, we will also talk about techniques to take your trained machine learning model to a full-blown data solution. In this regard, we will start with the model persistence and learn how to persist your trained model to the disk. Then we will build a machine learning API on top of the persisted trained model to create endpoints that can be used to make real-time predictions. From the tools perspective, we'll be using our tools from the previous module that are NumPy, pandas, and scikit-learn. However, in this module, you will learn a couple of new Python libraries. First one is pickle that is used for model persistence, and second one is Flask that we will use to create the machine learning API. So now we have set up our expectations for this module, let's jump right into the machine learning model tuning phase with a discussion on underfitting versus overfitting in the next clip.

Underfitting vs. Overfitting

If you are working on any machine learning project, you will hear the terms underfitting and overfitting several times. So let's have a brief discussion on underfitting and overfitting and why should you care about it. Let's again take an example. Suppose you are working on a binary classification problem. So in the training data, we have two classes represented by green and orange dots, and you need to build a classifier that separates both these classes. Now suppose you come up with this simple model. So if you look at the decision boundary that we got from the simple model, it is not able to separate green dots with orange dots properly, and we have so

many misclassifications. So in this case, your model performance is very poor, and your model is not able to learn the pattern even in the training data. This scenario is known as underfitting. And obviously, this is not what we want to accomplish through machine learning. So, you can use a more complex model to avoid underfitting. However, when you choose a complex model, you can run into another situation. Let's look at another case where you build a model in such a way that it created a very complex decision boundary, but able to separate all of the points. From the accuracy perspective, you may say that this is a great model, as you have a very high accuracy on training data, and you are able to separate all green dots from orange dots. But this is again not an ideal scenario because in this case, the model has simply memorized the training dataset and adjusted itself too much according to the training data. But such models often don't work well on unseen cases and thus has poor generalization. But as we discussed in the previous module, generalization is very important for the machine learning model because we want to use the machine learning model on unseen cases. This scenario is also popularly known as overfitting. So ideally, you should aim to avoid overfitting as well. So your accuracy on the training data may not be 100%, but your model performance on unknown or unseen cases will be better. So what are the ways to avoid overfitting? Well, one such technique is called as regularization that we will cover briefly in the next clip.

Regularization

In simple terms, you can think regularization as a technique that helps to tackle the model overfitting scenario by reducing the complexity of the model in order to make it more balanced so that it is neither underfit nor overfit. Now let's take a deeper dive in the regularization in the context of logistic regression model that we had built in the previous module. So if you can recall, in the previous module, we started by creating a model object. Then, we trained our model by passing the training data to the. fit function on the model object. And once the model got trained, we evaluated its performance using the. score function and by passing the test data to it. We also extracted the coefficient, or model parameters, using the. coef_ property of the model object. Well, all these steps are fine, but if you look closely in the first step once again where for creating the logistic regression model, we have only specified the random_state. But actually, the LogisticRegression function has a bunch of important parameters that we didn't even look into. We used all of the default values for the rest of the parameters. Let's look at one particular parameter C. This is a very important parameter as this is the regularization parameter for the LogisticRegression. So if you set the C value to be too large or too high, then it will increase the model complexity a lot, and the model will be overfit. While on the other side, if you set the value

of C to be too small, then the model complexity will be very low, and the model will be underfit. So you can think C value as a knob, and based on how you move this knob, your model can move into the overfit and underfit zones. Well, other than C, if you look into the LogisticRegression function, we have other parameters as well for which we have taken the default values. Let's look at another very important parameter, or I should say another knob. That is penalty. Penalty is another term for regularization. There are different types of regularization techniques, and we will not be going into the details of these techniques. But for now, it is sufficient to know that it is another key knob that you can change to fine-tune your model. The common penalties, or regularization types, are L1 and L2. For the logistic regression model in scikit-learn, L2 is used as the default one. I would again highly recommend that you explore different regularization techniques. However, one thing I want to point out that in the logistic regression, we have less number of knobs, but for more advanced machine learning algorithms, the number of knobs can be even more. In fact, these knobs are often referred as hyperparameters, and it is up to the model creator to choose the proper settings for these hyperparameters. So the next obvious question is how do you optimize the hyperparameters? How do you come up with the best possible combination? Well, there are several standard approaches for optimizing the hyperparameters, and all of them come under the umbrella of hyperparameter optimization. So now let's have a quick look at one of the most commonly used hyperparameter optimization techniques, that is grid search, in the next clip.

Hyperparameter Optimization: GridSearch

Grid search is possibly the most commonly used technique for hyperparameter optimization. Let's look at one example to understand how it works. Let's say you have a model where you have one hyperparameter A that you want to optimize. So you can create a grid. Let's say for hyperparameter A you want to try out 3 values, a1, a2, and a3. So, you want to build three models for each possible value of A and then select the model for which you are getting the best model performance. If you have two hyperparameters, A and B, then your grid may look like a two-dimensional array. And you want to try out different combinations of values of A and B. In fact, you can extend this technique to more number of hyperparameters as well. So in simple terms, first create a grid with different combinations of hyperparameter values, then evaluate the model performance for each of the combinations, and finally, select the best combination that leads to best model performance. But the next question can come how to assess model performance itself? Well, we have already seen one technique for model evaluation. That is train test split in the previous module where you split your data into two parts and use the training portion to train the

model and use the test portion to evaluate the model. However, for hyperparameter tuning, we typically follow a slightly different approach that is known as cross-validation. So let's talk about cross-validation in the next clip.

Crossvalidation

In the cross-validation setup, instead of two parts, we split the data into three parts, training, test, and cross-validation. Now, you pass the training set to the model and apply the training process to get the trained model, and then we evaluate the performance of the trained model on a cross-validation dataset. So for each of the hyperparameter combinations, you build different models, and for each model, you evaluate performance using the cross-validation set. So now, for each of the hyperparameter combinations, you can get one score, and then you will end up with a list of scores. Then, among all of those scores, you find out which one is the best. And once you have the best model for which you have the best model score, then you use the test set only once to evaluate the final score. In summary, you use the cross-validation set multiple times during the training process itself to finalize the best model, and you use the test set only once to test the model performance to get the final score. Well I know this whole process may seem very complicated to you, so I would highly recommend that you go through this concept several times until you develop the correct intuition. Well, there are several ways to partition the data for cross-validation. We have already seen one technique here where the cross-validation set is fixed and you use the same set while testing the different hyperparameter combinations. But there is an advanced technique to partition data for cross-validation. That is known as k-fold cross-validation that we will discuss next.

K-Fold Crossvalidation

K-fold cross-validation is a very common and popular technique to fine-tune the machine learning models. Let's try to understand k-fold cross-validation using a simple 3-fold example. So essentially, what you do in 3-fold cross-validation is that first you start with your training dataset. Then you divide the dataset into three portions. Then you use one portion for testing your model performance and the rest two portions for training the model itself. So based on this setup, you calculate the model performance score. Let's say you get the first score denoted as a score 1 here. Then, you take a different partition of the dataset and repeat the same thing and get the second score. And again, another partition of the dataset and get the third score. And once you have all three scores, you can simply summarize those scores by calculating the mean to get the mean

score. You can also calculate the standard deviation to assess if the scores are fluctuating too much by a different choice of partitioning. So this was about k-fold cross-validation. Well, now we have developed a basic understanding of grid search and cross-validation techniques, let's see how you can perform hyperparameter optimization in our Titanic disaster problem in the next clip.

Demo: Hyperparameter Optimization Using GridSearchCV

In this demo, you will learn to use scikit-learn GridSearchCV function to perform hyperparameter optimization. Here we are in the Jupyter Notebook, and we are continuing our journey from where we have left in the last module. So first we are creating our basic logistic regression model. So let's execute the cell. Then, we are importing the GridSearchCV function that we will use for hyperparameter optimization. Next, we are creating a parameter dictionary where for each of the hyperparameters we have listed out the values that we want to try out during the optimization process. So for C, we have specified a few values ranging from 1 to 1000, and for the penalty parameter, we want to try L1 and L2. Once we have our parameters set, then we are creating our grid search object, clf, using the GridSearchCV function. In this function, we need to specify the base model itself. Then the grid parameters are set using the param_grid attribute. We have also set the cv value to 3. This will perform the 3-fold cross-validation. Now let's execute the cell. So now we have our grid search setup done, let's pass the training data to train different models with different hyperparameter combinations. Now let's see what is the best combination we got. So you can use the best_params property to get the best settings. So we got 1 for C and 11 for the penalty. Now let's look at the best score. So you can use the best_score_ property. And here we have the best score. It may not be a significant jump because we are already reaching the maximum limit of the logistic regression model. But for more advanced algorithms, you will definitely get performance improvement. Now let's get the final score on the test dataset. So in the next line, we are using the clf object, and we are passing the X_test and y_test to calculate the test score. Internally, it will use the best model to give the final performance output. And it is also around 0. 83, or 83%. Now let's create our next submission file for Kaggle and see if we are moving ahead in the leadership board on Kaggle.

Demo: Making Third Kaggle Submission

In this demo, you will make your third Kaggle submission. Here we are in the Jupyter Notebook, and here again we are using the get_submission_file function that we had created in the last

module. However, this time we are passing our clf model object and the new submission filename. So let's execute the cell. So we have our third submission file ready. Let's submit it on the Kaggle platform. Let's upload the submission file. Here's the third file. We have also added some comments. Let's make the submission. And great, we have our third Kaggle submission, and our rank in the leaderboard has also improved slightly. Well again, don't pay too much attention on the Kaggle leaderboard. Our focus in this course is to learn the right approach for building predictive models and to go step by step. So now we know how to perform hyperparameter tuning, let's discuss another useful machine learning technique that is feature normalization and standardization that may also help to boost your model performance.

Feature Normalization and Standardization

Many machine learning algorithms can work better if you provide features on the same scale. For a logistic regression model that we are building in this course, it may not make a significant impact, but if you are using more advanced machine learning algorithms, such as neural networks, then it is always suggested to perform feature normalization before fitting the data to the model. Now let's try to understand feature normalization using a very simple example. Let's talk about some of the features in our Titanic dataset. So in the Titanic dataset, Age ranged from 0. 4 to 80, passenger Fare ranged from 0 to 512, and FamilySize can go from 1 to 11. So here you can see different features are at different scales. Ideally, we should scale all of the features on the same scale. Well, most commonly used scale is 0 to 1 scale, so you should map your input features on the scale of 0 to 1. Minus 1 to 1 is another very commonly used scale. However, you don't want to use -1 to 1 scale for features where negative values don't make much sense. For example, for Age, Fare, and FamilySize, negative values won't make much sense, so you should better choose the 0 to 1 scale. But you should know that there are other options as well. Many of times, we also utilize feature standardization, especially when the machine learning algorithm not only bothered about the range, but also about the distribution. Let's take our previous example. Suppose the distribution of each of these features are very different with different mean values and standard deviations. Then what we need to do is to standardize the features in such a way that all of the features have 0. 0 mean and unit variance. Such a standardization can boost model performance, especially for those models that rely heavily on data distributions. So now you have a fair idea about feature normalization and standardization, let's see how to perform these on the real-world dataset using Python.

Demo: Feature Normalization and Standardization Using Scikit-Learn

In this demo, you will learn to use scikit-learn to perform feature normalization and standardization. Here we are in the Jupyter Notebook. Well, sklearn provides a couple of very useful functions to accomplish the goal of feature normalization and standardization. You can use the MinMaxScaler function for normalization and StandardScaler function for standardization. So let's first import these functions. Once the functions are imported, the next step is to create the scaler object itself. And once the scaler object is created, we can pass the data to the. fit_transform function. The. fit_transform function is a combination of two steps. First step is to use the X_train to fit the scaler, and the second step is to transform the X_train to create the scaled output that we are also saving here in the variable X_train_scaled. So let's execute the cell. So now we have the scaled output. So it should have been scaled to 0 to 1 level. That is the default for the MinMaxScaler function. So in the next cell, we are extracting one column and printing its minimum and maximum value to check the impact of scaler. And here we have 0 and 1 as expected. Now let's also scale our test data because if you will train your model on the normalized data, then the model will expect that you pass the test data also in the normalized form. Similarly for standardization, we can use the StandardScaler method, and the steps are exactly the same, just that this time we are using the StandardScaler function that will standardize all of the features with 0 mean and unit variance as discussed in the previous clip. So now we know how to standardize features, let's use both hyperparameter optimization and a standardization to build our next version of model and see if we are getting any performance improvement. So in the next cell, we are creating our model using the grid search technique. These are the same lines of code that we have seen already. However, this time we are passing the standardized training data to the model. So let's execute the cell. So now we have our model, let's print the best score. Let's also print the score like we did for the previous models. And it is more or less the same as the previous version. Well, for the logistic regression, feature standardization may not add too much value, and there are technical reasons for such behavior that is beyond the scope of this course. But if you want to read more, you can use this link on Stack Exchange that will give you some idea. Again, the whole objective of walking you through the feature normalization step is that it is one of the important steps that you may have to take in your predictive modeling journey. So you should definitely try this to check if you are getting performance improvement. In fact, the whole predictive modeling is an iterative approach where you try different things. So one approach or one model may work or may not work, so you need to keep trying different things. So now suppose you have built your final model, and you and your stakeholders are happy about its performance. Now they want to use the model in the real time

to make predictions. So how can you take your model that you have built by running through different cells of a Jupyter Notebook to a real-world environment? Well, in the rest of the module, we will cover exactly the same, starting with the concept of model persistence in the next clip.

Model Persistence

Model persistence is a technique where you take your trained model and write or persist it to the disk. And once you have your model saved on the disk, you can use it whenever you want. So simply read and load the file and get the trained model back that you can use for making predictions. This is a very powerful technique because now you don't have to train the model every time by executing different cells of the Jupyter Notebook in order to use the trained model. You can persist your trained model once, and then you can use it later. You can also share your training model with others without sharing the training data and all of the steps to train the model. Not only this, you can use the persisted model to create a machine learning API layer on top of it. In fact, we will cover the API development process towards the end of this module. Well for now, let's quickly jump into the demo to see how you can persist your model.

Demo: Model Persistence Using Pickle

In this demo, you will learn to use pickle library to persist your model. Here we are in the Jupyter Notebook, and we will be using the pickle library to save the trained model on the disk. The process itself is very simple. First we import the pickle library that is available in the Anaconda distribution. If you are not using the Anaconda distribution or you don't have the pickle library installed on your local machine, then you can use the pip installer to install it first. I'm assuming that you have the pickle library and you can import the library. So let's import this. Once the library is imported, we can create the file path where we want to place our persisted models. Here we are putting all of our models in the models folder as per the cookie-cutter data science template that we discussed in the starting phase of this course. Also, if you can notice, I'm using the. pkl extension for the pickle files. Well, this is not mandatory, but it is a common practice to use the. pkl file extension for persisted pickle files. Not only this, if you can see, we are creating another file to persist the scaler object as well. Remember, in the last demo, we created a StandardScaler object for feature standardization, and you need to save the information stored in the scaler object as well so that later on you can standardize the inputs. Next, we are opening both of these files in the write mode. Here we are specifying the wb so that we can write in the binary format. Once we have the files opened, we can write to these files. So we can simply use

the. dump function in the pickle library to accomplish this task. So we are writing both the model, as well as the scaler objects. And finally, we are closing the files. So now we should have a couple of pickle files in the models folder, one for the model and another for the scaler. And here we have two pickle files in the models folder. So coming back to the Jupyter Notebook, let's test whether these pickle files are useful or not. So, in order to test that, we can load the persisted files in the read mode. Here, we can use the. load function to read the pickle files and load into the memory. We are also assigning the loaded model to a variable clf_loaded, and loaded scaler object to a variable scaler_loaded. And finally, we are closing the files. So let's execute the cell. So now we should have the classifier object back. Let's confirm it by printing the classifier itself. And it is the same classifier that we created in the last demo. Now let's print the scaler object as well. And here we have the StandardScaler object. Now let's use this classifier to get the score on the test dataset like we did previously. However, this time we will use the loaded scaler and the model object. So first we are using the loaded scaler object to transform the test to DataFrame. Then we are passing the scaled test data to calculate the score. And we got the same model score like we got in the previous demo. So as you can see, pickle library makes it really easy to persist the model or any other Python object to the disk. Well, now you have a fair understanding of model persistence, and we can end our data science journey here itself, but let's take one final stretch, and I promise you that it will be really worth it. In the next clip, we will talk about API development that will help to expose your model in such a way that any client or any application can make use of your model. It may look a bit tricky at first, but it will really add a new dimension in your data science journey.

Machine Learning API Development

Well, if you recall, we have covered some background about the APIs in the data extraction module and talked specifically about the REST APIs. Let's do a quick recap here. So in the REST API, the client can make a request over the HTTP protocol, and the server can send back the HTTP response. Also, in the REST world, you can use the common HTTP verbs such as GET to get some information and POST to send some data and get the response. Moreover, in the data extraction module, we used an existing API and invoked the API using the Python Requests library to get the data. However, this time we will create our own API, and then we will use the Requests library to invoke the API. So how will a machine learning API look like? Well, the job of a machine learning API is to return model predictions when the input data is given to it. So the client will send the input data wrapped in an HTTP request object for which the prediction has to be made, and the API hosted on the server will extract the input data from the request object,

and then it will process the data, if required. Then the processed data will be passed to the machine learning model. Remember in the previous clip we learned how to persist a model. We will use the persisted model here to make predictions. So once we have the predictions from the model, it will be sent back to the client wrapped in an HTTP response object. So now you have got some idea of what is a machine learning API and how it will work, let's create the machine learning API using another very popular Python library known as Flask. And we will again use the Requests library to invoke the API that we will create using Flask. However, as the API development may be a totally new venue for many of you, so let's build a Hello World API quickly to give you an idea what the Flask skeleton code look like. Then, we will change this skeleton to build a machine learning API. So let's build a Hello World API using Flask in the next clip.

Demo: Hello World API Using Flask

In this demo, you will learn to use Flask to develop a very simple Hello World API. Here we are in a new Jupyter Notebook, and here we are creating a Python script, hello_world_api. py using the Jupyter %%writefile magic function. Also, we are placing this new script into the models subfolder inside the src folder. So let's create the path first. So in the next cell, we are writing our code to create the Flask application. So first we are importing a couple of functions from the Flask library. Then, we are creating the Flask app using the Flask function. And once the app is created, you can define your API function. Here, we are using the Python decorator to create the API route. We are creating one API endpoint here, and we have given it a name /api. We have also mentioned that it will be used for the POST request, which means it will take some input, will process it, and will return back some response. Next, we are writing our Hello World function itself that will be called internally once the API is invoked. The purpose of this API is very simple. You pass one name, and the name will be appended after the word hello, and it will be returned. And as we will be sending the JSON object as the input, we are using the. get_json method to extract the input. We are also saving the input to the data variable. Next, we can extract the JSON object from the data variable. Finally, we are appending the name after the word hello and then returning it back. Towards the end, we have the main block. That is the entry point of the script. Here, we are saying that run the Flask application on port 10001. We have picked up this port just to avoid any port conflict. You can pick any of your available ports here. We have also set the debug property to True. So if you have set the debug to True and you encounter certain issues in the API call, then you will get the detailed stack trace that you can use to identify possible reasons. So in the development scenario, it is always useful to set the debug property to True. However, in the production setup, you may want to set this property to False. Now let's execute the cell to create

the script. So now we have our script ready, let's run the Flask application. Let's open a terminal where we have the script. So we are currently in the models folder. Let's look at the content of this folder. And we have our hello_world_api script here. So let's execute this script using Python hello_world_api. So now we have our Flask application up and running. In the development environment, we are hosting the API on our local machine, but you can load the same code on any cloud infrastructure like AWS, or Google Cloud, or Azure, and you will have your API hosted on the cloud. Sounds cool, right? And I would highly encourage you to explore the cloud deployment step. So coming back to the Jupyter Notebook, let's invoke the API that is running in the background using the Python Requests library. So first we are importing the Requests library. We are also importing the JSON library that will be used to convert the input to a JSON string. Next, we have the URL of our API. So we have the local host IP and 10001 as the port number, and finally, the API endpoint. Next, we are creating our input data that we will post to the API. We are creating a JSON object here with name as key and abhi as the value. Then we are using the json. dumps function to convert the object to a JSON formatted string. Finally, we are using the. post function to post the data to the API and get the response back. We are also storing the response into the variable r. So let's import the libraries and make the API call. So now we have our response, let's get the response text using the. text property. And here we have our response hello abhi. So as you can see with a few lines of code, we are able to create our own API. Now, let's use this boilerplate code and create the machine learning API in the next clip.

Demo: Machine Learning API Using Flask

In this clip, we will create the machine learning API using Flask. Here we are in the Jupyter Notebook, and we are creating another Python script, machine_learning_api. py. And we have placed this script into the same models subfolder inside the src folder. So let's create the path. Then, in the next cell, we have the Flask code to create the machine learning API. High-level steps are exactly the same like the previous clip, but we have modified the script from the machine learning prediction aspect. So here we are importing a bunch of libraries, such as pandas, NumPy, pickle, and OS, as we will use them in the script below. Then we have set up the Flask app like we did in the previous clip. Then, we are loading our model and scaler from the pickle files. This is exactly the same as what we did in the model persistence demo. However, this time we have slightly modified the model part as per the location of the machine learning API script in the Titanic project structure. Also, this time we have loaded the pickle file in one go. Then, we have all of our columns in an order that the machine learning model can expect. Next, we have our actual function, make_prediction, that will be executed when the machine learning API will be invoked.

Inside the make_prediction function, we are first reading the JSON object, and then we are converting it to the JSON string. Then, we are creating a pandas DataFrame by reading the JSON formatted string. Then we are extracting all of the PassengerIDs from the DataFrame and saving it to the variable PassengerIds. We are also saving the actual Survived value in a variable. In the real-world scenario, you will not have access to actual result, but here I simply want to showcase how the API works and whether we are getting the same performance from the API or not. So you can think as a validation step. Then, we are extracting the required columns from the DataFrame based on the model requirement, and we are converting it to the matrix X. Then we are transforming the input matrix X using the scaler that we loaded using the pickle file. Finally, we can use the. predict function on the model object to make predictions. Then, we are creating our DataFrame as our final response where we are putting the PassengerIds, Actual survived values, and Predicted survived values. Again, to reiterate, you will not have access to actually results while doing prediction in the real-world setup. So coming back to our script, once we have the response DataFrame, we are using the. to_json method to return the JSON version of the DataFrame. Now let's execute the cell to create our script. So now we have our script ready, let's go to the terminal and start the Flask application like we did in the Hello World case. Let's stop the previous Flask application and run the machine learning API Flask application. So now we have our API up and running. Let's use it. Coming back to the Jupyter Notebook, first we are using the processed train data itself to test if our machine learning API is working fine or not. So, we are reading our processed training data file. Now let's pick five passengers who survived the Titanic disaster. And here are these five passengers. So we will pass the information about these passengers to the API, and the API should also return survived for these passengers. So let's test this. Here, we have created a helper function to make the API request. It is the same code that we looked at in the previous clip; however, this time we are extracting the JSON object from the response instead of the plain text. So let's execute the cell to create the function. Now let's use this helper function to make the API call. And here you can see we got the predicted value of 1 for all 5 passengers. So that ensures that our API is working fine. Now let's pass the complete training DataFrame to test the overall accuracy. So in the next cell, we are passing the complete train_df DataFrame to the API request helper function. Then we are converting the result to the JSON string and reading it using pandas to get the df_result DataFrame. We are also printing the top five rows of the result. Now let's see what is the accuracy we got. So in the next cell, we are using a simple NumPy trick for this. We are first comparing the actual results predicted that will result into Boolean values of true and false, and then we are taking the mean to get the overall accuracy. So let's execute the cell. We have not imported the NumPy library, so let's import the NumPy here. Now let's execute the cell. And we got the accuracy level of 0. 83. This is the same as what

we got in the model persistence demo. So now we know how to make a machine learning API. Well, you can further improve the API code. If you look closely, we are passing the processed data to the API to make the predictions. But if you can have the data processing logic, as well as the part of the API, then you can pass the raw data to the API rather than the processed one. I would highly encourage you to take this initial version of the API and come up with your own API that can take raw, unprocessed values as input and let the API do all the heavy lifting, such as first processing it and then passing the processed data to the model to get the final predictions. Also, please do share your approach in the discussion forum, and I'll be more than happy to guide you in case you are facing any issues. But overall, the goal of taking you through the API development process is that now your model can be integrated with any other system or can be a part of the full-blown data solution, and you can use it to make real-time predictions. So now we have reached to the end of this module.

Demo: Committing Changes to Git

However, before we end this module, let's commit all of our work to the Git versioning system as a good practice. So here we have our terminal on the Titanic root folder. Let's confirm the path using the pwd. Let's check the git log as well. So, so far, we have three commits that we did in the previous modules. Now let's use the git status to see the new changes. And here are the files that we have changed over the course of the last couple of modules. Now let's add these changes and commit them using git add dot. So we got some warnings for line endings, so we can safely ignore them. Now let's commit our changes. We have also added an appropriate commit message using the -m flag. Now let's check our git log for the final time. So we have all of our commits checked into the local Git versioning system. And if you are pushing your changes to any cloud repository such as GitHub, then don't forget to push the changes. So this wraps up our data science journey.

Summary

So in this module, you learned about some very important machine learning aspects related to model tuning starting with underfitting and overfitting. And then we looked into ways to avoid overfitting by performing hyperparameter tuning. We also used the scikit-learn GridSearchCV function to perform hyperparameter optimization to come up with the best possible settings of hyperparameters. We also discussed cross-validation and how it can be useful to avoid model overfitting scenarios. In the meanwhile, we also made our third Kaggle submission and got some

substantial jump in the ranking leaderboard. Then, we talked about techniques to rescale input features such as feature normalization and standardization. Many advanced machine learning algorithms work better if you have rescaled your inputs before fitting it to the model for the training process. So you should definitely include this step as a part of your machine learning project work. Then, we learned about model persistence to persist the trained model to the disk. Once we had our model persisted, we exposed our machine learning model as an API endpoint so that any client can invoke the API using a simple HTTP request. We also saw how Flask makes the whole API development process very easy so that you can focus more on the actual modeling work and let the Flask do all the heavy lifting for you. So now we have reached to the end of this course, and I'm really hopeful that you have learned the end-to-end data science project cycle. Even though we cannot cover all of the aspects of data science in one single course, the objective was to give you an idea about end-to-end cycle and how it looks like. Now it's totally up to you to take this journey to the next level.

Where to Go from Here?

So where can you go from here? The first thing that you can try is to use the learnings of this course and apply to different types of datasets. We have already seen some of the resources from where you can get publically available datasets. So practice on different datasets, try to explore, and work with them. The more you will practice, the better you will be as a data scientist. The second aspect that you can focus is on the machine learning algorithms itself. We only looked at one algorithm in this course, that is logistic regression, because the focus of this course was more on the end-to-end data science lifecycle, but there are several advanced machine learning algorithms out there that can provide better model performance, so you can try out random forest, support vector machines, or the fascinating world of neural networks. So try to explore them. And in fact, you can leverage some of the open source Python libraries like scikit-learn for your exploration. Then, you can also focus on productionizing your machine learning models. You can learn about creating pipelines, developing APIs, and deploying the machine learning models in the production environment. Also, try to keep yourself updated as the data science is a rapidly changing field, so it is very important that you keep track of the current work. So for that, you can involve yourself in the data science community by participating on different blogs or channels. You can also start one for yourself if you are really interested in it. You can also participate in data science and machine learning competitions as well, as they will help you to be on your toes, and you can even win name and fame in the data science world. So as parting thoughts, I'll just say that your data science journey has just started, and the sky is the limit. Also, keep sharing your

data science journey stories in the discussion board so that all of us can learn from each other. Thanks again for watching this course. I can't wait to know more about your data science journey. Have a great time. Bye-bye.

Course author

Abhishek Kumar

Abhishek Kumar is a data science consultant, author, and Google Developers Expert (GDE) in machine learning. He holds a master's degree from the University of California, Berkeley, and has been...

Course info

Level	Beginner
Rating	★★★★½ (207)
My rating	★★★★★
Duration	6h 25m
Released	28 Dec 2017

Share course