

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 01
Why Python for Data Science?

Welcome to this course on Python for Data Science. This is a 4-week course; we are going to teach you some very basic programming aspects in python. And since this is a course that is geared towards data science, towards the end of the course, based on what has been taught in the course, we will also show you two different case studies; one is what we call as a function approximation case study, another one a classification case study.

And then tell you how to solve those case studies using the programming platform that you have learned. So, in this first introductory lecture, I am just going to talk about why we are looking at python for data science.

(Refer Slide Time: 01:10)

What is Data Science?

- **Data Science** is the science of analyzing **raw data** using statistics and machine learning techniques with the purpose of drawing insights from the data
- **Data Science** is used in many industries to allow them to make better business decisions, and in the sciences to test models or theories
- This requires a process of inspecting, cleaning, transforming, modeling, analyzing, and interpreting raw data



Python for Data Science

So, to look at that first, we are going to look at what data science is. This is something that you would have seen in other videos of courses in the NPTEL and other places. Data science is basically the science of analyzing raw data and deriving insights from this data. And you could use multiple techniques to derive insights; you could use simple statistical techniques to derive insights, you could use more complicated and more sophisticated machine learning techniques to derive insights and so on.

Nonetheless, the key focus of data science is actually deriving these insights using whatever techniques that you want to use. Now there is a lot of excitement about data science, and this excitement comes because it's been shown that you can get very valuable insights, from large data and you can get insights about how different variables change together, how one variable affects another variable and so on with large data which is not very easy to simply see by very simple computation.

So, you need to invest some time and energy into understanding how you could look at this data and derive these insights from data. And from a utilitarian viewpoint, if you look at data science in industries, if you do proper data science, it allows these industries to make better decisions. These decisions could be in multiple fields; for example, companies could make better purchasing decisions, better hiring decisions, better decisions in terms of how to operate their processes, and so on.

So, when we talk about decisions, the decisions could be across multiple verticals in an industry. And data science is not only useful from an industrial perspective, but it is also useful in actual science as themselves. So, where you look at lots of data to model your system or test your hypotheses or theories about systems and so on. So, when we talk about data science, we start by assuming that we have a large amount of data for the problem of interest. And we are going to basically look at this data we are going to inspect the data; we are going to clean and curate the data then we will do some transformation of the data modeling and so on before we can derive insights that are valuable to the organization or to test a theory and so on.

(Refer Slide Time: 03:47)

The image shows a video slide with a dark blue header bar at the top. Below the header, the title "Data perspective" is displayed in a white font. Underneath the title, there is a bulleted list of five items: "Read data", "Data processing and cleaning", "Summarizing data", "Visualization", and "Deriving insights from data". To the right of the list, there is a video player window showing a man in a striped shirt speaking. The video player has a dark blue footer bar with the text "Python for Data Science" in white. The overall background of the slide is light gray.

Now, coming to a more practical view of what we do once we have data. I have these four bullet points, which roughly tell you, supposing you were solving a data science problem, what are the steps you will do? So, you will start with just having data someone gives you data; and you are trying to derive insights from this data. So, the very first step is really to bring this data into your system. So, you have to read the data. So, the data comes into this programming platform so that you can use this data. Now data could be in multiple formats so you could have data in a simple excel sheet or some other format.

So, we will teach you how to pull data into your programming platform from multiple data formats. So, that is the first step, really. If you think about how you are going to solve a problem, these steps would be first to simply read the data. And then, once you read the data many times, you have to do some processing with this data; you could have data that is not correct. For example, we all know that if you have your mobile numbers, there are 10 numbers in a mobile number, and if there is a column of mobile numbers and then say there is one row where there are just five numbers, then you know there is something wrong. So, this is a very simple check I am talking about in real data processing; this gets much more complicated.

So, once you bring the data in when you try to process this data, you are going to get errors such as this. So, how do you remove such errors? How do you clean the data? It is one

activity that usually precedes doing you more useful stuff with the data. This is not the only issue that we look at there could be data that is missing.

So, for example, there is a variable for which you get a value in multiple situations, but in some situations, the value is missing. So, what do you do with this data do you throw the record away? Or you do something to fill that data and so on. So, these are all data processing cleaning steps. So, in this course, we will tell you the tools that are available in python so that you can do this data processing cleaning and so on.

Now what you have done at this point is you have been able to get the data into the system, you have been able to process and clean the data and get to a certain data file or data structure that is complete so that you think you can work with this data set at which point what you will do is you will try to summarize this data. And usually, summarization of this data, a very simple technique would be very very simple statistical measures that you will compute; you could, for example, compute a median, mode, mean of a particular column.

So, those are simple ideas or summarizing the data you could compute variance and so on. So, we are going to teach you how to use these notions of statistical quantities that you can use to summarize the data. Once you summarize the data, then another activity that is usually taken up is what is called visualization. So, visualization means you look at this data and more pictorially to get insights about the data before you bring in heavy-duty algorithms to bear on this data. And this is a creative aspect of data science; the same data could be visualized by multiple people in multiple ways. And some visualizations are not only eye-catching but are also much more informative than other types of visualization.

So, this notion of plotting this data so that some of the attributes or aspects of the data are made apparent is this notion of visualization. And there are tools in python that will teach you in terms of how you visualize this data. So, at this point, you have taken the data, you have cleaned the data, got a set of data points or data structure that you can work with, you have done some basic summary of this data that gives you some insights. You also looked at it more visually, and you have got some more insights, but when you have a large amount of big data, the last step is really deriving those insights which are not readily apparent either through visualization or through a simple summary of data.

So, how do we then go and look at more sophisticated analytics or analysis of data so that these insights come out? And that is where machine learning comes, and as a part of this

course when you see the progress of this course, you will notice that you will go through all of this so that you are ready to look at data science problems in a structured format and then use python as a tool to solve some of these problems.

(Refer Slide Time: 08:57)

The slide has a dark blue header bar and a light gray footer bar. The title 'Data science using Python' is centered in the header. The footer bar contains the text 'Python for Data Science'. On the right side of the slide, there is a small video frame showing a man in a striped shirt sitting at a desk and gesturing with his hands while speaking.

Data science using Python

- Python libraries provide key feature sets which are essential for data science
- Data manipulation and pre-processing
 - Python's 'pandas' library offers a variety of functions for data wrangling and manipulation
- Data summary
- Visualization
 - Plotting libraries like 'matplotlib' and 'seaborn' aid in condensing statistical information and help in identifying trends and relationships
- Machine learning libraries like 'sci-kit learn' offer a bouquet of machine learning algorithms

Now, why python for doing all of this? The number one reason is that there are these python libraries, which are already geared towards doing many of the things that we talked about so that it becomes easy for one to program, and very quickly, you can get some interesting outcomes out of whatever we are trying to do.

So, there are, as we talked about in the previous slide, you need to do data manipulation and pre-processing. There are lots of functions libraries in python where you can do data wrangling manipulation and so on. From a data summary viewpoint, there are many of these statistical calculations that you want to do are already pre-programmed, and you have to simply invoke them with your data to be able to show data summary. The next step we talked about visualization, there are libraries in python, which can be used to do the visualization.

And finally, for the more sophisticated analysis that we talked about all kinds of machine learning algorithms are already pre-coded available as libraries in python. So, again once you understand some bit about these functions and once you get comfortable working in python, then applying certain machine learning algorithms for these problems becomes trivial. So, you simply call these libraries and then run these algorithms.

(Refer Slide Time: 10:29)

Advantages of Python

- Provides good ecosystem of libraries that are robust and varied
- Tight knit integration with big data frameworks like Hadoop, Spark etc
- Supports both object oriented and functional programming paradigms
- Python is reasonably fast to prototype
- Provides support for reading files from local, databases and cloud



Python for Data Science

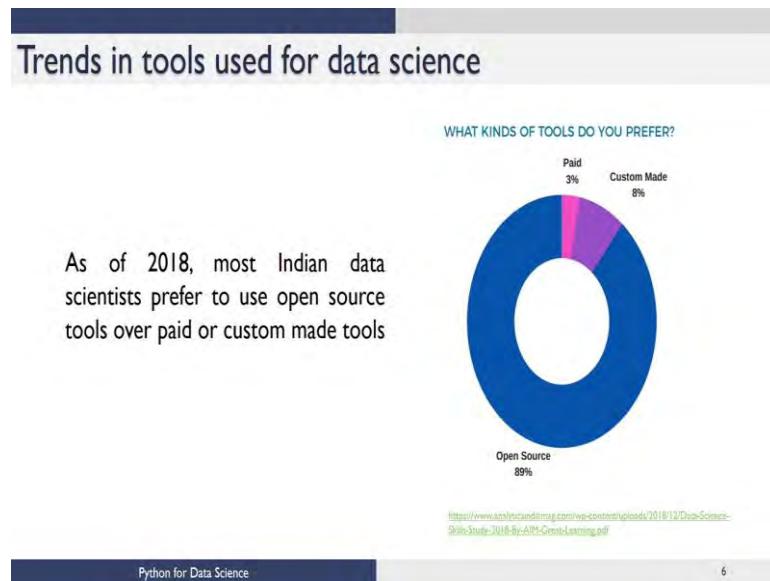
In the previous slide, we talked about the flow process for how I get the data in clean it and all the way up to insights, and then parallelly, we said why python makes it easy for us to do all of this. If you go back if you go forward a little more and then ask in terms of the other advantages of python, which are little more than just very simple data science activities. Python provides you several libraries, and it's continuously improved so, anytime there is a new algorithm that is coming into the set of libraries. So, in that sense, it's very varied, and there is also a good user community.

So, if there are some issues with new libraries and so on and those are fixed so that you get a robust library to work with and we talk about data, and data can be of different scale. So, the examples that you will see in this course are data of reasonably small size, but in real-life problems, you are going to look at data that is much larger, which we call big data. So, python has an ability to integrate with big data frameworks like Hadoop spark and so on.

And python also allows you to do more sophisticated programming object-oriented programming and functional programming. Python, with all of these sophisticated tools and abilities, is still reasonably a simple language to learn its reasonably fast to prototype. And it also gives you the ability to work with data which is in your local machine or in a cloud and so on. So, these are all things that one looks for when one looks at a programming platform that is capable of solving problems in real life right.

So, these are real problems that you can solve; these are not only toy examples, but real applications that you can build data science applications that you can build with python.

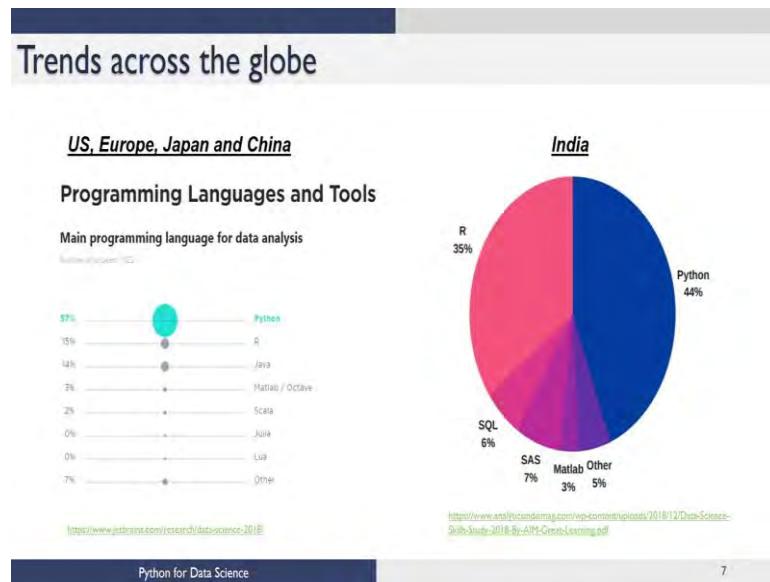
(Refer Slide Time: 12:49)



And just as another pointer in terms of why we believe that python is something that a lot of our students and professionals in India should learn. As you know, there are tools which are paid tools for machine learning with all of these libraries, and so on.

And there are also open-source tools and in India, based on a survey, most people, of course, prefer open-source tools for a variety of reasons cause being one because its free to use. But also if it is just free to use, but it does not have a robust user community, then it's not really very useful; that is where python really scores in terms of a robust user community, which can help with people working in python. So, it is both open-source, and there is a robust user community, both of which are advantageous for python.

(Refer Slide Time: 13:48).



And if you think of other competing languages for machine learning, if you look at this chart in India, about 44 percent of the people who were surveyed said they use python, or they prefer python. And of course, a close second is R. In fact, R was much more preferred a few years back, but over the last few years in India, a python is starting to become the programming platform of choice. So, in that sense, its a good language to learn because of the opportunities for jobs and so on or a lot more when you are comfortable with python as a language.

So, with this, I will stop this brief introduction on why python for data science. I hope I have given you an idea of the fact that while we are going to teach you Python as a programming language, please keep in mind that each module that we teach in this is actually geared towards data science. So, as we teach python, we will make the connections to how you will use some of the things that you see in data science; and all of this, we will culminate with these two case studies that will bring all of these ideas together. In terms of both are giving you an idea and an understanding of how the data science problem will be solved and also how it will be solved in python, which is a program of choice currently in India.

So, I hope this short four-week course helps you quickly get on to this programming platform. And then, learn data science, and then, you can enhance your skills with a much

more detailed understanding of both the programming language and data science techniques.

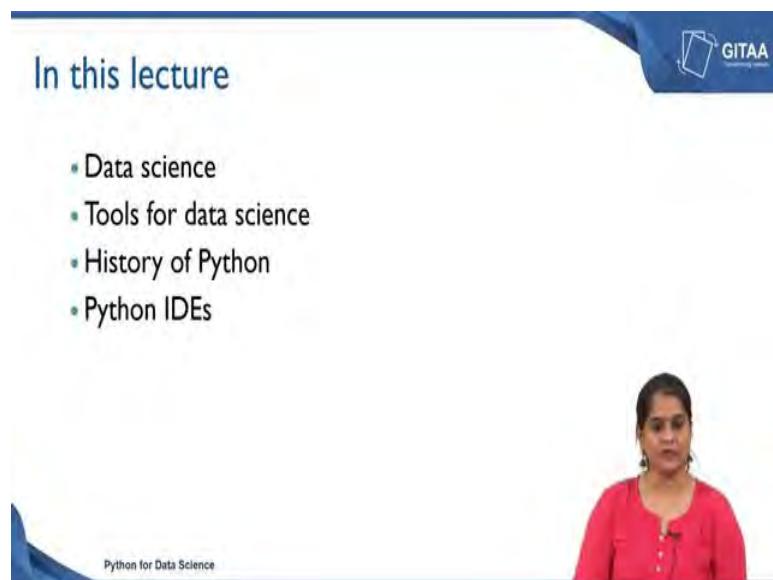
Thank you.

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 02
Introduction to Python

Welcome to the lecture of Introduction to Python.

(Refer Slide Time: 00:17)



The image shows a screenshot of a video lecture interface. At the top, there is a blue header bar with the text "In this lecture". Below the header, there is a list of topics in a bulleted format:

- Data science
- Tools for data science
- History of Python
- Python IDEs

At the bottom of the slide, there is a video player window showing a woman in a red shirt speaking. The video player has a blue border and a play button icon. The text "Python for Data Science" is visible at the bottom left of the video player area.

In this lecture we are going to see what data science is in brief, we are also going to look at what are the commonly used tools for data science, we will also look at the history of python and followed by that will look at what an IDE means.

(Refer Slide Time: 00:33)

The slide has a blue header bar with the word 'Introduction' in white. In the top right corner, there is a logo for 'GITAA' with a small icon. The main content area contains a bulleted list:

- We live in a world that's drowning in data
- Data is generated from various sources
 - Websites track every user's every click
 - Your smartphone is building up a record of your location
 - Sensors from electronic devices record real time information
 - E-commerce websites collect purchasing habits

At the bottom left of the slide, it says 'Python for Data Science'. On the right side of the slide, there is a video player showing a woman with dark hair, wearing a red top, speaking. The video player has a play button and some other controls.

So, we live in a world that is drowning with data, wherever you go wherever we are data is getting generated from various sources.

Now, when you browse through few websites, the websites track every users click and this forms a part of web analytics. The other instance of where data gets generated is when you use a smartphone; you are basically building up a record of your location. So, all these information go and sit somewhere and get collected in the form of data.

We also have sensors from electronic devices that record real time information and you also have e-commerce website that collect purchasing habits. So, whenever you log into any of these e-commerce sites, you will see some recommendations based on your previous purchase history or previous view history.

(Refer Slide Time: 01:19)

The slide has a blue header bar with the GITAA logo. The main title 'Data science' is in large blue font. Below it is a bulleted list of two points:

- Interdisciplinary field that brings together computer science, statistics and mathematics to extract useful insights from data
- Analyzing and generating insights from data aids in arriving at better business decisions

At the bottom left, it says 'Python for Data Science'. On the right side, there is a video frame showing a woman in a pink shirt speaking.

So, now let us see what data science is all about. So, it is an interdisciplinary field that brings together computer science, statistics and mathematics to get useful inferences and insights from a data. Now these insights are very crucial from a business perspective because, it will help you in making better business decisions.

(Refer Slide Time: 01:40)

The slide has a blue header bar with the GITAA logo. The main title 'Popular tools used in data science' is in large blue font. Below it is a bulleted list of three categories:

- Data pre-processing and analysis
 - Python, R, Microsoft Excel, SAS, SPSS
- Data exploration and visualization
 - Tableau, Qlikview, Microsoft Excel
- Parallel and distributed computing incase of big data
 - Apache Spark,Apache Hadoop

At the bottom left, it says 'Python for Data Science'. On the right side, there is a video frame showing a woman in a pink shirt speaking.

Now, currently there are many tools that are being used in data science; now these tools can be bucketed into 3 categories. The first category is where you are going to be looking at data preprocessing and analysis, now tools and software's that fall under this category

are python, R, MS Excel, SAS and SPSS. Now all these tools are required for you to preprocess and analyze the data. So, apart from data preprocessing and analysis, there is a fair share of effort that is given for data exploration and visualization and these are done even before you analyze your data.

Now, the commonly used data exploration and visualization tools are Tableau, Qlikview and of course, you always have your MS Excel. So, the next bucket that we are going to look into is when you have huge chunks of data, now when you are collecting data on a real time basis you are going to be collecting data over every second every minute. Now if you want to store all these data and preprocesses it the regular desktop or computing systems that you have might not be useful.

So, that is when you use parallel or distributed computing, where you distribute the work across different systems popular tools that are being used for big data Apache Spark and Apache Hadoop. So, in this course we are going to be mainly focusing on tools that are required for data preprocessing and analysis and in specific we are going to look into python.

(Refer Slide Time: 03:08)

Evolution of Python

- Python was developed by Guido van Rossum in the late eighties at the 'National Research Institute for Mathematics and Computer Science' at Netherlands
- Python Editions
 - Python 1.0-1991,
 - Python 2.0- 2000
 - Python 3.0 - 2008 (Python 3.7 – latest)



So, let us look at the evolution of python. So, python was developed by Guido van Rossum in the late eighties at the national research institute for mathematics and computer science and this institute is located at Netherlands.

So, there are different versions of python, the first version that it was released was in 1991; the second version was released in 2000 and the third version was released in 2008 with version 3.7 being the latest. So, let us look at the advantages of using python.

(Refer Slide Time: 03:41)

Advantages of using python

- Python has several features that make it well suited for data science
- Open source and community development
 - Developed under Open Source Initiative approved license making it free to use and distribute even commercially

Python for Data Science

So, python has features that make it well suited for data science. So, let us look at what these features are. So, the first and foremost feature of python is that it is an open source tool and python community provides immense support and development to its users. So, python was developed under the open source initiative approved license thereby making it free to use and distribute even if its for commercial purposes.

(Refer Slide Time: 04:05)

The slide has a blue header bar with the text 'Advantages of using python'. In the top right corner, there is a logo for 'GITAA' featuring a stylized book icon. Below the title, there are three green bullet points: 1. Syntax used is simple to understand and code. 2. Libraries designed for specific data science tasks. 3. Combines well with majority of the cloud platform service providers. At the bottom left, it says 'Python for Data Science'. On the right side, there is a video player interface showing a woman in a red shirt speaking, with a play button and a progress bar.

The next feature is that the syntax that python use fairly simple to understand and code and this breaks all kinds of programming barriers if you are going to switch to a newer programming language. So, the next important advantage of using python is that, the libraries which are contained in python get installed at the time of installation and these libraries are designed keeping in mind specific data science task and activities.

Python also integrates well with most of the cloud platform service providers; and this is a huge advantage if you are looking to use big data. So, if you are going to download python from the website and install it, you will see that most of the scripting is done in shell. So, there are applications that provide better graphical user interfaced for the end users and these are taken care by the integrated development environment.

(Refer Slide Time: 04:57)

Integrated development environment (IDE)

- Software application consisting of a cohesive unit of tools required for development
- Designed to simplify software development
- Utilities provided by IDEs include tools for managing, compiling, deploying and debugging software

Python for Data Science

GITAA Learning Series

So, now, let us see what an integrated development environment is, an IDE as how its abbreviated is a software application and it consists of tools which are required for development. All these tools are consolidated and brought together under one roof inside the application. IDEs are also designed to simplify the software development this is very useful because as an end user, if you are not a developer you might want all the tools available at a single click. Using an IDE will be very beneficial in that case also the features provided by IDEs include tools for managing, compiling, deploying and debugging a software. So, these also form the core features of any IDEs.

(Refer Slide Time: 05:44)

Features of IDE

- IDE should centralize three key tools that form the crux of software development
 - Source code editor
 - Compiler
 - Debugger
- Additional features
 - Syntax and error highlighting
 - Code completion
 - Version control

Python for Data Science

GITAA Learning Series

So, now let us look at what are the features of an IDE in depth. So, any IDE should consist of three important features; the first is the source code or text editor, the second is a compiler and the third is a debugger. Now all these three features form the crux of any software development.

The IDEs can also have additional features like syntax and error highlighting, code completion and version control.

(Refer Slide Time: 06:09)

Commonly used IDEs

- Spyder
- PyCharm
- Jupyter Notebook
- Atom

Python for Data Science

GITAA

So, let us see what are the commonly used IDEs for python, the most frequently used as spyder, PyCharm, Jupyter Notebook and Atom. And these are basically from the endpoint of the user, depending on what he or she is comfortable with.

(Refer Slide Time: 06:24)

- Supported across Linux, Mac OS X and Windows platforms
- Available as open source version
- Bundled with Anaconda distribution which comes with all Python libraries
- Developed for Python and specifically data science

So, now let us look at spyder, the spyder is an IDE and it supported across Linux, Mac and Windows platforms. It is also an open source software and it is bundled up with Anaconda distribution which comes up with all inbuilt python libraries.

So, if you want to work with spyder you do not have to install any of the libraries. So, all the necessary libraries are taken care by Anaconda. So, another important feature of spyder is that it was specifically developed for data science and it was developed in python and for python.

(Refer Slide Time: 06:57)

```
x = 1; y = 2; print((x,y) or (x,y))
In [1]: x=1; y=2; print((x,y) or (x,y))
Out[1]: (1, 2)
In [2]: xy
Out[2]: False
In [3]: xxy
Out[3]: False
In [4]: print(False or False)
Out[4]: False
In [5]: print((x,y) or (x,y))
Out[5]: (1, 2)
In [6]: print((x,y) or (x,y))
Out[6]: (1, 2)
```

So, this is how the interface of spyder looks, you have the scripting window and you have other console output here, you have a variable explorer here. All these features we are going to be looking at in the next few lectures to come.

(Refer Slide Time: 07:11)

The screenshot shows the Spyder Python IDE interface. At the top, there's a navigation bar with icons for file, edit, view, insert, cell, run, and help. Below the bar, the word "Spyder" is displayed in a large, blue, sans-serif font. To the right of "Spyder" is a logo for GITAA featuring a stylized book icon and the text "GITAA" above "GATEWAY TO KNOWLEDGE". A vertical toolbar on the left contains icons for file, edit, cell, run, and help. The main area is titled "Features include" and lists the following:

- Features include
 - Code editor with robust syntax and error highlighting
 - Code completion and navigation
 - Debugger
 - Integrated document
- Interface similar to MATLAB and RStudio

At the bottom of the slide, the text "Python for Data Science" is visible. Overlaid on the bottom right is a video player showing a woman in a pink shirt speaking. The video player has a play button and a progress bar.

The other features of spyder includes a code editor, with robust syntax error highlighting features; it also helps in code completion and navigation it consist of a debugger, it also consist of an integrated documents that can be viewed within the python interface on the web. Another advantage of using spyder is that it has a interface which is very similar to MATLAB and RStudio's. So, if you are a person who is already work with these two programming languages and are looking to switch to python, then the transition is also going to be seamless.

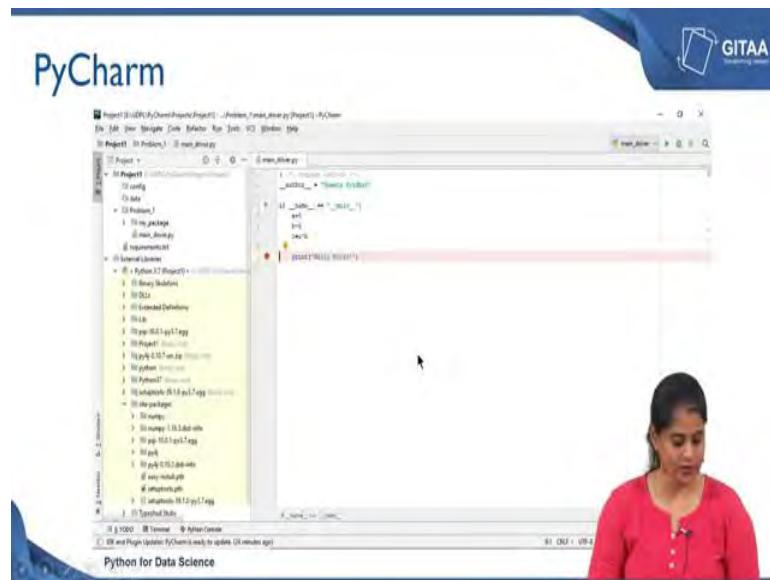
(Refer Slide Time: 07:44)



- Supported across Linux, Mac OS X and Windows platforms
- Available as community (free open source) and professional (paid) version
- Supports only Python
- Bundled with Anaconda distribution which comes with all Python libraries
 - Can also be installed separately

So, now let us look at the second IDE which is pyCharm. So, pyCharm is also supported across all OS systems which is Linux, Mac and windows. It has two versions to it one is the community version which is an open source software; the other is the professional version which is a paid software. So, pyCharm supports only python and it is bundled up and packaged with Anaconda distribution which comes with all the inbuilt python libraries. But; however, if you want to install pyCharm separately then that can also be done.

(Refer Slide Time: 08:14)



So, this is how the interface of pyCharm looks, you have a very very well define structure for naming your directories and you have the scripting window here.

(Refer Slide Time: 08:25)

The screenshot shows the PyCharm IDE interface. At the top, it says "PyCharm". On the right side, there is a logo for "GITAA" with a book icon. Below the title, there is a bulleted list under the heading "Features include":

- Features include
 - Code editor provides syntax and error highlighting
 - Code completion and navigation
 - Unit testing
 - Debugger
 - Version control

At the bottom left of the slide, it says "Python for Data Science".

So, let us look at some of the features that pyCharm consists of. The first is that it consists of a code editor which provides syntax and error highlighting; then it consists of a code completion and navigation feature it also consists of a unit testing tool which will help the compiler go through each and every line of the code. It also consists of a debugger and controls the versions.

(Refer Slide Time: 08:48)

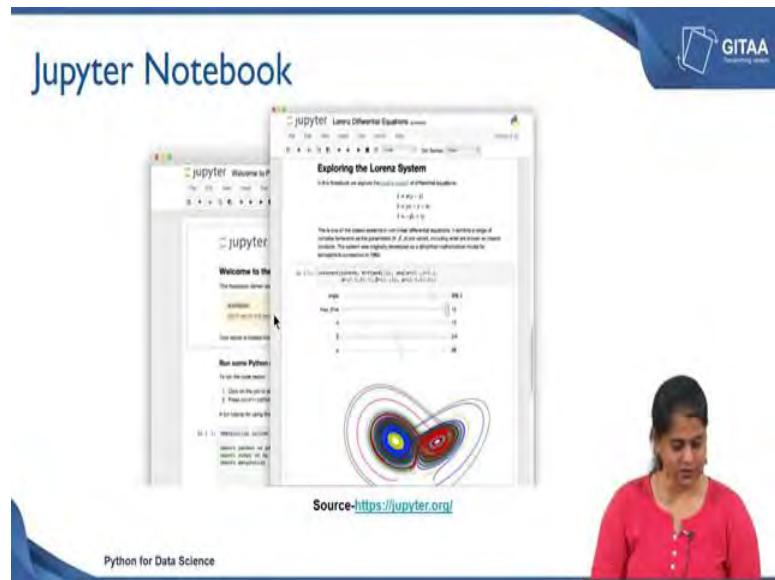
The screenshot shows the Jupyter Notebook interface. At the top, it says "Jupyter Notebook". On the right side, there is a logo for "GITAA" with a book icon. Below the title, there is a bulleted list:

- Web application that allows creation and manipulation of notebook documents called 'notebook'
- Supported across Linux, Mac OS X and Windows platforms
- Available as open source version

At the bottom left of the slide, it says "Python for Data Science".

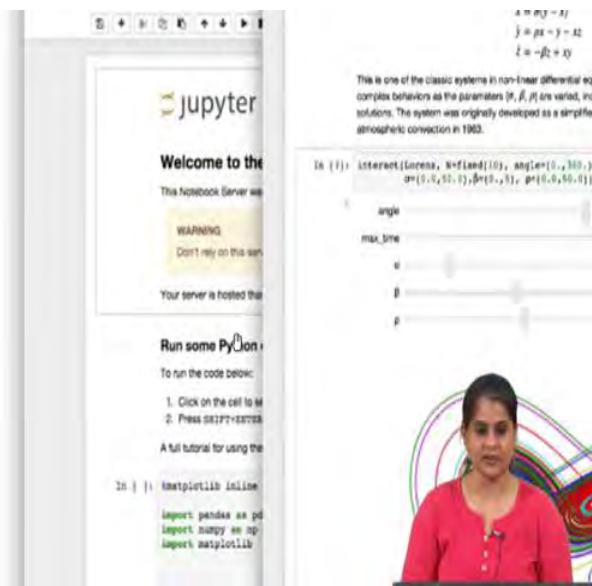
So, now let us look at the next IDE which is Jupyter notebook. So, now, Jupyter notebook is very different from the earlier two IDEs in the sense that it is a web application which allows creation and manipulation of the codes; now these codes are called notebook documents and hence that is how Jupyter gets its name Jupyter notebook. Now Jupyter is supported across all operating systems and it is available as an open source version.

(Refer Slide Time: 09:18)



Now, this is the interface of Jupyter, you can see that you have few cells here as an input you also have some output let me just zoom in and show you how the interface looks.

(Refer Slide Time: 09:33)



So, here you can see some of the codes that is written, if you just scroll up and see this is some narrative about whatever you have written.

(Refer Slide Time: 09:41)

A promotional slide for Jupyter Notebook. At the top, it says 'Jupyter Notebook' and has a 'GITAA' logo. Below that is a bulleted list of features: 'Bundled with Anaconda distribution or can be installed separately', 'Supports Julia, Python, R and Scala', and 'Consists of ordered collection of input and output cells that contain code, text, plots etc.' To the right of the list is a video player showing a woman speaking. The video player has a 'Python for Data Science' watermark at the bottom. The source of the video is cited as 'Source-<https://jupyter.org>'.

So, Jupyter is bundled with Anaconda distribution, but it can also be install separately. It primarily supports Julia, python, R and Scala. So, if you look at the name Jupyter it basically takes the first two letters from Julia the next two from python and then R.

So, that is how Jupyter gets its name as Jupyter it also consists of an ordered collection of input and output cells like how we earlier saw; and these can contain narrative text, code, plots and any kind of media.

(Refer Slide Time: 10:13)

The slide has a blue header bar with the text 'Jupyter Notebook' and the GITAA logo. Below the header is a bulleted list:

- Allows sharing of code and narrative text through output formats like PDF, HTML etc.
 - Education and presentation tool
- Lacks most of the features of a good IDE

On the right side of the slide, there is a screenshot of a Jupyter notebook interface showing a plot of a lens system. Below the screenshot is a photo of a woman in a pink shirt. At the bottom left of the slide, it says 'Python for Data Science'.

One of the key features of Jupyter notebook is that, it allows sharing of code and narrative text through output formats like HTML markdown or PDF. If you are working in an education environment or if you would like to have a better presentation tool, then you can use these kind of output formats to present. So, though Jupyter consist of features that give a very good aesthetic appeal to it, it is deficit of the important features of a good IDE. So, by good IDE, I mean it should consist of a source code editor and compiler and a debugger; and all three of these are not provided by Jupyter.

(Refer Slide Time: 10:50)

The slide has a blue header bar with the word 'ATOM' in white. Below the header, the word 'Atom' is written in a large, bold, blue font. To the right of the word 'Atom' is a small logo consisting of a blue square with a white geometric shape inside, followed by the word 'GITAA' in white capital letters. The main content area contains a bulleted list of features:

- Open source text and source code editor
- Supported across Linux, Mac OS X and Windows platforms
- Supports Python, PHP, Java etc.
- Well suited for developers
- Enables users to install plug ins or packages
 - Packages can be installed for code completion, debugging

At the bottom left of the slide, there is a small text 'Python for Data Science'.

On the right side of the slide, there is a video player interface showing a woman in a pink shirt speaking. The video player has a play button icon and a progress bar.

So, the next IDE that we are going to look into is atom. So, atom is an open source text and source code editor and is supported again across all over systems; it again supports programming languages like python, PHP, Java etc. And it is very very well suited for developers, it also helps the users to install plug-ins or packages. So, one common drawback with all these text editors and source code editor is that these do not come installed with basic libraries of any programming languages; you have to install these kind of packages as and when you have a need for them.

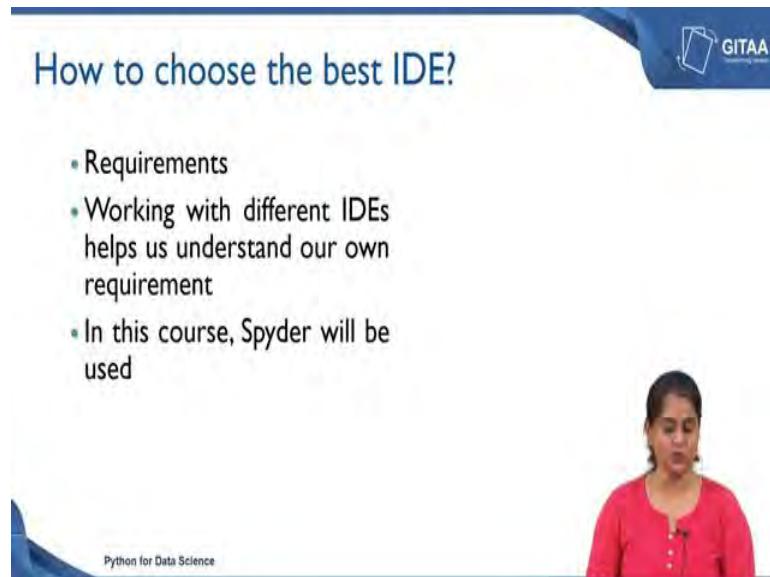
So, that is one major drawback for using any kind of text editor or the source code editor. But; however, atom does provide packages or libraries that are suited for data science and code completion or code navigation or debugging. So, you can install it, so if you are a developer and if you want to code an text editor environment then you can go ahead with atom. But you will have to install all these packages as and when you require.

(Refer Slide Time: 11:52)



So, this is the interface of atom, this is how it looks, it is a proper text editor interface.

(Refer Slide Time: 12:00)



So, how will you choose the best IDEs then important question. So, it basically depends on your requirements, but it is a good habit to work first with different IDEs to understand what your own requirements are. So, if you are new to python then it is better that you work across all these IDEs and there are several other IDEs out there you can work with all these IDEs see what suits you and then take a call on which IDE to use.

But in this course we are going to be looking at spyder; and that is primarily because it is a very good software that has been developed only for data science and python; and it is an interface that is very very appealing and easy to use for beginners.

(Refer Slide Time: 12:43)

Summary

- Popular tools used data science
- Evolution of Python
- Integrated development environment
 - Spyder
 - PyCharm
 - Jupyter Notebook
 - Atom

Python for Data Science

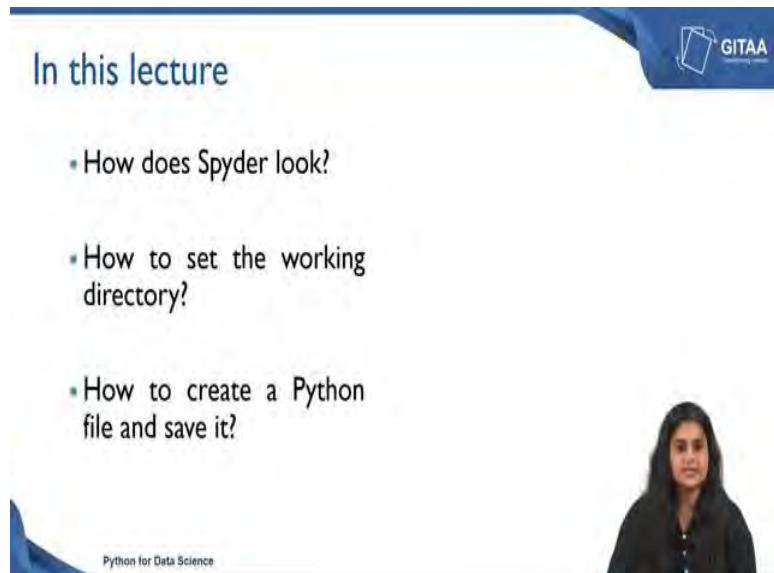
So, to summarize in this lecture we saw what are the popular tools used in data science environment. We also saw how python evolved and what are the commonly used integrated development environment. We also looked at what each of these IDE have to offer us and some of the common pros and cons of each of these.

Thank you.

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 03
Introduction to Spyder
Part -1

(Refer Slide Time: 00:17)



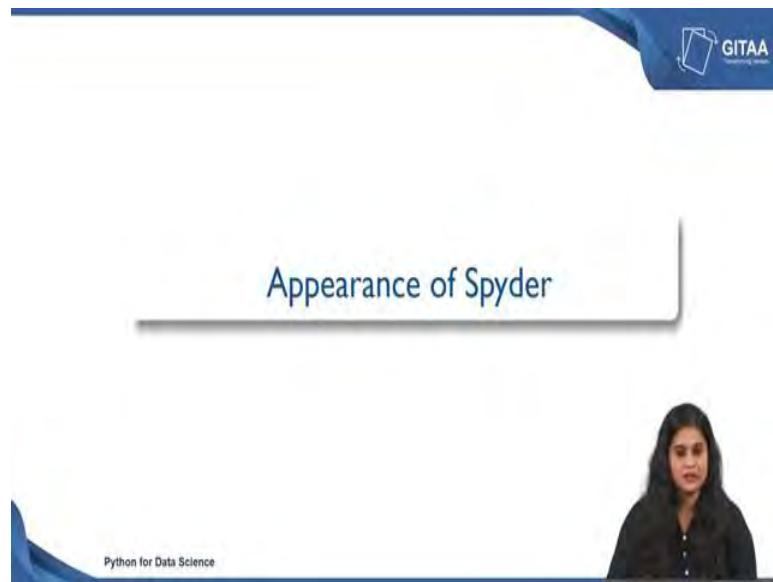
In this lecture

- How does Spyder look?
- How to set the working directory?
- How to create a Python file and save it?

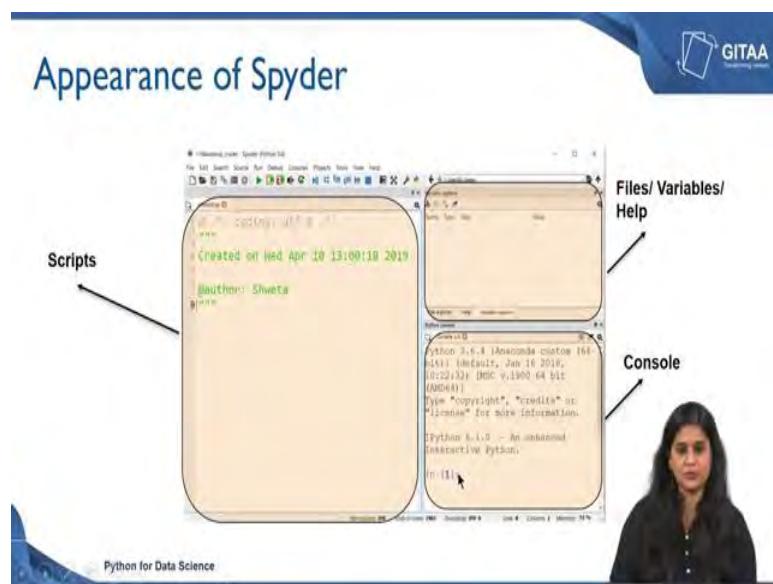
Python for Data Science

Welcome to the lecture on Introduction to Spyder, in this lecture we are going to see how does the interface of spyder look? How to set the working directory and how to create and save a Python file?

(Refer Slide Time: 00:28)



(Refer Slide Time: 00:32)



So, let us see how does the appearance of spyder look. So, on my left you can see a snapshot of the screen that would appear once you open Spyder. So, the Python version that I am using to illustrate this lecture is version 3.6. So, once you open you will get a small description of the author name and when the file was created. There are a couple of windows though here so let us see what each of these windows mean.

So, the entire interface is split into three windows, the window on my left is called the scripting window and all your lines of codes and commands that you are going to type

will be displayed here. So, you have to write all your commands and codes here on my right I have two windows, the top section is where you would find tabs that read as file explorer, help and variable explorer.

Now under file explorer once you set the directory if you have any files that are existing in your current working directory, then all these files will be displayed under file explorer under variable explorer you will basically be having a display of all the objects and variables that you have used in your code. Now, along with the variables you also have their name, type and size. Now, name is the name of the variable, type is the data type and size is whether it is an array or a single value. Now, the first few values will be displayed if it is only a single value then the single value be displayed under, the heading value the section on the bottom is the console.

So, console so is an output window where you will be seeing all your printed statements and outputs, you can also perform elementary operations in your console, but the only disadvantage is that you will not be able to save it. Now however, whatever you type in the scripting window can always be saved. So, we are going to look into how to save the lines of commands that you have used in your scripting window and we will do that once the lecture proceeds.

(Refer Slide Time: 02:34)

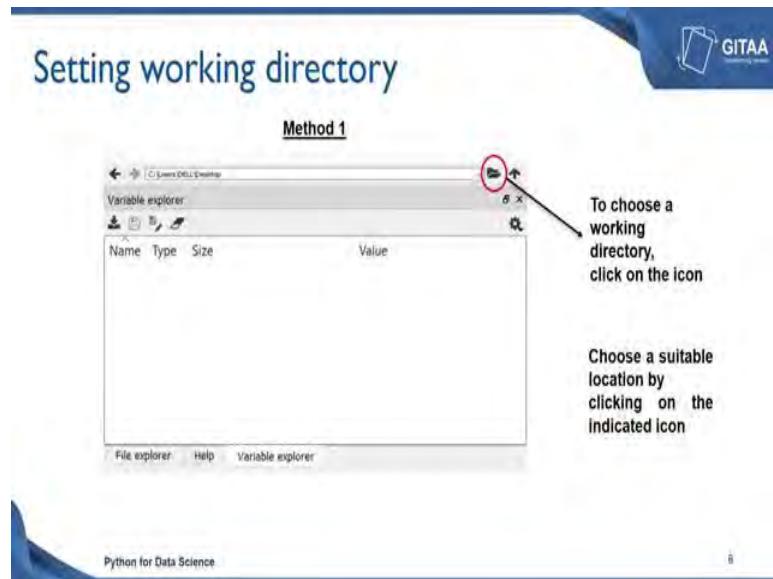
Setting working directory



- There are three ways to set a working directory
 - Icon
 - Using library **os**
 - Using command **cd**

Now, let us see how to set the working directory, there are three ways to set a working directory the first is using an icon, the second is using the inbuilt library OS and the third is using a command cd which means change directory.

(Refer Slide Time: 02:47)



Now, let us see how to set a working directory using the icon. If you look at the top section here you will see an icon here with a folder open, now you can choose a working directory by clicking on this icon. Once you choose you will be prompted to choose a location or a folder. Now, you can choose a suitable folder or a suitable location by clicking on the icon and once you click on the location your directory is considered to be set. Now this is an easy method and if you do not want to be typing commands every single time, then you can just do a drag and drop.

(Refer Slide Time: 03:28)

Setting working directory

Type the following in the console

Method 2

```
# Import os to setup the working directory
import os

# Setting the working directory
os.chdir('C:/Users/DELL/Desktop')
```

Method 3

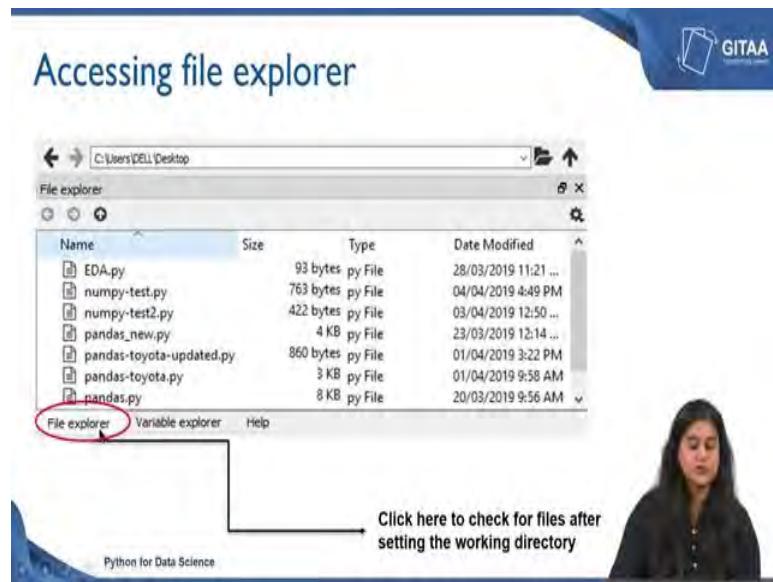
```
cd C:/Users/DELL/Desktop
```

Python for Data Science

Now, let us look at the second and the third methods, now you need to import a library called os, os stands for Operating Systems. Before you use a function from this library to change the directory you need to import it. So, import is a function that you will use to load a library to your environment.

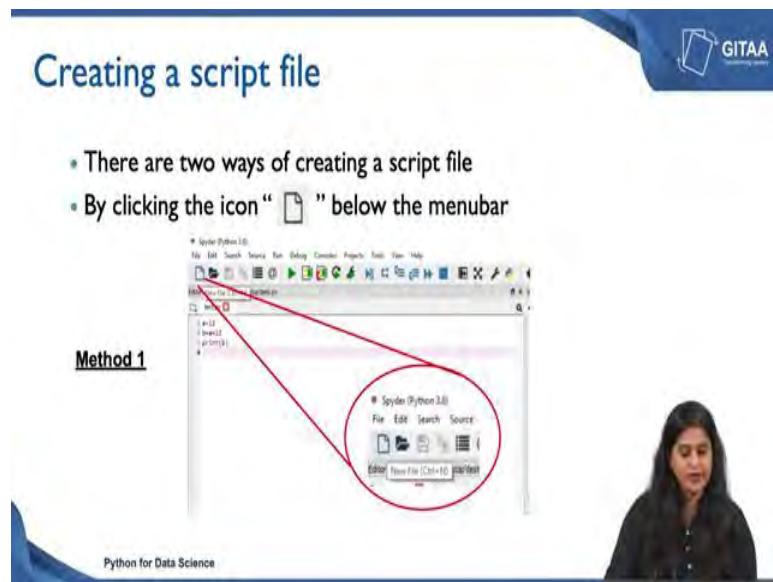
Now, once you load the library OS on your environment you can use the function chdir which means change directory. So, I need to use the name of the library which is OS in this case followed by a dot and then use chdir. Now, within parenthesis you we can give single or double quotes. So, copy the entire path from your directory and then paste it here or you can also type it out. The third method is using the command cd, cd also means Change Directory and you can give a space after the command and then give the path. So, this how you set a working directory.

(Refer Slide Time: 04:32)



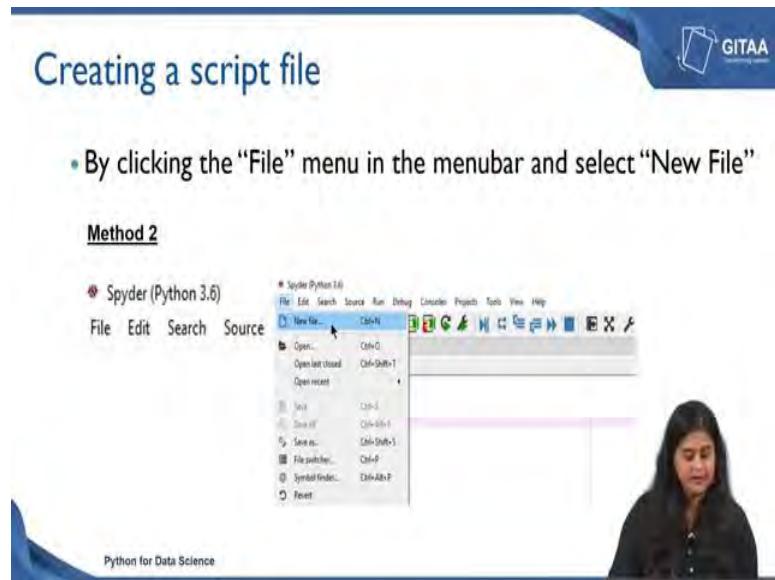
Now, once you set the working directory if you have any folders or any sub folders or any other files inside the working directory, all of that will be displayed under file explorer. For me I have a couple of files under this directory and hence it is being displayed here for me. But of course, if you are opening a new folder you are likely to see this space as empty now you can check all your files and sub file and sub directories here under file explorer.

(Refer Slide Time: 05:09)



So, we have seen how to set a working directory, now let us see how to create a file. So, there are two ways to go about it the first is by clicking an icon that looks like a page folded on the right. Now, this you can find on the toolbar. So, on the icon bar towards your extreme left you will see a page that is folded on the right, now if you click on that a new script file will open. I have also shown you a zoomed in version of the icon, so this is how it looks, the moment you click on it a new script file will pop up.

(Refer Slide Time: 05:39)



Now, the second method is by clicking on the file menu and then selecting new file. So, you can see the file menu here and then from that click on new file. Now, apart from these two methods you always have a fallback option of using the keyboard shortcut which is Ctrl + N, in all these three methods right away open a script file for you till. Now, we have set the working directory we have created a script file. So, now let us type few pieces of code before we save our script file, but even before we go there let us look at what a variable means.

(Refer Slide Time: 06:00)

The slide has a blue header with the word 'Variable' in white. In the top right corner is a logo for 'GITAA' with a stylized book icon. The main content area contains a bulleted list:

- An identifier containing a known information
- Information is referred to as value
- Variable name points to a memory address or a storage location and used to reference the stored value

At the bottom left of the slide, it says 'Python for Data Science'. On the right side of the slide, there is a video player showing a woman with long dark hair speaking.

So, variable is an identifier that contains a known information, the known information that is contained within an identifier referred to as a value. So, a variable name will actually point to a memory address or a storage location and then this location is actually used to cross refer to the stored value. So, variable name can be descriptive or can also consist of single alphabets. So, we will look into the naming conventions of naming a variable in the lectures to come.

(Refer Slide Time: 06:47)

The slide has a blue header with the words 'Creating variables' in white. In the top right corner is a logo for 'GITAA' with a stylized book icon. The main content area shows two side-by-side Python environments:

- Spyder (Python 3.6):** A graphical interface with multiple panes. The code editor pane shows the following code:

```
a=11  
b=a*10  
print(a,b)
```
- Jupyter Notebook:** A web-based interface with a code cell containing:

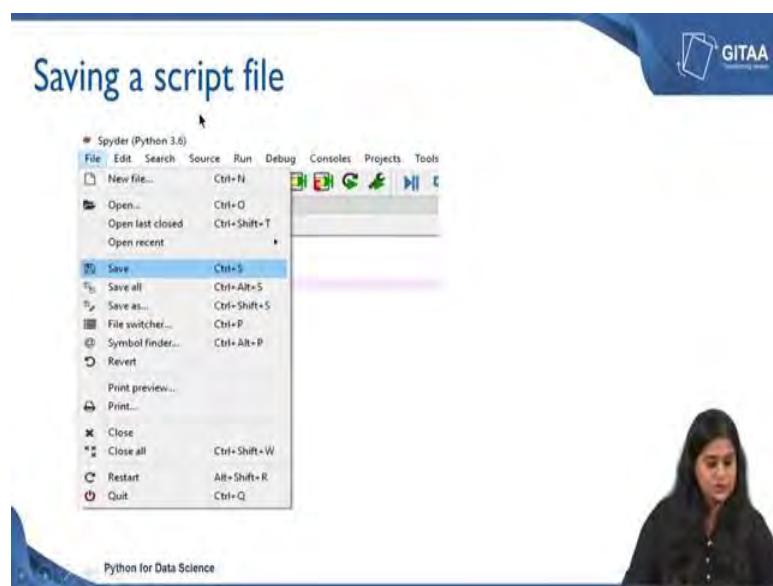
```
a=11  
b=a*10  
print(a,b)
```

At the bottom left of the slide, it says 'Python for Data Science'. On the right side of the slide, there is a video player showing a woman with long dark hair speaking.

So, let us go ahead and create few variables, now you will see a snapshot of a code here on my left I have zoomed in the lines of code on my right. So, let me again zoom in and show you now I am assigning a value of 11 to a. So, in Python the assignment operator that you will be using to assign a value is equal to. So, I am storing a value of 11 in a, a is my variable name and I am saying $b = 8 * 10$.

So, this is a multiplication and the multiplication operator in Python is referred as asterisk. So, once I create both my variables I would like to print the values of a and b, now because I want to print two values together; I am going to separate them with a comma inside the print statement. So, the print statement will help me print the output and since I want to print two outputs here I am going to separate them with a comma. However, if you just want to print one statement you can just give a single object inside the parentheses.

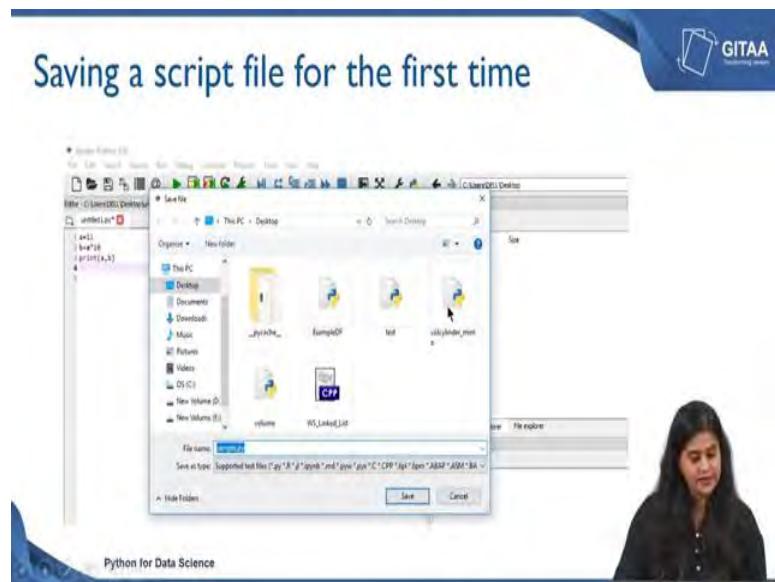
(Refer Slide Time: 07:59)



So, now let us go ahead and save our script files. So, to save your script file you can click on the file menu again and you can see there are three different options here. So, let us see what these options are I am going to zoom in a bit to show you the list of options that you have. So, the first option is save which is represented as Ctrl + S in your keyboard shortcut. Now, if you already have a file now if you are making some changes to it, then if you would like to save changes that you made then you can just simply click on save.

Now if you are making changes across multiple files. So, now, if you are opening multiple files and making changes in all of them then you can use the option save all. So, what save all does is that it will save all the changes made across all the files that are open. So, this is the use of save all. So, the third option is what is called as save as, now if you are creating a new file and you would like to rename it and save it then you would be using save as. So, let us see how to save a new script file for the very first time.

(Refer Slide Time: 09:07)



So, once you click on save as it will prompt you to give a name for the file. Now, you can choose your directory here as to where you want to just save it or if you already in your working directory then you can just go there and save it. So, dot py is the extension that is used to save a Python script file. Now once you do this you can just click on save and your file is saved.

(Refer Slide Time: 09:35)

The image shows a presentation slide with a blue header containing the word 'Summary'. Below the header is a bulleted list of three items: 'Interface of Spyder', 'Setting the working directory', and 'Create and save Python script file'. At the bottom left of the slide, it says 'Python for Data Science'. In the top right corner of the slide area, there is a small video frame showing a person with long dark hair. The background of the slide is white.

- Interface of Spyder
- Setting the working directory
- Create and save Python script file

Python for Data Science

So, to summarize in this lecture we saw how the interface of Spyder looks, we saw how to set the working directory and how to create and save Python script files.

Thank you.

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 04
Introduction to Spyder Part -2

Welcome to the 2nd lecture on Introduction to Spyder.

(Refer Slide Time: 00:17)

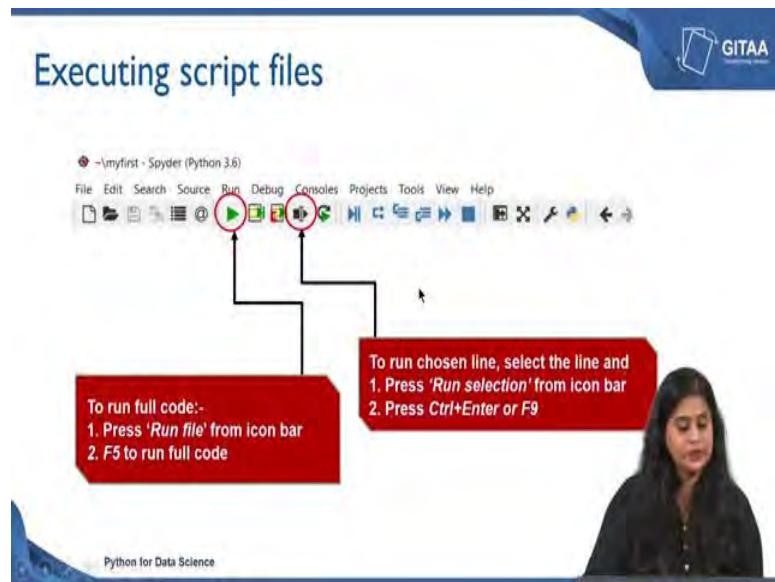
The image shows a presentation slide with a blue header bar. On the left, the text 'In this lecture' is displayed in a blue font. On the right, there is a logo for 'GITAA Learning Center' featuring a book icon and the text 'GITAA Learning Center'. Below the header, there is a list of bullet points in black text:

- How to execute a Python file?
- How to execute pieces of code - Run?
- How to add comments?
- How to reset and clear console

At the bottom of the slide, there is a video player window showing a woman with long dark hair speaking. The video player has a play button in the center. At the very bottom of the slide, the text 'Python for Data Science' is visible.

In this lecture, we are going to see how to execute a Python file, how to execute few pieces of code using run and how to add comments and we will also see how to reset the environment and clear the console.

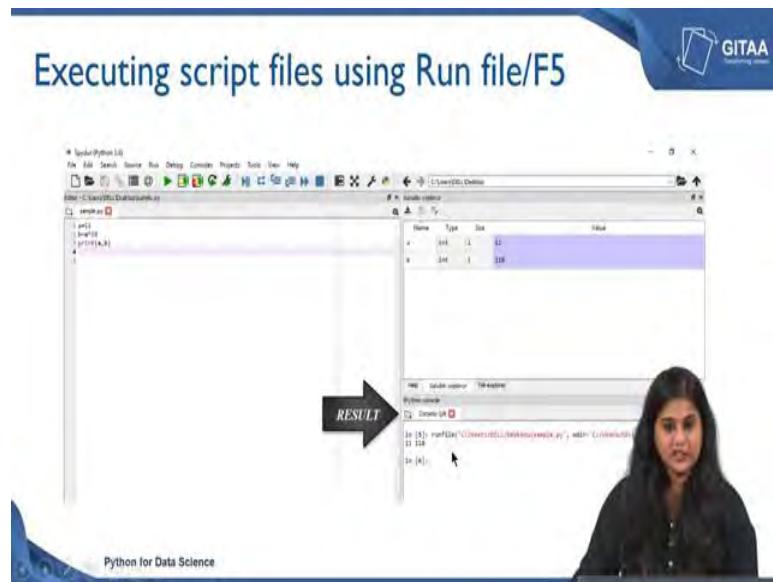
(Refer Slide Time: 00:34)



So, let us begin with file execution. On the top in the icon bar you will see a green triangle with its end pointed to right; now this is called the run file option and this will help you run an entire file at once and equivalent shortcut from the keyboard to press F5.

Now, if you want run few section of code or few lines of code, then you can click on the run selection option and this will help you run a chosen line. And equivalent shortcut from the keyboard is to either press F9 or press 'Ctrl+Enter'; after choosing the line. So, let us see how each of these options work and what are the corresponding outputs that they give?

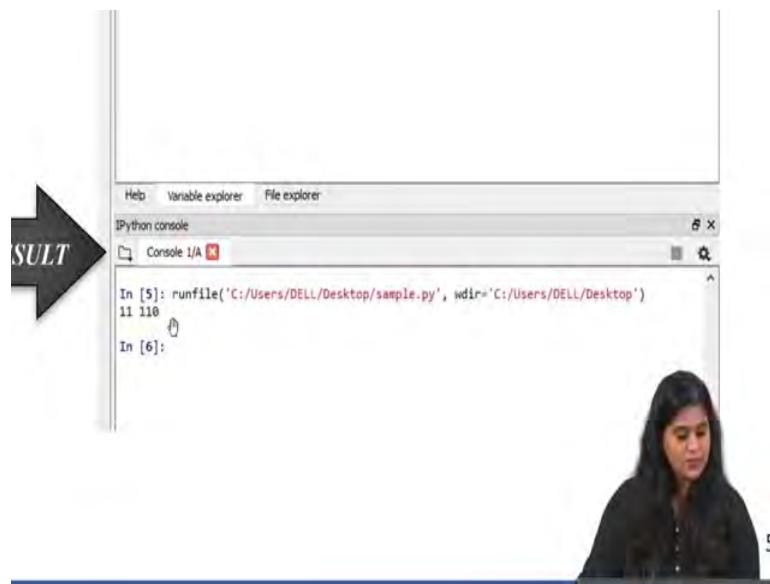
(Refer Slide Time: 01:15)



So, I am now first starting with the run file option which will run an entire file at once. So, once you have code ready you can click on the green triangle icon and you will see an output here. In this case, I am running the same script file that I used in my earlier lecture.

Now which says `a=11, b=8 *10` and `print(a,b)`. So, the output I am likely to get are the values of `a` and `b` here. So, once you run the code; you will see the values being stored in the environment and this you can find under the variable explorer. So, in the variable explorer you can see I have my values of `a` and `b` and they are integer type; the size is 1 because the both have only one value and the corresponding value is displayed. Now, after I run on my console; I have the output here. So, let me just zoom in to show you how does the output look.

(Refer Slide Time: 02:13)



So, this my output I have my value of a and b; a is 11, b is 110. Now, since these work contain within the print statements; I will be getting only these two values as my output. So, another thing to note is that when whenever you click on the run file option, corresponding function is run. So, the run file command is actually the function that will be used to run an entire file at once.

If you are not using the icon or the keyboard shortcut, you can also run your file using this command. The input to the run file function is the name of the file along with its entire directory. Now, you also have another parameter called wdir which means working directory and you can specify in whichever a directory the file is residing.

So, me for it is residing the desktop; so I have given the same directory. So, if do not want to use the icon or the keyboard shortcut, we can also use that run file command. So, in this case since the output is contained within the print statement; you will be able to see only the output a and b.

(Refer Slide Time: 03:23)

Executing script files using Run selection/F9

Step 1: Assign a new value of 14 to 'a' in the script and press F9

The screenshot shows the Spyder Python IDE interface. A red circle highlights the line of code 'In [1]: a = 14'. An arrow points from this line to the text 'Console output' located below the code editor. The code editor window contains the following Python script:

```
1 #!/usr/bin/python
2 a=10
3 print(a,b)
4
```

A woman is visible on the right side of the slide, likely the instructor.

So, now let us see how to execute few pieces of code using the run selection option or the F9 command from the keyboard. Now, to my earlier code I am assigning a value of 14 to a and I am selecting the line and then I pressing F9. So, I am just going to use the shortcut here; you can also use this icon. Now once you select the line and press F9, you will see a corresponding output being displayed in your console; now this says a = 14. So, whenever you use run selection command or F9; all these lines of code will be displayed in your console. So, whatever I have shown you here is the console output.

(Refer Slide Time: 04:08)

Executing script files using Run selection/F9

Step 2: Select line 2 and press F9

In [2]: b = a*10 ————— Console output

Step 3: Select line 3 and press F9

In [4]: print(a,b) ————— Console output
14 140

The screenshot shows the Spyder Python IDE interface. It displays three steps of code execution:

- Step 2: The line 'In [2]: b = a*10' is highlighted with a red circle, and an arrow points to the text 'Console output' below it.
- Step 3: The line 'In [4]: print(a,b)' is highlighted with a red circle, and an arrow points to the text 'Console output' below it.

A woman is visible on the right side of the slide, likely the instructor.

So, now run the second line which is `b=a*10` and then press F9, now once you do that you will corresponding see the code in your console. Then you can run the line which says `print(a,b)` and once you run this line `print(a,b);` you will see a corresponding output as well. So, if you have noticed in run every time you select a line and run; it the corresponding code is also displayed in the console. But however, if you going to use run file command which runs an entire file at once then all these lines of code is not printed in the console. So, this is also one of the difference between using a run file and run selection command.

So, you can also use run selection to debug. So, if you want to go through each and every line and if you want to find out bags or falls on mistakes that you would have done then you can use the run selection option. So, this is one of the major advantages of using run selection, but again if you have a 1000 line or if you have. So, again if you have a large code then it is going to be impossible for you to use the run selection option.

(Refer Slide Time: 05:20)

The screenshot shows a computer monitor displaying a Python code editor. The window title is "Editor - C:\Users\DELL\Desktop\sample.py". The code in the editor is as follows:

```
1 #Simple Example
2
3 #Calculate Volume of Cylinder
4 dia = 5
5 dia=5
6 dia is for diameter
7 len=4
8 len is for length
9 vol=3.14*(dia**2)*len/4
10
11
```

On the left side of the slide, there is a bulleted list of three items:

- Adding comments will help in understanding algorithms used while developing codes
- In practice, commented statements will be added before the code and begin with a '#'
- Multiple lines can also be commented

At the bottom left of the slide, it says "Python for Data Science".

So, now let us move on to commenting script files. So, adding a comment will aid in the understanding of algorithms that have been used to develop a code. On my right, you can see a snapshot now this is a very trivial example that describes how a volume of cylinder is calculated to comment any line you basically begin with '#'. So, here I have described the title of the task that I am going to do.

Now, I am going to calculate the volume of cylinder; now apart from describing the task or the task objective you can also define what each of the variables mean in your code. So, here I said dia is diameter, len is length vol is volume. Now, this is a very good practice because if you are going to give your code to someone or if you are going to revisited in the future; you might want to know what you have done and why you done it. Now, you can also comment multiple lines instead of just one line.

(Refer Slide Time: 06:13)

The screenshot shows a presentation slide titled "Commenting multiple lines" from a course on "Python for Data Science". The slide contains three bullet points:

- Select lines that have to be commented and then press "Ctrl + 1"
- Select "Edit" in menu and select "Comment/Uncomment"
- Uses - to add description, render lines of code inert during testing

A screenshot of the Spyder Python IDE interface is overlaid on the slide. A context menu is open, and the "Comment/Uncomment" option is highlighted with a blue box. The keyboard shortcut "Ctrl+1" is displayed next to it. Other options in the menu include "Add block comment", "Remove block comment", "Indent", "Unindent", "Toggle Uppercase", and "Toggle Lowercase". The Spyder menu bar at the top includes File, Edit, Search, Block, Run, Debug, Cluster, Projects, and Tasks. The GITAA logo is visible in the top right corner.

So, to comment to multiple lines select on the lines that have to be commented and then press on “Ctrl+1”; now, this is the keyboard shortcut and alternative ways to go to the edit option in the menu and then select comment or uncomment lines.

You can see the keyboard shortcut also being displayed just adjacent to the comment/uncomment option under the edit menu. Like I earlier said you can add description to your code to make it more comprehensible, but apart from just making it incomprehensible if you are in the beginning stage of developing a code where you are trying and testing out of you think then you can also use commenting as a way of making a few lines inert.

(Refer Slide Time: 06:58)

Commenting multiple lines

- Select lines that have to be commented and then press "Ctrl + 1"
- Select "Edit" in menu and select "Comment/Uncomment"
- Uses - to add description, render lines of code inert during testing

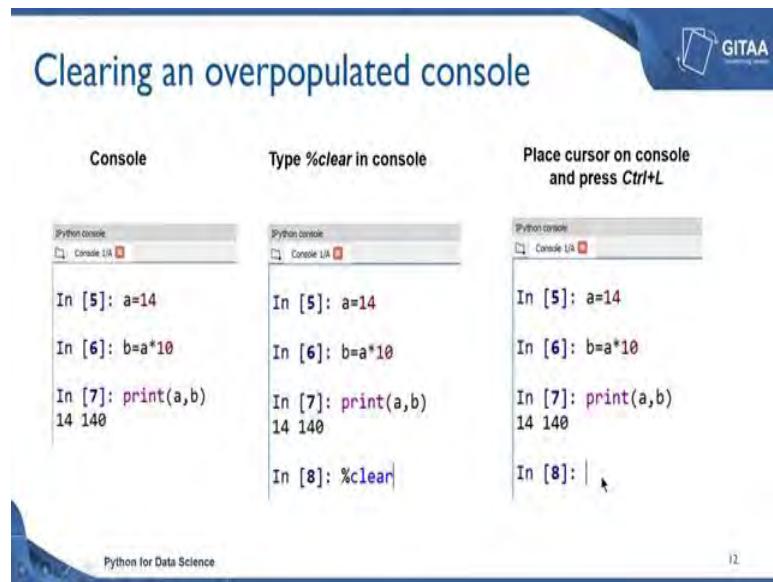
Python for Data Science

Spyder (Python 3.6)
File Edit Search Source Run
Editor - C:\Users\DELL\Desktop\sample.py
sample.py*
1 #
2 b=a*10
3 print(a,b)
4
5|

So, what do I mean by this is that. So, let us take the previous example where I say `a=14`, `b=8*10` and `print(a,b)`. Now I am just trying and testing out and seeing what will happen if I just comment `a`. So, I am basically just making the first line inert and then running the successive lines.

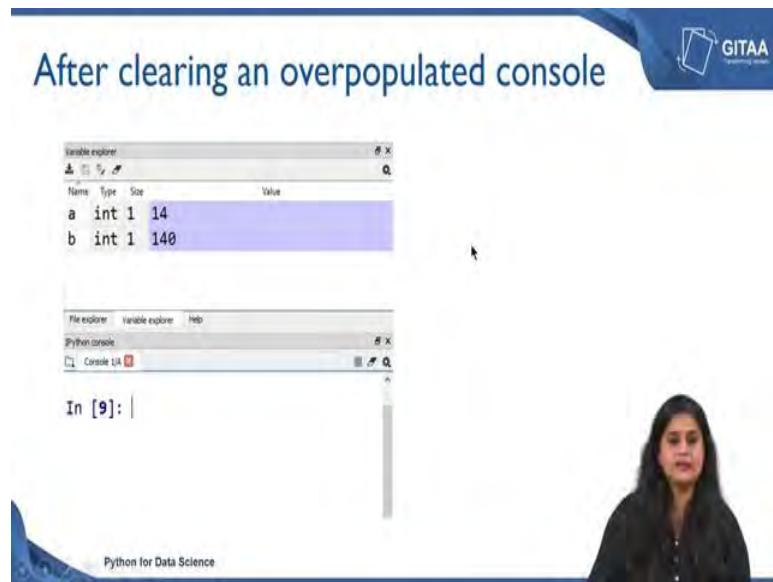
Now, if you are playing with your code and you are in the developing stage then you can also use commenting as a way of keeping lines inert. So, now this is another use of committing. So, till now we seen how to execute an entire file at once and how to execute few lines of course, we have also looked at commenting as a way of adding description to your code. Now let us see how to clear the console and the environment.

(Refer Slide Time: 07:43)



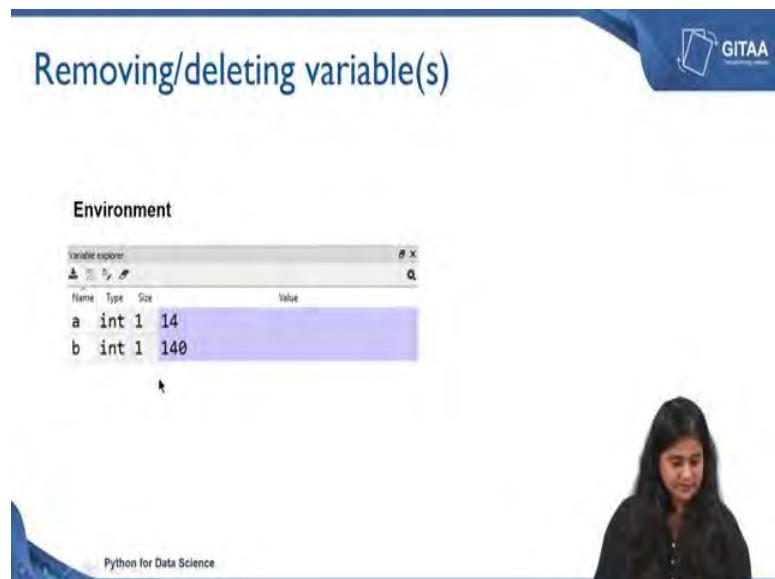
If you have an overpopulated console where you have printed multiple lines of codes and multiple outputs; then you might also want to just clear it off and start a fresh. So, let us take this example where I have run the codes; I have the same codes as earlier and I am just running them. So, this is how my console looks right now; now if I want to clear it. So, you can type “%clear” in the console and once you hit enter, your entire console is clear. Now, an alternate ways to use the “Ctrl+l” shortcut from the keyboard that will also work; so this is to clear the console.

(Refer Slide Time: 08:24)



Now, once you clear the console this is how it looks; I have snapshot here that tells you how does the console look once you clear it. An important point to note here is that; so whenever I clear my console only the output windows cleared the variable explorer still remains intact all the variables are still there; so, clearing a console only means that you are just clearing or flushing out the output window. So, now let us see instead of just clearing the console is there a way to just clear the environment as well.

(Refer Slide Time: 08:57)



So, I might also be interested in removing or deleting a few variables from my environment. So to begin with I have two variables in my environment which is a and b and they have a value of 14 and 140 respectively. So, let us see how to remove or delete these variables.

(Refer Slide Time: 09:15)

The screenshot shows a Python Data Science interface with two side-by-side Variable explorer windows and a Python console window below them.

Left Window (Removing single variable): Shows a table with one row: Name (a), Type (int), and Size (14). A callout points to the word "del" in the Python console below, which has a red circle around it. The console shows "In [9]: del b".

Right Window (Removing multiple variables): Shows a table with two rows: Name (a) and Name (b). A callout points to the command in the Python console below, which has a red circle around it. The console shows "In [13]: del a,b".

Python console:

- In [9]: **del** b
- In [13]: **del** a,b

Using `del` followed by variable name

PyCharm logo and GITAA watermark

Now, to remove a single variable; you can just give del space followed by the variable name. Now this you can type in the console; so del stands for delete and it has to be followed by a space and then a variable name. Now, once you hit enter what you will see is that one of the variables in this case b has been removed from the environment. So, here you can see variable b has been removed from the environment. Now, instead of removing a single variable we can also remove multiple variables from the environment. Now still use the same command del, but instead of just giving one variable; you will give two variables and you have to ensure that you separate the variables with the comma.

Now, if you are typing along with me please ensure that b is also present in the environment and then you can type this code. So, once you hit enter you can see that the entire memory has been flushed out and both these values have been deleted.

(Refer Slide Time: 10:18)

The screenshot shows a Jupyter Notebook interface. On the left, there is a sidebar with 'File', 'Variable explorer', 'File browser', and a search bar. A red box highlights the 'Create New' button. The main area shows a code cell with the following content:

```
In [13]: %reset
```

```
In [14]: #<cell output>
```

```
In [15]: #<cell output>
```

```
In [16]: #<cell output>
```

```
In [17]: #<cell output>
```

Below the code cell, a message reads: "Once deleted, variables cannot be recovered. Proceed? (y/n)[n]" with a cursor at the end of the line. To the right of the notebook, a woman with long dark hair is looking down at the screen. The top right corner of the slide features the GITAA logo.

So, instead of dropping variables one by one; you might also be interested in clearing the entire environment at once. So, there are two ways to go about it the first way is to use a “%reset” command in the console.

Now, once you type %reset and hit enter; it will prompt you with the line that reads as once deleted, variables cannot be recovered; proceed yes or no; y stands for yes and n stands for no. Now, this is to ensure that you have not typed percentage reset accidentally and this is just another layer of check to make sure that you do not flush out the important variables in your environment. Now, if you would like to proceed then type ‘y’; otherwise you can type ‘n’ and now once you hit enter, you will see that the entire environment has been cleared out.

(Refer Slide Time: 11:07)

The screenshot shows a Jupyter Notebook interface. At the top, there's a blue header bar with the text "Clearing the entire environment at once" and the GITAA logo. Below the header, a list item says "• There are two ways to clear the environment". A callout box labeled "Method 1" points to the "Variable explorer" window, which lists variables "a" and "b" with their values. Another callout box labeled "Method 2" points to the "Kernel" menu, which has an option "Clear all" highlighted with a red box. A woman is visible in the bottom right corner of the slide.

So, this is using a command.

(Refer Slide Time: 11:13)

The screenshot shows a Jupyter Notebook interface. At the top, there's a blue header bar with the text "Clearing the entire environment at once" and the GITAA logo. Below the header, a callout box labeled "Method 2" points to the "Variable explorer" window, which lists variables "a" and "b" with their values. A text overlay on the left of the variable explorer says "Click the ✎ symbol to remove variables in environment". Another callout box labeled "Method 2" points to the "Kernel" menu, which has an option "Clear all" highlighted with a red box. A woman is visible in the bottom right corner of the slide.

Now, let us see how to clear the environment using an icon. Now, above the variable explorer there are a couple of icons here; the one on the extreme right looks like an eraser. So, now click on the icon it will prompt you to the dialogue box; if you click all the variables will be removed. So, while it is removing variables it also prompts with the line that says removing all variables. So, till now we have seen how to execute an entire

files or few lines of codes and how to comment the code that you have written and how to clear the console and the environment.

(Refer Slide Time: 11:56)

The screenshot shows a presentation slide titled "Basic libraries in Python". On the left, there is a list of basic libraries:

- Basic libraries
 - NumPy – Numerical Python
 - Pandas – Dataframe Python
 - Matplotlib - Visualization
 - Sklearn – Machine Learning
- Modules within a library. E.g.-

Below the list is a code snippet:

```
import numpy  
content = dir(numpy)  
print(content)
```

An arrow points from the code to a screenshot of a Python terminal window. The terminal window shows the output of the code, which is a list of module names:

```
'ascalar',  
'atleast_1d',  
'atleast_2d',  
'atleast_3d',  
'average',  
'bartlett',  
'base_repr',  
'bench',  
'binary_repr',  
'bincount',  
'bitwise_and',  
'bitwise_not',  
'bitwise_or',  
'bitwise_xor',  
'blackman',  
'block',
```

So, now let us take a look at some of the basic libraries in Python. Now, there are four major libraries that get installed at the time of installation of Python. These are NumPy which stands for Numerical Python, Pandas which Paneled Dataframe, Matplotlib which stands for Visualization and Sklearn which is used for machine learning. So, these are four major library that are important to solve a data science problem.

So, these are parent libraries; there also sub libraries contained within these. So, to access the contents of a library, you need to first import the library; in this case I am importing NumPy. Now 'dir' represents directory; so this is the directory of the library; in this case it is numpy. Now, I am just saving the entire directory on to a variable name called content and I am just printing the object content. So, once you run these three lines of code on your console all the sub libraries will be printed. So, this is one way to actually access the sub library. So, now this is the little tedious because you actually have to skim through all the sub libraries to know what they consist of and also your console get overpopulated.

(Refer Slide Time: 13:12)

The screenshot shows a web-based help interface titled "Help in Python". At the top, there is a search bar with the placeholder text "Type the name of the library in 'Object'". To the right of the search bar, the text "The following are the sub libraries" is displayed, followed by a list of available subpackages: doc, lib, random, linalg, fft, polynomial, testing, f2py, and distutils. Below this, there is a note: "Note: You can click the details of the sublibraries by typing `libraryname.sublibraryname` under Eg- `numpy.lib` in object". A video player window is overlaid on the bottom right, showing a woman speaking.

Now, under the help tab you have a search box that's titled as object; let me just zoom into show you how it looks.

(Refer Slide Time: 13:19)

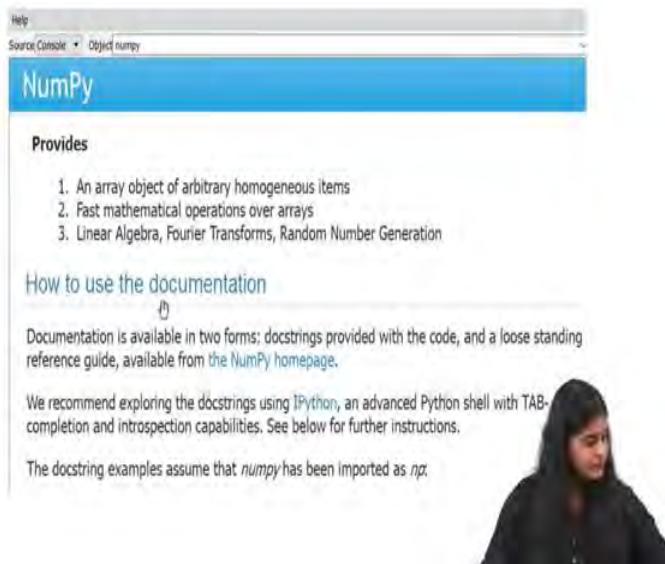
Help in Python

Type the name of the library in 'Object'

The screenshot shows a web-based help interface titled "Help in Python". In the search bar, the text "Object numpy" is typed. The results show the "NumPy" library documentation. The "Provides" section lists: 1. An array object of arbitrary homogeneous items, 2. Fast mathematical operations over arrays, and 3. Linear Algebra, Fourier Transforms, Random Number Generation. Below this, there is a "How to use the documentation" section. A video player window is overlaid on the bottom right, showing a woman speaking.

Now, under object you are going to mention your library name. So, in this case I have mentioned NumPy; the moment you hit enter a documentation pops up.

(Refer Slide Time: 13:30)



The screenshot shows a computer screen displaying the NumPy documentation. The title bar says "NumPy". The main content area has a blue header "Provides" with three bullet points: 1. An array object of arbitrary homogeneous items, 2. Fast mathematical operations over arrays, 3. Linear Algebra, Fourier Transforms, Random Number Generation. Below this is a section titled "How to use the documentation" with a small icon of a person. It contains two paragraphs: one about docstrings and another about exploring them in IPython. At the bottom right of the slide, there is a small portrait of a woman.

Now, the documentation tells you what Python does; what it provides and how to use the documentation. Now, apart from that if you scroll down, you can also see a list of sub libraries that are available under NumPy.

So, the sub libraries available under NumPy are linear algebra, Fourier transform routines, polynomial tools so on and so forth. Now, if you want a specific documentation for each of the sub libraries; so you can type in the library name in the search box, follow it up with dot and then the sub library name. So, let us say if I want to access the sub library lib from numpy; then I am going to write 'numpy.lib' under the search box object. So, this is how you get a detailed documentation of all the libraries and sub libraries in Python.

(Refer Slide Time: 14:23)

Summary

- Execute Python script file
- Commenting lines of code
- Clearing console and environment
- Basic libraries in Python

Python for Data Science

So, to summarize in this lecture we saw how to execute Python script files, how to comment single lines of codes and multiple lines of codes, how to clear the console and the environment and how to access some of the basic libraries in Python.

Thank you.

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

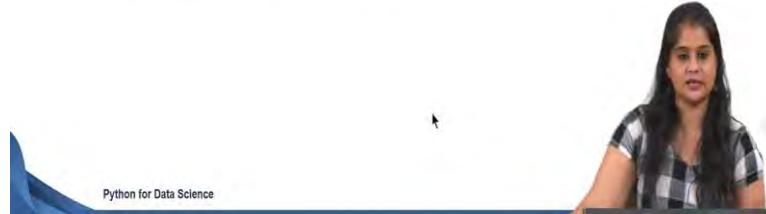
Lecture – 05
Variables and Data Types

(Refer Slide Time: 00:18)

In this lecture



- Naming variables
- Basic data types
 - Identify data type of an object
 - Verify if an object is of a certain data type
 - Coerce object to new data type



Welcome to the lecture on Variables and Data Types. In this lecture, we are going to look at how name variables, some of the common rules and conventions in naming a variable. We are also going to look at some of the basic data types that we are going to use throughout this course and in Python. As a part of this task, we are going to look at how to identify a data type of an object, how to verify if an object is of a certain data type and how to coerce objects to a new data type.

(Refer Slide Time: 00:43)

The slide has a blue header bar with the GITAA logo and the text 'Transforming Careers'. The main title 'Naming variables' is in large blue font. Below it is a bulleted list of guidelines:

- Values assigned to variables using an assignment operator '='
- Variable name should be short and descriptive
 - Avoid using variable names that clash with inbuilt functions
- Designed to indicate the intent of its use to the end user
- Avoid one character variable names
 - One character variable names are usually used in looping constructs, functions, etc

At the bottom left, it says 'Python for Data Science'. On the right side, there is a video thumbnail of a woman with long dark hair, wearing a plaid shirt, speaking.

So, let us begin with naming variables. So, values are assigned to variables using the assignment operator equal to(=). So, the variable name should be short and descriptive, and that is because there is an intent for creating the variable and it is supposed to convey an information hence it is better that it is short and descriptive. So, avoid using variable names that clash with inbuilt functions.

Like I earlier said, the variable names are designed to indicate the intent and purpose of its use to the end user. So, hence it is better to avoid one character variable names. So, one character variable names are usually used in iterations, and functions and looping constructs.

(Refer Slide Time: 01:23)

Naming variables



- Variables can be named alphanumerically

Age=55 age=55 age2=55 Age2=55

- However the first letter must start with an alphabet (lowercase or uppercase)

```
In [3]: 2age=55
         File "<ipython-input-3-00efac86f1ae>", line 1
             2age=55
                 ^
SyntaxError: invalid syntax
```



Python for Data Science

So, variables can be named alpha numerically. So, if I have a variable called age whose value is 55, then I can either have the entire age in lower case or I can also begin with an upper case. You can also add a number 2 weight. So, let us say if I am creating another variable called age 2, then I can add the number after the alphabets. So, one thing that you would have noticed here is that the first letter should always begin with an alphabet, but however, if you were to begin with the number the compiler throws an error saying invalid syntax.

(Refer Slide Time: 02:01)

Naming variables



- Other special character
 - Underscore (_)

→ Employee_id=501

- Use of any other special character will throw an error

```
In [6]: Employee@id=501
         File "<ipython-input-6-e2ec8cb31eb>", line 1
             Employee@id=501
                         ^
SyntaxError: can't assign to operator
```

- Variable names should not begin or end with underscore even though both are allowed

→ _age=55
age_=55



Python for Data Science

So, the only other special character that is allowed while naming a variable is underscore. Now, let us say if I want to create a variable that conveys the employee id, then I can separate the employee and id with an underscore. Now, underscore is the only other special character that is allowed.

Now, if you use any of the other special characters you will get an error that says cannot assign to the operator. So, though underscore is allowed, it is better to not begin or end with an underscore and that is a common unaccepted naming convention, though it is accepted by the compiler. So, if you begin or end with an underscore you are not likely to get an error. So, it is usually not a practice that is followed.

(Refer Slide Time: 02:46)

The slide has a blue header bar with the GITAA logo and the text 'Transforming careers'. The main title 'Naming conventions' is in bold blue text. Below the title is a bulleted list of commonly accepted case types:

- Commonly accepted case types
 - Camel (lower and upper)
ageEmp=45 AgeEmp=45
 - Snake
age_emp=45 Age_emp=45
 - Pascal
AgeEmp=45

At the bottom left, there is a small text 'Python for Data Science'.

So, there are a few case types that are commonly accepted while naming variables, the first is the camel case. It can be lower or upper. First example, that you have here where age of the employees given with E in capital. Now, this is a lower camel case, so the example on the right is an example for the upper camel case. Now, in this example the letter 'a' in age of the employee begins with the capital letter.

So, the next case type is the snake case where I am separating age and emp by an underscore. So, an underscore can be used between two sets of letters or between two letters and that is a snake case. The letter after the underscore should always be in lower case, but the first letter of the variable name can be in lower or upper case.

The next case type is Pascal; so, where I am again going to take AgeEmp. Now, here the first letter of age is in uppercase and the first letter of emp is also in uppercase. Now, these are the commonly used case types in Python. Now, the compilers is not going to throw an error if you name the variables wrongly, but these are convention that are in books and are accepted. However, there are other case types which use hyphen, but those case types will not be allowed in Python.

(Refer Slide Time: 04:06)

The screenshot shows a Jupyter Notebook interface. At the top, the title "Assigning values to multiple variables" is displayed. Below it, under the "Code" section, is the Python code: `Physics,Chemistry,Mathematics=89,90,75`. Under the "Values reflected in environment" section, a "Variable explorer" table shows the assigned values:

Name	Type	Size	Value
Chemistry	int	1	90
Mathematics	int	1	75
Physics	int	1	89

A woman is visible in the background of the slide, appearing to be speaking or presenting.

So, if you want to assign multiple values to a variable you can create all the variable at once and sequentially assign the value. Now, if you run this command you will see that the values of variables chemistry mathematics and physics have been assigned accordingly.

(Refer Slide Time: 04:24)

The image shows a video frame of a woman with long dark hair, wearing a plaid shirt, speaking. Above her, a white bar contains the text 'Data types' in blue. In the top right corner of the slide area, there is a logo for 'GITAA Transforming careers' featuring a stylized book icon.

So, now let us look at the commonly used data types in Python.

(Refer Slide Time: 04:27)

The image shows a slide titled 'Basic data types' in blue text. Below the title is a table comparing five basic data types in Python. The table has four columns: 'Basic data types', 'Description', 'Values', and 'Representation'. The rows are as follows:

Basic data types	Description	Values	Representation
Boolean	represents two values of logic and associated with conditional statements	True and False	bool
Integer	positive and negative whole numbers	set of all integers, \mathbb{Z}	int
Complex	contains real and imaginary part ($a+ib$)	set of complex numbers	complex
Float	real numbers	floating point numbers	float
String	all strings or characters enclosed between single or double quotes	sequence of characters	str

At the bottom left of the slide, it says 'Python for Data Science'. At the bottom right, there is a small number '9'.

So, the basic data types are Boolean which represents two values of logic and are associated with the conditional statement. So, the output values that you would get, when you use a Boolean data type is true or false, and it is represented as bool. The next data type is integer. It consists of set of all integers which are positive or negative whole numbers. It is represented by the letters int. The next data type is complex which

contains real and imaginary part. So, any expression of the form $a+ib$ is of a complex data type.

It consists of all complex numbers and it is represented by complex. Now, float data type consists of all real numbers which are of floating point numbers, it is represented by float. String data type consists of all strings and characters, so anything that is enclosed between single or double quotes is treated as a string data type. The value that is enclosed between the quotes can be a number, a special character, alphabets anything. So, anything that is contained within quotes is treated as a string data type. It is represented by the letters str.

(Refer Slide Time: 05:38)

Statistically vs dynamically typed language



- Statistically typed language
 - Type of variable is known at compile time
 - Type of variables declared upfront
 - Eg- Java, C, C++
- Dynamically typed language
 - Type of variable known at run time
 - Variable type need not be declared
 - Eg- Python, PHP

Python for Data Science



Now, before we go ahead with data type operations, it is important to know the difference between statistically and a dynamically typed language. So, a statistically typed language is the one, where the type of the variable is known at the compile time and you also have to declare the data type of the variables upfront. So, examples of such programming languages are C, C++ and Java.

So, contrary to this a dynamically typed language is the one where the data type need not be declared upfront, the type of the variable is known only at the run time. So, whenever you declare a variable and you assign a value of it; the moment you run that specific line, the data type of the variable is known. Examples of such languages are Python, PHP. So, Python that we are using here is a dynamically typed language.

(Refer Slide Time: 06:29)

Identifying object data type

- Find data type of object using
- Syntax: `type(object)`

Employee_name="Ram"
Age=55
Height=150.6

In [10]: `type(Employee_name)`
Out[10]: str

In [11]: `type(Age)`
Out[11]: int

In [12]: `type(Height)`
Out[12]: float

Checking the data type of an object

Python for Data Science

So, let's go further and see how to identify the data type of an object. Now, to find the data type of an object you will be using the function `type` and give the object as an input. Now, an object can be a variable, can be any of the data structures, it can be array or anything. In this case in particular, we are just going to look at how to find the data type of variables. Now, let us take a small example here. I have 3 variables here, the first is `Employee_name` which is the name of the employee and the value is "Ram".

Next is age of the employee presented by the variable name `Age`, the value for which is 55. The third variable is height, which is the height of the employee whose value is 150.6. Now, if you want to check the data type you can give `type` of `Employee_name`, now `Employee_name` is a string because it is enclosed between double quotes. If you give `type` of `Age`, `Age` is an integer and hence the output is int and if you give `type` of `Height` which is a floating point number the data type is float.

(Refer Slide Time: 07:40)

Verifying object data type



- Verify if an object is of a certain data type
- Syntax: **type(object) is datatype**

```
Employee_name="Ram"  
Age=55  
Height=150.6  
  
In [13]: type(Height) is int  
Out[13]: False  
  
In [14]: type(Age) is float  
Out[14]: False  
  
In [15]: type(Employee_name) is str  
Out[15]: True
```

Python for Data Science

12

So, now, let us see how to verify the data type of an object. Now, if you want to verify if an object belongs to a certain data type, then you basically give type of the object in this case it is a variable name, followed by the keyword is and the data type name. So, this data type will basically be the representation of the data type; int is for integer, str is for string so on and so forth. Now, I am going to use the same example that we have used earlier. Now, here I would like to verify if height is of integer data type. So, I am giving type of height followed by is and the keyword int which is the abbreviation for integer data type.

So, one important thing to keep in mind while verifying the data type is that in this case the output is just going to be Boolean. So, the output is going to be true or false. We are actually checking given a variable does it belong to a desired data type or not. We are not trying to assign it or change it, but what we are just trying to do is just cross verification. So, hence the output for this is either going to be true or false. So, I want to know if Age belongs to a float data type, which is false, because Age is actually an integer. Now, in this case I want to check if *Employee_name* is a string. Yes, the output is true because it is a string.

(Refer Slide Time: 09:02)

Employee_name="Ram"
Age=55
Height=150.6

In [16]: type(Height)
Out[16]: float

In [17]: ht=int(Height)

In [18]: type(ht)
Out[18]: int

In [19]: Height=int(Height)

In [20]: type(Height)
Out[20]: int

So, till now we have seen how to find the data type of an object. We have also seen how to verify if an object has a desired data type. Now, we are going to look at how to coerce object to new a data types. Now, if I want to convert the data type of an object to another, I will be using the data type, you need to replace data type with the abbreviation used for each of the data types and you give the object as the input here. Now, all the changes that have been made to the variables can be stored in the same variable or different variable.

Now, in this case height is of a float data type, let us say I want to convert it to an integer data type. So, I say `int(Height)` which converts it into an integer data type, but if I want these changes to be reflected in my environment, I have to store it on to a variable. Now, the variable can have the same name or a different name. So, in this case I am converting height to an integer data type and I am storing it onto a new variable called `ht`. Now, if I get the type of `ht` the output is `int`, which means height has been converted to an integer data type.

Now, if I want to reflect the changes on the same variable name, I will just say `Height=int(Height)`. So, earlier height was float, now once the operation is done if you check the type of height again the output that you will be getting is `int`. So, this is how you would coerce existing objects to new a data types.

(Refer Slide Time: 10:37)

The slide shows a Jupyter Notebook interface with the title "Coercing object to new data type".

Code examples:

- In [20]: `type(Salary_tier)`
Out[20]: `str`
- In [21]: `Salary_tier=int(Salary_tier)`
- In [24]: `type(Salary_tier)`
Out[24]: `int`

Annotations:

- A callout box labeled "Coercing the data type of an object" points to the code `Salary_tier=int(Salary_tier)`.
- A watermark "Python for Data Science" is visible at the bottom left.
- A page number "14" is visible at the bottom right.

So, one important point here is that only few types of coercions are accepted. Now, let us say I have a variable call `Salary_tier`, which is a string data type. Now, `Salary_tier` contains an integer which is enclosed between single quotes. So, in this case the `Salary_tier` is basically a number which is 1, but that is enclosed within single quotes so it is still treated as a string data type.

Now, if I want to change it to an integer, I will just say `Salary_tier=int(Salary_tier)`. Now, this will convert the data type for `Salary_tier`. So now, the data type is converted from string to an integer.

(Refer Slide Time: 11:15)

Coercing object to new data type



- However if the value enclosed within the quotes is a string then conversions will not be possible

```
In [19]: Employee_name="Ram"  
In [20]: Employee_name=float(Employee_name)  
Traceback (most recent call last):  
  File "<ipython-input-20-b0286eb77de0>", line 1, in <module>  
    Employee_name=float(Employee_name)  
ValueError: could not convert string to float: 'Ram'
```

Python for Data Science



However, not all types of coercion is possible. For instance, if I would like to convert the *Employee_name*, Ram, to an integer or a float I will be getting an error. So, the value which is enclosed between codes, if it is a float or a integer type then such types of coercion will be possible. However, if it is a string or a set of characters than those types of coercion would not be possible.

(Refer Slide Time: 11:42)

Summary



- Conventions to name a variable
- Basic data types
 - Get data type of a variable
 - Verify if a variable is of a certain data type
 - Coerce variable to new data type

Python for Data Science



So, to summarize in this lecturer we looked at some of the common naming conventions to name a variable in Python. We saw some of the basic data type related operations, one

of that was to get the data type of a variable. Then, verify if a variable is of a certain data type and how to coerce the data type of an existing variable to a newer one.

Thank you.

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 06
Operators

(Refer Slide Time: 00:17)

In this lecture



- Operators and operands
- Different types of operators
 - Arithmetic
 - Assignment
 - Relational or comparison
 - Logical
 - Bitwise
- Precedence of operators



Welcome, to the lecture on Operators! In this lecture, we are going to see what an operator is and what an operand is. We will also look at the different types of operators that is used in python and these are arithmetic, assignment, relational, logical and bitwise. We will also be looking at the precedence of operators and how to use them in an expression.

(Refer Slide Time: 00:37)

Operators and operands



- Operators are special symbols that help in carrying out an assignment operation or arithmetic or logical computation
- Value that the operator operates on is called operand

In [1]: $2+3$
Out[1]: 5

Python for Data Science



So, let us see what operators and operands are. An operator is a special symbol that will help you carrying out an assignment operation or some kind of a computation the nature of the computation can be arithmetic or logical.

Now, the value that the operator operates on is called an operand. So, let us take a small example here to illustrate what an operator and operand is. In this case, the plus symbol that you see in the plus sign that is an operator. This operator denotes addition. So, you see two numbers before and after the operator. So, 2 and 3 in this case are called operands.

(Refer Slide Time: 01:19)

The slide has a blue header bar with the text 'Arithmetic operators' in white. In the top right corner, there is a logo for 'GITAA' with the tagline 'Transforming Careers'. Below the header, there is a bulleted list:

- Used to perform mathematical operations between two operands
- Create two variable a and b with values 10 and 5 respectively

Below the list, there is a code snippet:

```
a=10    b=5
```

Symbol	Operation	Example
+	Addition	In [3]: a+b Out[3]: 15
-	Subtraction	In [4]: a-b Out[4]: 5

At the bottom left of the slide, it says 'Python for Data Science'. On the right side of the slide, there is a video player showing a woman in a pink shirt speaking.

So, let us look at arithmetic operators. Now, arithmetic operators are used to perform mathematical operations between any two operands. So, let us take an example to illustrate the use of arithmetic operators. Create two variables a and b with values 10 and 5. In the previous lectures we have already seen how to create a variable. So, we were going to use the same procedure here.

So, the first operator that we are going to look at is the addition operator. It is denoted by a plus symbol and this is how an addition operation is carried out. If I want to add two variables, then I am just going to separate the variables with the plus symbol in this case a is 10, b is 5 so, the result in sum that you get is 15. The next operation is subtraction denoted by a hyphen. I have the corresponding example here. So, in this case I am going to subtract a and b and the output that I get here is 5.

(Refer Slide Time: 02:17)

Symbol	Operation	Example
*	Multiplication	In [5]: a*b Out[5]: 50
/	Division	In [6]: a/b Out[6]: 2.0
%	Remainder	In [8]: a%b Out[8]: 0
**	Exponent	In [7]: a**b Out[7]: 100000

Python for Data Science

So, the next arithmetic operation that we are going to look at is the multiplication operation, it is denoted by an asterisk and if you want to multiply two variables separate the variable and insert an asterisk symbol. The product that you get in this example is 50; a is 10, b is 5 again. Now, 10 into 5 is 50 and that is the output that you get .

The next operation is division that is denoted by a forward slash. So, you separate the variables and insert a forward slash between them, what you get is basically the quotient. So, in this case 10 by 5 gives you a quotient of 2 which is the output here. So, the next operation is getting a remainder and it is denoted by the percentage symbol. So, let us take the same example here. I am trying to get the remainder when I divide a and b and you just separate the variables and insert the percentage symbol and that returns the remainder. In this case a is 10 and b is 5. So, 10 is divisible by 5 and hence I get a remainder of 0.

The next operation is exponent and it is denoted by double asterisk. So, let us say if I want to raise a variable to the power of another variable then I am going to use this operation. So, let us say in this case I want to raise a to the power of b, then I just say a double asterisk b. So, in this case since we have a as 10 and b as 5, I am just going to raise 10 to the power of 5 and the corresponding output that you get here is 1 lakh.

(Refer Slide Time: 03:51)

The screenshot shows a presentation slide with the title "Hierarchy of arithmetic operators". Below the title is a table with two columns: "Decreasing order of precedence" and "Operation". The operations listed from top to bottom are: Parentheses, Exponent, Division, Multiplication, and Addition and subtraction. To the right of the table is a mathematical expression $A = 7 - 2 \times \frac{27}{3^2} + 4$. Below the expression are two code snippets: "In [10]: A=7-2*(27/3**2)+4" and "In [11]: print(A)". The output of the second snippet is "5.0". At the bottom left of the slide, it says "Python for Data Science". On the right side of the slide, there is a photograph of a woman in a red shirt standing at a podium.

Decreasing order of precedence	Operation
Parentheses	()
Exponent	**
Division	/
Multiplication	*
Addition and subtraction	+,-

$A = 7 - 2 \times \frac{27}{3^2} + 4$

In [10]: `A=7-2*(27/3**2)+4`

In [11]: `print(A)`

5.0

Python for Data Science

So, let us look at the hierarchy of operators. Now, I have ordered the operators in the decreasing order of precedence. So, the first is parentheses. So, parentheses is not really an operator, but anything that is enclosed within parentheses gets the topmost priority. So, therefore, I have included parenthesis also as an operation. Now, this is followed by exponential operations and then division, multiplication and addition and subtraction are given the same precedence. So, let us take an example here in this case.

I have the following expression. I have

$$A = 7 - 2 * \frac{27}{3^2} + 4$$

So, to avoid confusion I am going to add bracket for this 27 by 3 square term. So, 27 is the numerator, the denominator is 3 square and to denote 3 square I add a double asterisk here and the entire term I am enclosing it within parentheses. So, once you execute the command in your console, if you print the value of A, it should return 5. So, this is how you will use the operators in an expression. You can also try out this example.

(Refer Slide Time: 05:03)

Assignment operators



- Used to assign values to variables

Symbol	Operation	Example
=	Assign values from right side operands to left side operand	a=10 b=5 .
+=	Adds right operand to left operand and stores result on left side operand (a=a+b)	In [55]: a+=b ...: print(a) 15
-=	Subtracts right operand from left operand and stores result on left side operand (a=a-b)	In [57]: a-=b ...: print(a) 5

Python for Data Science

So, now let us look at assignment operators. So, an assignment operator is used to assign a value to a variable. So, the first assignment operator that we are going to look into is the equal to symbol. Now, this is the most commonly huge assignment operation and that is because whenever you want to create a variable you want to assign a value to it. So, we have learnt how to use this operator in our earlier lectures.

So, what it basically does is that it will assign values from the right side operand to the left side operand. Now, the left side operand is a variable name and the right side operand is the value that is given to the variable. So, let us take an example in this case. I am retaining the same variable names a and b, I am assigning a value of 10 to a. In this case 10 is my right side operand and a is my left side operand. So, the same definition also holds good for b equal to 5.

So, the next operator that we are going to look into is the `+=` operator. So, what it basically does is that, it first adds the right operand to the left operand and then it will store the result on the left side operand. Now, let us take the same variables a and b. Now, if I were to denote `a += b` then this would translate to `a = a + b`. I have indicated that within parentheses here. So, whenever I give `a += b`, then I am saying `a = a + b`, so, the first operation that happens is the addition operation which is `a + b`. So, the value of `a + b` gets stored in `a` and hence you can see that the value of `a` gets updated to 15 a was earlier 10, and now the value gets updated to 15.

The next operator is the minus equal to operator. So, this is also similar to the addition operator that we earlier saw. It basically subtracts the right operand from the left and it will store the result on the left side operand. Now, whenever I give $a -= b$ then it translates to $a = a - b$. Now, whenever I compute the difference in this case I am getting a difference of 5 and that is what I am printing.

(Refer Slide Time: 07:19)

The screenshot shows a presentation slide with a blue header containing the text 'Assignment operators'. In the top right corner, there is a logo for 'GITAA' with the tagline 'transforming careers'. Below the header, there is a table with three columns: 'Symbol', 'Operation', and 'Example'. The table contains two rows. The first row corresponds to the asterisk operator (*=) and the second row corresponds to the forward slash operator (/=). The 'Python for Data Science' watermark is visible at the bottom of the slide.

Symbol	Operation	Example
$\ast=$	Multiples right operand from left operand and stores result on left side operand ($a=a\ast b$)	In [59]: a*=b ...: print(a) 50
$/=$	Divides right operand from left operand and stores result on left side operand ($a=a/b$)	In [62]: a/=b ...: print(a) 2.0

So, asterisk operator will multiply the right operand from the left and will store the result on the left operand. Now, in this case again I am going to retain the same values of a and b; a is 10, b is 5 which means I am multiplying a and b first. So, $a * b$ will give me a product of 50. Now, whenever I print the value of a it will give me the updated value which is 50. Forward slash equal to means division. So, whenever I use this operator, I am going to divide the right operand from the left and store the value on the left operand.

Now, in this case $a /= b$ translates to $a = a / b$. Now, if you print the value of a you can see that the value of two has been updated 2 to from 10.

(Refer Slide Time: 08:09)

Relational or comparison operators



- Tests numerical equalities and inequalities between two operands and returns a boolean value
- All operators have same precedence
- Create two variables x and y with values 5 and 7 respectively

```
In [1]: x = 5  
In [2]: y = 7
```

Symbol	Operation	Example
<	Strictly less than	In [3]: print(x<y) True
<=	Less than equal to	In [4]: print(x<=y) True

So, now let us see what relational or comparison operators are. A relational operator will test for a numerical equality or an inequality between two operands. The value that a relational operator returns is Boolean in nature which means it will basically return true or false. Now, all the relational operators that we are going to look into have the same precedence which means they all have the same priority.

So, let us create two variables x and y. I am assigning a value of 5 to x and 7 to y. The first relational operation is the strictly less than operation. It is denoted by the angle operator with its tip towards the left. So, let us take an example and see how this operator works. Now, we already have the values for x and y. Now, I am just giving the relation $x < y$. So, now, what will happen is that it will check whether x is strictly less than y? In our case, yes, x is 5, y is 7. So, yes, 5 is strictly less than 7 and hence the output that you will see is true.

The next operation is the less than equal to operation. It is denoted again with the angled operator with its tip towards the left followed by an equal to symbol. Now, let us take an example. Now, in this case I am trying to print the output for $x \leq y$. So, in this case we are checking if $x < y$ or is $x = y$. So, these are the two conditions that we are going to check. So, x is of course, not equal to y, but x is less than y. So, that condition is satisfied and hence the output that you get is true.

(Refer Slide Time: 09:49)

Relational or comparison operators



Symbol	Operation	Example
>	Strictly greater than	In [5]: print(x>y) False
>=	Greater than equal to	In [6]: print(x>=y) False
==	Equal to equal to	In [7]: print(x==y) False
!=	Not equal to	In [8]: print(x!=y) True

Python for Data Science

So, similar to the strictly less than and less than equal to operator, you have the strictly greater than and greater than equal to operator. So, contrary to what a strictly less than operator does, in this case you are strictly checking whether x is greater than y. Of course, it is not and hence the output is false. For greater than equal to you basically are checking whether $x > y$ or is $x = y$. So, both the conditions are false anyways and hence the output that you get is also false. So, a strictly greater than operator is denoted by an angled operator with its tip to the right and a greater than equal to operator is denoted by an angled operator with its tip to the right followed by an equal to symbol.

The next operation is the equal to equal to operation. It is denoted by a double equal to symbol and what we really check when we give double equal to? We check if the left hand side operand is it exactly equal to the right hand side operand. In this case the value of x and y are 5 and 7 respectively and I am checking if 5 is exactly equal to 7 or not. No, it is not and hence the output is false.

The next operation is not equal to. It is denoted by an exclamation followed by an equal to symbol(!=). So, in this case the output is going to be true as long as x is not equal to y. So, this operator is frequently used when you are iterating through a loop and you want to run the loop or you want to iterate through the loop as long as a certain condition is obeyed. So, you can use not equal to in that case. Now, as long as x is not equal to y my output is always going to remain as true.

(Refer Slide Time: 11:31)

Logical operators



- Used when operands are conditional statements and returns boolean value
- In python, logical operators are designed to work with scalars or boolean values

Symbol	Operation	Example
or	Logical OR	In [9]: print((x>y) or (x<y)) True
and	Logical AND	In [10]: print((x>y) and (x<y)) False
not	Logical NOT	In [11]: print(not (x==y)) True

So, the next set of operators are the logical operators. A logical operator is used when the operands are conditional statements. The output for logical operators are Boolean in nature which means they return true or false. So, strictly from the point of view of python logical operators are designed to work only with scalars and Boolean values. So, if you want to compare two arrays, then a logical operator cannot be used. So, let us take the first logical operation which is logical or it is denoted by the letters or, both letters in lowercase.

Now, let us take a small example here, I am retaining the same values for x and y; x is 5, y is 7. Now, if I give $(x > y)$ or $(x < y)$, I get an output that says true. Now, why does it happen? So, a logical OR is designed to give an output true when one of the statement is satisfied. In this case $x > y$ it is not satisfied. So, it is a false, but however, $x < y$ that statement is satisfied. So, this gives you an output which is true. So, the inputs to the or operator is basically false and true. So, whenever you have a false and a true operand, then the resultant is always true. So, hence you are also getting an output which is true.

The next is the logical AND which is represented by the letters and, all letters again in lower case. Let us take a small example here, I am taking the same expressions that I have considered above. Now, in this case instead of or I am replacing them by and. So, for the same conditions I am getting a different output and in this case it is false. So, why does this happen? If you look at the conditional statements the first is $x > y$, we know

that this is a false statement it gives you a Boolean value of false. The second conditional statement is $x < y$. Now, this is true like I earlier said.

So, the way a logical AND works is that whenever you have a false and a true condition as the operands you will basically get a false output and this is because logical AND expects you to satisfy both the conditions and unless both these values are true it will never return the output as true. So, even when you have false true or true false the output is always false.

So, the next logical operator is not represented by the letter not, again in lowercase. So, not basically negates your statement. So, I have taken the example of $x == y$. Now, we know the value of x is 5, value of y is 7 of course, both of them are not equal. Now, the output that you will get from this conditional statement is false. So, we are trying to negate false which means not false. So, that gives you a result which is true. So, that is why you get the output as true.

Now, another important point to note in logical operators is that, whenever you giving these conditional or relational statements, make sure that you enclose them within parentheses because if you are not going to do it then you are likely to get an error.

(Refer Slide Time: 14:55)

Bitwise operators



- Used when operands are integers
- Integers are treated as a string of binary digits
- Operates bit by bit
- Can also operate on conditional statements which compare scalar values or arrays
- Bitwise OR (|), AND(&)

So, let us move on to bitwise operators. So, bitwise operators are used when the operands are integers. So, these integers are treated as a string of binary digits and are binary

encoded. So, when you are going to use a bitwise operator on two integers which are binary coded, the operator is going to compare bit by bit of the binary code and that is how the operator got its name bitwise.

The other advantage of using a bitwise operator is that, they can operate on conditional statements. Now, these conditional statements can compare scalar values or they can also compare arrays. Now, if you would like to compare arrays you would be using a bitwise operator. We earlier saw that we cannot use logical operators to handle arrays and this is where bitwise operators step in.

So, throughout the course we are going to be looking into two bitwise operators. The first says bitwise OR which is represented by a pipe and second operator is the bitwise AND represented by an ampersand.

(Refer Slide Time: 15:53)

The slide has a blue header bar with the GITAA logo and the text 'Transforming Careers'. The main title 'Bitwise operators' is centered in a large blue font. Below the title, there is a bulleted list of points. On the right side of the slide, there is a video frame showing a woman in a pink shirt speaking. At the bottom left, it says 'Python for Data Science'.

- Create two variables x and y with values 5 and 7 respectively
- In [1]: x = 5
In [2]: y = 7
- Binary code for 5 is **0000 0101** and for 7 is **0000 0111**
- 0 corresponds to **False** and 1 corresponds to **True**
- In bitwise OR (|), operator copies a bit to the result if it exists in either operand
- In bitwise AND (&), operator copies a bit to the result if it exists in both operands

So, create two variables x and y with values 5 and 7. Now, these are the binary codes for 5 and 7 and we are going to be using these variables for our example. So, 0 corresponds to false and 1 corresponds to true. And, in a bitwise OR the operator will copy bit by bit of the result if it is there in either of the operands. But, in a bitwise AND the operator will copy the bit only if it exists across both the operands.

So, let us take an example and see what these two statements mean.

(Refer Slide Time: 16:23)

Code and output in console

In [47]: $x \mid y$
Out[47]: 7

Binary code for 5

Binary code for 7

Python for Data Science

Now, I am going to be illustrating a bitwise OR on integers. Now, I am using the bitwise OR operator which is a pipe symbol between x and y ; x and y are my operands. The output that you will be getting is 7. So, let us see how this output was achieved.

So, I have created two arrays here. The cells in these arrays consists of the individual binary code for 5 and 7, and I have color coded them for reference. So, the first two positions of both the binary codes is 0. So, both these serve as my input operands. Now, both these positions have 0 and hence the resultant will also contain 0.

(Refer Slide Time: 17:07)

Bitwise OR on integers

Binary code for 5

Binary code for 7

- 0 present in positions 2-5, therefore resultant cell will also contain 0
- In the 6th position, 1 is present in both operands and hence resultant will also contain 1

Python for Data Science

Now, let us take the second position. The second position also has 0 for both the binary codes and hence my corresponding position in the resultant binary code is also going to be 0. I have highlighted the positions using circles to just show you which cells I am referring to.

So, now you can also see that positions 3, 4 and 5 consists of 0's for both the binary codes. So, hence the corresponding positions of the resultant binary code will also contain 0. Another important point to note is that the sixth position of both the binary codes consists of 1. So, the binary code for 5 and the binary code for 7, in both of these codes the sixth position corresponds to 1 and hence in the resultant binary code I am copying 1 for the sixth position.

(Refer Slide Time: 17:59)

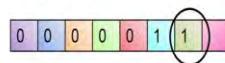
Bitwise OR on integers



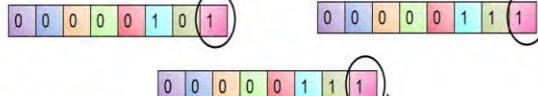
- The 7th position has 0 in the first operand and 1 in the second



- Since this is an OR operator, only the True condition is considered



Binary code for 5 Binary code for 7



28

Moving further, if you compare the 7th position for both these binary codes you can see that for the binary code 5 the 7th position has 0 and for the binary code 7, the 7th position has 1. So, in this case there is a difference in values between both these operands. So, we are going to see how to fill in the corresponding position of the resultant binary code.

So, since we are using an OR operator, when one of the condition is true the resultant always becomes true. In this case, if you can recall so, like I earlier said 0 corresponds to false and 1 corresponds to true so, we have one true condition. So, the resultant will also

contain the true value which is 1. So, an OR operator will give you the output as true when one of the operands is true.

Now, we are left with the last position and for both these binary codes the last position is 1 and I am going to be copying this value to the corresponding position in my resultant binary code. So, this is the binary code that you get when you apply a bitwise OR between two integers. This is the binary code for 7 that we earlier started with and this is how a bitwise or operator works between two integers.

(Refer Slide Time: 19:17)

Bitwise operators



- Bitwise operators can also operate on conditional statements

Symbol	Operation	Example
	Bitwise OR	In [9]: print((x<y) (x==y)) True



We can also use bitwise operators for conditional statements. Now, if I were to use the bitwise OR for a relational statement, this is how it could look. So, I am giving two conditional statements here. The first is where I am checking if x is less than y; the second is where I am checking if x is equal to equal to y.

Now, I am in this case x is less than y that results in a value which is true and whereas, the second conditional statements which is x equal to equal to y will result in false. In this case, the first condition is true and hence the output that you get is also true.

(Refer Slide Time: 19:53)

The slide has a blue header with the GITAA logo and the text 'Transforming careers'. The title 'Precedence of operators' is centered above a table. The table has two columns: 'Decreasing order of precedence' and 'Operation'. The operations listed from top to bottom are: Parentheses, Exponent, Division, Multiplication, Addition and subtraction, and Bitwise AND.

Decreasing order of precedence	Operation
Parentheses	()
Exponent	**
Division	/
Multiplication	*
Addition and subtraction	+,-
Bitwise AND	&

At the bottom left of the slide, it says 'Python for Data Science'.

A woman in a red shirt is visible on the right side of the slide, appearing to be speaking.

So, now let us look at the precedence of operators, I have ordered the operators in the decreasing order of precedence. So, like I earlier mentioned parenthesis is not an operator. Now, any expression with operators that are enclosed within parentheses they get the topmost priority. So, that is why parentheses always occupy the first line in terms of precedence.

Now, after parentheses I have the exponential operation followed by division, multiplication, addition and subtraction are given the same precedence, I then follow it up with my bitwise AND bitwise OR, all relational operators are given the same precedence.

(Refer Slide Time: 20:31)

Precedence of operators

Decreasing order of precedence	Operation
Bitwise OR	
Relational/comparison operators	==, !=, >, >=, <, <=
Logical NOT	not
Logical AND	and
Logical OR	or

Python for Data Science



And, then comes the logical NOT, logical AND, and logical OR. So, this is the decreasing order of precedence for all the operators put together.

(Refer Slide Time: 20:45)

Summary

- Important operators
 - Arithmetic
 - Assignment
 - Relational
 - Logical
 - Bitwise

Python for Data Science



So, to summarize in this lecture we saw what are the important operators. We looked at what arithmetic, assignment, relational, logical and bitwise operators do. We also took an example in each of these case to illustrate how the operator works and what is the nature of the output.

Thank you.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 13
Reading Data

(Refer Slide Time: 00:18)

The image shows a presentation slide with a blue header bar. On the left, the text 'In this lecture' is displayed in blue. On the right, there is a logo for 'GITAA' with a stylized book icon. Below the header, a bulleted list of topics is shown:

- File formats
- Commonly used file formats
- Read data from
 - .csv format
 - .xlsx format
 - .txt format

At the bottom of the slide, there is a video player interface showing a woman speaking. The video player has a blue progress bar at the bottom. The text 'Python for Data Science' is visible at the bottom left of the video area.

Hello, all welcome to the lecture on Reading Data. So, in this lecture we are going to look different file formats and what are the commonly used file formats, how do we read them into Spyder and the commonly used file formats that we will be looking into this lecture is dot csv format, the dot xlsx format and the .text format. We will see in detail about each and every data format on how to read them into Spyder.

(Refer Slide Time: 00:43)

File format

- Standard way in which data is collected and stored
- Most commonly used format for storing data is the spreadsheet format where data is stored in rows and columns
 - Each row is called a record
 - Each column in a spreadsheet holds data belonging to same data type
- Commonly used spreadsheet formats are comma separated values and excel sheets
- Other formats include plain text, json, html, mp3,mp4 etc.

Python for Data Science

So, now we will see what is the format of file. It is just a standard way in which the data is collected and stored and we know that most commonly used format for storing the data is the spreadsheet format, where the data will be stored in rows and columns and where you see each row will be called as a record or a sample and each column will be represented as a variable.

And each column in a spreadsheet holds data belonging to the same data type and the commonly used spreadsheet formats or comma separated values and excel sheets there are so, many other formats where we can do analytics on that. But since this course is driven towards data science, we will be predominantly looking at how to work with comma separated values and excel sheets.

So, this lecture will be driven towards the file formats like text, comma separated values in the excel sheets. As I mentioned there are other formats like plain text files, json files, html files, mp3 files, mp4 etcetera. So, all these formats can be read into Spyder and you can do any analysis on that based on the requirement but in this course, we will be looking at how to deal with the spreadsheet formatted files like comma separated values in excel sheets.

(Refer Slide Time: 02:03)

Comma separated values

GITAA

- Spreadsheet format
- Format '**.csv**'
- Each record is separated by a comma
- Files where records are separated using a tab are called tab separated values
- .csv files can be opened with notepad or Microsoft excel

Python for Data Science

So, now let us see comma separated values. Comma separated values can be stored in a spreadsheet formats and the format being represented here is csv, where it is just the abbreviation of comma separated values. So, whenever you see a spreadsheet with extension .csv then you call them as comma separated values. So, in this case each record is separated by a comma, but there are other ways where the record can also be separated. So, the files where records are separated using the tab or called tab separated values and the .csv files can also be opened with a notepad or the Microsoft excel.

(Refer Slide Time: 02:45)

iris_data_sample.csv - Notepad

File Edit Format View Help

,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species

1,5.1,3.5,1.4,0.2,Iris-setosa

2,4.9,,1.4,0.2,

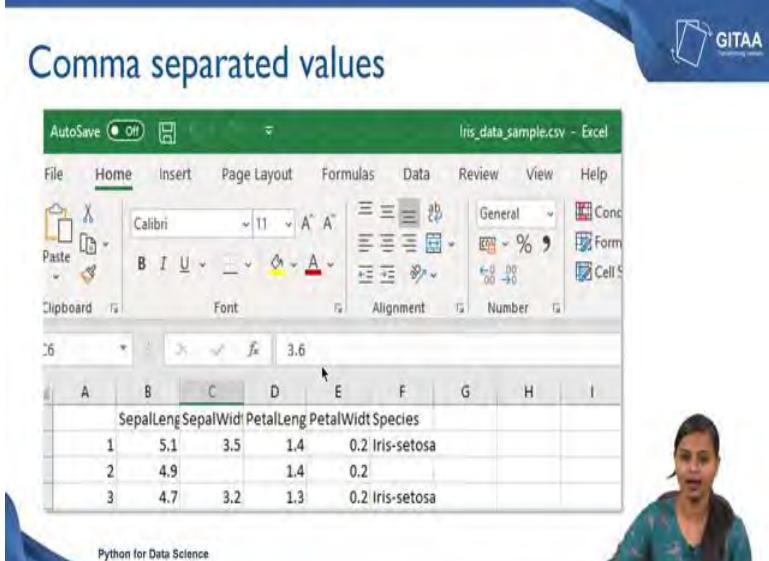
3,4.7,3.2,1.3,0.2,Iris-setosa

4,??,3.1,1.5,0.2,Iris-setosa

Python for Data Science

So, let us see how it looks whenever we open the comma separated values in a notepad. So, this is how it will look like whenever you open the csv files, the .csv files using the notepad. So, you see all the records here, but you will not be able to differentiate between which is the row and which is the column and which cell belongs to which variable. So, that is the problem with viewing the .csv file using the notepad. However, you can also view the .csv file using the notepad as well as excel sheet.

(Refer Slide Time: 03:19)



The screenshot shows a Microsoft Excel window titled "Comma separated values". The ribbon menu is visible with "File", "Home", "Insert", "Page Layout", "Formulas", "Data", "Review", "View", and "Help". The "Home" tab is selected. The font toolbar shows "Calibri" at size 11, bold, italic, underline, and various alignment options. The formula bar shows "iris_data_sample.csv - Excel". The main area displays a table with the following data:

	A	B	C	D	E	F	G	H	I
	SepalLeng	SepalWid	PetalLeng	PetalWidt	Species				
1	5.1	3.5	1.4	0.2	Iris-setosa				
2	4.9		1.4	0.2					
3	4.7	3.2	1.3	0.2	Iris-setosa				

Below the table, a watermark reads "Python for Data Science". In the bottom right corner of the slide, there is a small video player interface showing a woman speaking, indicating this is a recorded video.

So, now we will see how it looks like whenever you open any .csv files in a excel sheet. The same data is being stored in a different format like every data has been stored in a tabular fashion, where it has been represented in terms of rows and columns and each row is representing samples and each columns are representing the variables. Here we have few variables like sepal length, sepal width, petal length, petal width and the other being the species and you have three records being shown up here.

So, now, we have seen about the comma separated values and what is the extension to it and we have seen how to open those comma separated values using notepad and excel sheet.

(Refer Slide Time: 04:08)

Excel spreadsheets

- Spreadsheet format
- Part of Microsoft Office
- Format **'xlsx'**

Python for Data Science

Next we will move on to excel spreadsheets. It is also a spreadsheet format it is a part of Microsoft office. So, if you have a Microsoft Office in your machine then you will be able to work with excel and the format would be .xlsx. So, whenever you save any spreadsheet with the extension .xlsx, then that becomes the excel spreadsheets. So, this is the snippet on how it will look like whenever you save any records using the .xlsx extension.

So, this also gives you the same flavor on how you open your csv file using excel. All the variables have been represented in terms of columns and all the samples have being represented in terms of rows.

(Refer Slide Time: 04:31)

A screenshot of Microsoft Excel showing the 'iris_data_sample.xlsx' file. The spreadsheet has four columns: SepalLength, SepalWidth, PetalLength, and PetalWidth, with one additional column for Species. The first four rows of data are visible:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa

(Refer Slide Time: 04:57)

A screenshot of Notepad showing the contents of 'SherlockHolmes.txt'. The text is the Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle. A portion of the text is highlighted in red, showing the license information:

This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.net.

Next we will move on to text format. Till now we have been covering about the spreadsheet formatted files where we have seen the two types; one is comma separated values and the other one is being excel sheets. Now, we are going to move to text formats. Text formats basically consists of plain text or records. So, whatever data which consists of plain text or records, we call them as text formatted data and the format would be with .txt extension.

So, whenever you save any file with the extension .txt, then that becomes the text formatted data. So, this is the snippet to show you how a text formatted data will look like. So, in the previous example you been shown up for the data where you have rows and columns. Text file also can contain only plain text but in this lecture, we will be completely focusing on how to read data from different formats like csv and excel.

(Refer Slide Time: 05:55)

The screenshot shows a Python code editor with the title "Importing data into Spyder". The code being typed is:

```
import os
import pandas as pd
```

Annotations explain the imports:

- `import os` → 'os' library to change the working directory
- `import pandas as pd` → 'pandas' library to work with dataframes

Below the code, the command `os.chdir("D:\\Pandas")` is shown, with the path highlighted in green. A video player interface at the bottom right shows a woman speaking, with the text "Python for Data Science" below it.

So, now we will see how to import the data. So, we are going to look how to import data into Spyder. So, before importing data, you have to load or import the necessary libraries. The first library being os, in order to import any library we use the command import followed by the library name. So, we import os library to change the working directory. There might be cases where you want to work with some data in that case you can basically change your working directory to wherever you have saved your data.

So, that is why we have imported the os library and then we are importing the pandas library. So basically, we input pandas library to work with data frames. Whenever we read any data into Spyder, that becomes a data frame. That is what we call it as a data frame, where the data frames being represented in terms of a tabular fashioned data where each row will be represented as sample and each column will be represent are so variable.

So, that is why we are importing the pandas library as pd. So, here pd is just analyze to find out. So, whenever I want to access any function from the pandas library, I can just use pd with the .operator I can use the other related functions. So, that is why I have used pandas as pd. Next we have imported the os library so, using the os library you are using the .operator I am using the function called chdir which stands for changing directory and inside the function you just need to give the path of your file, wherever it is lying on your drive or in your system.

(Refer Slide Time: 07:37)

The slide has a blue header with the text 'Comma separated values' and the GITAA logo. Below the header is a code block:

```
• Importing data  
data_csv=pd.read_csv('Iris_data_sample.csv')
```

Below the code is another bullet point:

- Blank cells read as 'nan'

Next is a screenshot of a Jupyter Notebook cell displaying a DataFrame:

Index	Unname#0	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-set...
1	2	4.9	nan	1.4	0.2	nan
2	3	4.7	3.2	1.3	0.2	Iris-set...
3	4	??	3.1	1.5	0.2	Iris-set..
4	5	5	3.6	###	0.2	Iris-set...

At the bottom left of the slide, it says 'Python for Data Science'.

So, now let us see how to read a csv file into Spyder. So, read_csv is the command that is used to read any csv files into Spyder and that comes from the package called pandas. I have used pd.read_csv. Inside the function you just need to give the file name and the file name being Iris_data_sample and I have given the file name with the extension .csv and I have also given the file name within single quote.

So, these two things are very mandatory to read any file into Spyder. One is the extension and one is single or double quotes and the other is enclosing the file name inside the single or double quotes and if you see whenever I am reading, I am saving into an object called data_csv. Now data_csv becomes data frame whenever you execute this line and whenever you read any data into Spyder, all the blank cells will be read as nan because there might be cases where you have some missing values in your data as a blank value. So, in that case the Python will default all the blank values to nan. so that

whenever you are going to use any function which is going to describe how many missing values are there in your record, then the nan will account for that.

So, this is just a screenshot of the data whenever you open it from your variable explorer tag. So, you have so many variables here starting from index to species and index being represented as the row labels starting from 0 to 4 here from the snippet and you have another column called unnamed colon 0 where the values are starting from 1 to 5.

If you feel this represents an id to each of the rows, then you can keep this column unwanted 0 because you do not want too many columns which have the same information to it I want to keep the unnamed 0 column and I want to get rid of the index column. So, that I have some ids to each of the observation that can be useful for the future analysis.

(Refer Slide Time: 09:41)

Comma separated values

- Removing the extra id column by passing `index_col=0`

```
data_csv=pd.read_csv('Iris_data_sample.csv',index_col=0)
```

Index	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-set...
2	4.9	nan	1.4	0.2	nan
3	4.7	3.2	1.3	0.2	Iris-set...
4	??	3.1	1.5	0.2	Iris-set...
5	5	3.6	###	0.2	Iris-set...

- Replacing '??' and '###' as missing values

So, let us see how to remove the unwanted column. So, if you want to remove the extra id column, you can pass an argument called `index_col` is equal to 0 whenever you read a data frame. So, let us see how to do that. So, I have used the same command called `pd.read_csv` inside the function. I have given the file name followed by that I have given another argument called `index_column` is equal to 0. That means, that I am making my

first column as index column. In that case my index column would go and whatever I had it in the name of unnamed 0 that became the index column here.

So now, you have 6 variables where the first variable represents the serial number or id for each and every records in your data frame iris data sample. So, this is how we read a csv file and this is how we also set your index column by specifying the column number here. So, if you see here there are special characters other than the numerical values like question marks and hash.

These can be the representation of missing values, but I have mentioned that all the blank values will only be replaced with nan because in Python by default all the blank values will only be replaced with nans not, other special characters but if you are sure that these special characters just represents the missing values in your data and if you want to consider them as nan you can also do that by replacing all the question marks and hash as missing values.

(Refer Slide Time: 11:19)

The screenshot shows a video slide with the title "Comma separated values". The slide content includes a bullet point and two code snippets. The bullet point states: "Junk values can be converted to missing values by passing them as a list to the parameter 'na_values'". Below this, two code examples are shown in a code editor window:

```
data_csv=pd.read_csv('Iris_data_sample.csv',
                     index_col=0,na_values=['??'])
```



```
data_csv=pd.read_csv('Iris_data_sample.csv',
                     index_col=0,na_values=['??',"###"])
```

A woman is visible in the bottom right corner of the slide, appearing to be speaking.

So, let us see how to do that. So, if you see all the junk values can be converted to missing values by passing them as a list to the parameters na_values. So, this can also be done whenever you read a data frame into Spyder. So, we are following the same command followed by the index call argument. You have to just give na_values equal to a character which should be enclosed inside the square bracket that represents it is a list.

So, I have first given question mark here that should be given in inside the single or double quotes, then it takes it as a character and it will check those characters in the data frame and if there are any question marks in the data frame then it defaults that question marks to nan values. So, this is what we want to do now.

So, I have included na_values argument and inside the square bracket I have just given question marks. So, once I read this, I will be able to get rid of all the question marks and that will be converted as nan values. Say, but in our case there are so many special characters that are representing the missing values.

So, I can just give multiple values inside the square brackets so that all these values will be read as nan. So, here I have just added the hash by using a comma. So, all the values have been treated as list here and it will check for these two values and replace all these special characters with a nan value.

(Refer Slide Time: 12:51)

The slide is titled 'Excel spreadsheets'. It shows a code snippet for importing data from an Excel file:

```
data_xlsx=pd.read_excel('Iris_data_sample.xlsx',
sheet_name='Iris_data')
```

Below the code, there is a screenshot of an Excel spreadsheet titled 'Iris_data'. A callout box points to the 'Sheet name' tab at the bottom of the spreadsheet window. The Python logo is visible in the bottom right corner of the slide.

So, now this is how we replace all those special characters, and which represents your missing values into nan. So, read_excel is the command to read any excel sheet into Spyder and that also comes from the pandas library that is why we have used pd.read_excel inside the function I have just given the file name with the extension .xlsx.

So, here the same file I am using with different formats. So, the extension being .xlsx here and if you are dealing with excel spreadsheets, then there might be cases where you

have multiple tabs and a tabs being differentiated with different tab names. In that case if you want to access data from separate tab, then you give the sheet name under the `sheet_name` argument.

So, here `iris_data` is the sheet name from which I want to access the data frame. So, I have given `iris_data` under saved into an object called `data_xlsx`. Now, this becomes a data frame. So, there is a snippet from which you can get your sheet name. So, in your excel sheet you will have sheet names. So, you can just give the sheet name under the `sheet_name` argument.

(Refer Slide Time: 14:03)

The screenshot shows a slide titled "Excel spreadsheets". The slide content includes a bulleted list and two code snippets. The first bullet point is "Importing data" followed by a code snippet:

```
data_xlsx=pd.read_excel('Iris_data_sample.xlsx',  
sheet_name='Iris_data')
```

The second bullet point is "Remove index column and replace '??' and '# #' as missing values" followed by another code snippet:

```
data_xlsx=pd.read_excel('Iris_data_sample.xlsx',index_col=0,  
na_values=['??','###'])
```

A small video player window is visible in the bottom right corner, showing a woman speaking. The video player has a progress bar at the bottom.

And here also since we are using the same data with different formats here also you have the problem with the question marks and hash which represents the missing values. So, you can follow the same steps here to get rid of those special characters and to convert them as nan. So, let us do that also I have just included another argument called `na_values` where I have set the values to be converted as nan and I have also set my first variable as the index column. So, now, we have seen how to read excel file into Spyder.

(Refer Slide Time: 14:37)

The screenshot shows a Python code editor window. At the top, there's a header with the text "Text format" and the GITAA logo. Below the header, the code is displayed:

```
• Importing data
data_txt1=pd.read_table('Iris_data_sample.txt')
```

Underneath the code, a table shows the structure of the variable "data_txt1":

Name	Type	Size
data_txt1	DataFrame	(150, 1)

In the bottom right corner of the slide, there is a small video frame showing a woman speaking.

Next we are going to see how to import any text formatted data into Spyder. So, `read_table` is the command that is used to read any text format data that also comes from the pandas library and inside the function you just need to give the file name. Here the file name being `iris_data_sample`. The same file that we are working with and extension being `.txt`.

I have also saved my data frame as `data_txt1` and once you read it this will be shown up in your variable explorer where you can see the name of your object is `data_txt1` and the type of your object is data frame and the size being 150 rows and 1 column. So, 150 comma 1 represents rows and columns. So, there are 150 rows and only 1 column but we know that we had 5 variables in our data frame.

(Refer Slide Time: 15:40)

The slide is titled "Text format". It contains a bullet point "Importing data" followed by a code snippet:

```
data_txt1=pd.read_table('Iris_data_sample.txt')
```

Below the code is a screenshot of a Jupyter Notebook cell showing the output of the command. The output displays a DataFrame with 3 rows and 5 columns. The columns are labeled "SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm", and "Species". The data is as follows:

Name	Type	Index	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species		
data_txt1 DataFrame		0	1	1	5.1	3.5	1.4	0.2	"Iris-setosa"
		1	2	2	4.9	3	1.4	0.2	"Iris-setosa"
		2	3	3	4.7	3.2	1.3	0.2	"Iris-setosa"

After the DataFrame, there are two bullet points:

- All columns read and stored in a single column of dataframe
- In order to avoid this, provide a delimiter to the parameters 'sep' or 'delimiter'

At the bottom left of the slide, it says "Python for Data Science". At the bottom right, it says "14".

So, in this case it has been showed up with only 1 column. Let us see what is the problem, because all the column have been read as a single column. That is why the dimension is being 150 cross 1. So, now, let us see how to encounter this problem. If you see all the columns are read and stored in a single column of a data frame. So, in order to avoid this problem provide a delimiter to the parameters sep or delimiter let us see how to do that.

(Refer Slide Time: 16:06)

The slide is titled "Text format". It contains a bullet point "Default delimiter is tab represented by '\t'" followed by two code snippets:

```
data_txt1=pd.read_table('Iris_data_sample.txt',sep='\t')  
data_txt1=pd.read_table('Iris_data_sample.txt',delimiter='\t')
```

Below the code is a screenshot of a Jupyter Notebook cell showing the output of the command. The output displays a DataFrame with 3 rows and 5 columns. The columns are labeled "SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm", and "Species". The data is as follows:

Index	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species		
0	1	1	5.1	3.5	1.4	0.2	"Iris-setosa"
1	2	2	4.9	3	1.4	0.2	"Iris-setosa"
2	3	3	4.7	3.2	1.3	0.2	"Iris-setosa"

After the DataFrame, there is one bullet point:

- Tab delimiter might not always work

At the bottom left of the slide, it says "Python for Data Science". At the bottom right, there is a small image of a person.

So, the default delimiter is tab represented by slash t and let us see how to use those parameters, sep is 1 of the parameters that is used to provide a delimiter. Here I have used the delimiter called tap even if you do not use tab here by default it will always take the tab delimiter data, but I have specified the tab delimiter here that can be given using two separate arguments; one is sep separator and another one is delimiter. So, you can use any one of it and the tab delimiter might not always work.

Let us see what is the problem, the data is not a tab delimiter data that is why it is not working. The other commonly used diameters or commas and blanks other than tabs. So, in this case, using a comma as a delimiter also gives the earlier output right. In this case I have tried with a comma and that is also giving the same output where all the columns have been represented as a single column.

(Refer Slide Time: 16:51)

The slide has a blue header bar with the GITAA logo. The main title is "Text format". Below the title is a bulleted list:

- Other commonly used delimiters are commas and blanks
- In this case using a comma as a delimiter also gives the earlier output

Below the list is a screenshot of a Jupyter Notebook cell showing a DataFrame named "data_txt1". The DataFrame has columns "SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm", and "Species". The data consists of three rows:

Name	Type	Index	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	
data_txt1	DataFrame	0	1	5.1	3.5	1.4	0.2	"Iris-setosa"
		1	2	4.9	3	1.4	0.2	"Iris-setosa"
		2	3	4.7	3.2	1.3	0.2	"Iris-setosa"

At the bottom left of the slide, there is a watermark that says "Python for Data Science".

(Refer Slide Time: 17:10)

The screenshot shows a presentation slide with a blue header containing the GITAA logo. The main title is "Text format". Below the title, there is a bulleted list of three items:

- Other commonly used delimiters are commas and blanks
- In this case using a comma as a delimiter also gives the earlier output
- Now if we use a blank as a delimiter then

Below the list, a code snippet is shown in a code editor:

```
data_txt1=pd.read_table('Iris_data_sample.txt',delimiter=" ")
```

After the code, the output is displayed in a table:

Name	Type	Size
data_txt1	DataFrame	(150, 6)

A video player interface is visible at the bottom of the slide, showing a woman speaking and the text "Python for Data Science".

So, now if we use a blank as a delimiter then let us see what happens. So, I have used blank space inside the double quote that represents the delimiter as blank. So, if I try to read this, I will be able to see the data has been read with 150 rows and 6 columns. So, now, not the tab delimiter worked here and also the command did not work here and I have since I have used the blank delimiter the data has been read with 150 rows and 6 columns.

So, you have to be really sure about your data and how many rows and how many columns it is expected to have and you will be able to cross verify that whether you got the correct data or not whenever you read it.

(Refer Slide Time: 17:56)

The slide has a blue header with the GITAA logo. The title 'Text format' is at the top left. Below it is a bulleted list:

- Other commonly used delimiters are commas and blanks
- In this case using a comma as a delimiter also gives the earlier output
- Now if we use a blank as a delimiter then

```
data_txt1=pd.read_table('Iris_data_sample.txt',delimiter=" ")
```

Index	Unnamed: 0	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	5.1	3.5	1.4	0.2	Iris-set...
2	2	4.9	3	1.4	0.2	Iris-set...

A woman is speaking in front of the slide.

(Refer Slide Time: 18:06)

The slide has a blue header with the GITAA logo. The title 'Text format' is at the top left. Below it is a bulleted list:

- Remove index column and replace '??' and '# #' as missing values
- Instead of using `read_table()`, `read_csv()` can also be used to read `.txt` files

```
data_txt2=pd.read_csv('Iris_data_sample.txt',delimiter=" ")
```

A woman is speaking in front of the slide.

So, now if you see here all the records have been separated by cells by using the blank delimiter. So, in this case also you have to remove the index column and replace all the question marks, hash as missing values. I have not shown here but you can try that using the same format which you have used for csv and excel formats. Instead of using the `read_table` function, you can also use `read_csv` function which we used to read the csv file.

Now I am trying to show you how to read a text file using the `.read_csv` command.

So, you can use the same function that we used to do it whenever we want to import any csv file into Spyder but you have to just give the delimiter whenever you are reading any text file. That delimiter should corresponds to the type of data that you have in your sheet. So, here our data is delimited by a blank and I have given the delimiter is equal to blank. So, now we have come to the end of the lecture we have where we have seen about importing different formats of data into Spyder.

(Refer Slide Time: 19:08)

Summary

- File formats
- Commonly used file formats
- Read data from
 - .csv format
 - .xlsx format
 - .txt format

Python for Data Science

So, let us summarize whatever we have seen in this lecture. So, we have seen different formats of file, what are the formats of file that we will be looking into in this course and what are the commonly used file formats. So, that we can read those formatted files into Spyder. We have seen three formats of data; one being .csv format, we have seen how a comma separated values data will look like and we have also seen how to read them into Spyder.

Followed by that we have seen how to read .xlsx format. We have also seen how an excel sheet and how the data inside the excel sheet will look like. We have also seen how to read the text formatted data using both `read_table` command and `read_csv` command. So, on the whole we have seen how to read different formats of data into Spyder.

Thank you.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 14
Pandas Dataframes
Part-I

(Refer Slide Time: 00:19)

In this lecture



- Introduction to pandas
- Importing data into Spyder
- Creating copy of original data
- Attributes of data
- Indexing and selecting data



Python for Data Science

Hello all, welcome to the lecture on Pandas Dataframes. In this lecture, we are going to see about the pandas library. Specifically we are going to look at the following topics. First we will get introduced to the pandas library; after that we are going to see how to import the data into Spyder. We will be looking at how to create a copy of original data. We will also be looking at how to get the attributes of data; followed by that we will see or how to do indexing and selecting data.

(Refer Slide Time: 00:44)



Introduction to Pandas

- Provides high-performance, easy-to-use data structures and analysis tools for the Python programming language
- Open-source Python library providing high-performance data manipulation and analysis tool using its powerful data structures
- Name pandas is derived from the word Panel Data – an econometrics term for multidimensional data

Python for Data Science



To introduce you to the pandas library, it provides high performance, easy to use data structures, and analysis tools for the python programming language. It is an open source python library which provides high performance data manipulation and analysis tool using its powerful data structures. And, it is also considered one of the powerful data structures when compared with other data structures because of its performance and data manipulation techniques it is available in pandas. And, the name pandas is derived from the word panel data which is an econometrics term for multi dimensional data.

(Refer Slide Time: 01:20)



Pandas

- Pandas deals with dataframes

Name	Dimension	Description
Dataframe	2	<ul style="list-style-type: none">• two-dimensional size-mutable• potentially heterogeneous tabular data structure with labeled axes (rows and columns)

Python for Data Science



And here is the description of about the dataframe. Dataframe consist of two dimensions; the first dimension is the row and the second dimension is the column that is what we mean by two-dimensional and size-mutable. And whenever we say dataframe, dataframe is a collection of data in a tabular fashion, and the data will be arranged in rows and columns. Where we say each row represent a sample or a record, and each column represent a variable. The variable in the sense the properties, that are associated with each sample. The second point is that potentially heterogeneous tabular data structure with labelled axes.

Heterogeneous tabular data structure in the sense whenever we read a data into spyder, it becomes a dataframe and each and every variable gets a data type associated with that whenever you read it. We do not need to explicitly specify the data type to each and every variables, and that is basically based on the data or the type of data that is contained in a each variable or a column. And we mean labelled axes each and every row and columns will be labelled, row labelling the index for each rows which is starting from 0 to n-1, and the labels for column in the sense the names for each variables those are called labelled axes.

So, whenever we say labelled axes, the row labels are nothing but the row index and the column labels are nothing, but the column names, and this is about the basic description of the dataframe.

(Refer Slide Time: 02:56)

The screenshot shows a Python code editor with three import statements:

- `import os` → 'os' library to change the working directory
- `import pandas as pd` → 'pandas' library to work with dataframes
- `import numpy as np` → 'numpy' library to perform numeric operations

Below the imports, there is a bullet point: • Changing the working directory. A line of code is shown: `os.chdir("D:\\Pandas")`. In the bottom right corner, there is a video player interface showing a woman speaking.

So, next we will see how to import the data into Spyder. In order to import data into Spyder, we need to import necessary libraries; one of it is called os. Whenever you import any library, we use the command import, and OS is the library that is used to change the working directory. Once you open your Spyder, the default working directory will be wherever you have installed your python, and we import os to basically change the working directory, so that you will be able to access the data from your directory.

Next we are going to import the pandas library using the command input pandas as pd. Pd is just an alias to pandas. So, whenever I am accessing or getting any functions from pandas library, I will be using it as pd. And we have imported pandas to work with dataframes. We are also importing the numpy library as np to perform any numerical operations. Now, we have imported the library called os. And chdir is the function which is used to set a path from which you can access the file from. And inside the function, I have just specified my path wherever the data that I am going to import into Spyder is like my data is in the D drive under the folder pandas.

So now, this is how we change or set the working directory. Once we set the working directory, we are set to import any data into Spyder.

(Refer Slide Time: 04:32)

```

Importing data into Spyder
• Importing data
cars_data = pd.read_csv('Toyota.csv')


| Index | Unnamed: 0 | Price | Age | KM    | FuelType | HP | MetColor | Automatic | CC   | Doors | Weight |
|-------|------------|-------|-----|-------|----------|----|----------|-----------|------|-------|--------|
| 0     | 0          | 13500 | 23  | 46986 | Diesel   | 90 | 1        | 0         | 2000 | three | 1165   |
| 1     | 1          | 13750 | 23  | 72937 | Diesel   | 90 | 1        | 0         | 2000 | 3     | 1165   |
| 2     | 2          | 13950 | 24  | 41711 | Diesel   | 90 | nan      | 0         | 2000 | 3     | 1165   |



- By passing index_col=0, first column becomes the index column


cars_data = pd.read_csv('Toyota.csv',index_col=0 )


| Index | Price | Age | KM    | FuelType | HP | MetColor | Automatic | CC   | Doors | Weight |
|-------|-------|-----|-------|----------|----|----------|-----------|------|-------|--------|
| 0     | 13500 | 23  | 46986 | Diesel   | 90 | 1        | 0         | 2000 | three | 1165   |
| 1     | 13750 | 23  | 72937 | Diesel   | 90 | 1        | 0         | 2000 | 3     | 1165   |
| 2     | 13950 | 24  | 41711 | Diesel   | 90 | nan      | 0         | 2000 | 3     | 1165   |


```

So, now we will see how to import the data into Spyder. So, to import the data into Spyder, we use the command read_csv. Since we are going to import a csv file and the read_csv is from the library pandas, so I have used pd.read_csv. And inside the function,

you just need to give the file name within single or double quote and along with the extension.csv. And I am saving it to an object called cars_data. So, once I read it and save it to an object, my cars_data becomes the dataframe.

And once you read it, you will get the details in the environment tab, where you will see the object name, the type of the object and number of elements under that object. And once you double click on that object or the dataframe, you will get a window where you will be able to see all the data that is available from your Toyota file. This is just a snippet of three rows with all the columns. And I have multiple variables here first being the index. Whenever you read any dataframe into Spyder, the first column will be index; it is just the row labels for all the rows.

The next is the unnamed colon zero column. According to our data, we already have a column which serves the purpose for row labels. So, this is just an unwanted column. And next being the price variable which describes the price of the cars, because this data is about the details of the cars, and the properties that are associated with each cars. So, the each rows represents the car details, the car details being price, age, kilometre, fuel type, horsepower and so on.

First let us look at what each variable means price being price of the car all the details are about the pre owned cars. Next being the age of the car, and the age is being represented in terms of months and the kilometre, how many kilometre that the car has travelled, the fuel type that the car possess, one of the type is diesel, next being the horsepower. And we have another variable called MetColor that basically represents whether the car has a metallic colour or not; 0 means the car does not have a metallic colour and 1 means the car does have a metallic colour.

And next being automatic, what is the type of gearbox that the car possess; if it is automatic, it will be represented as 1; and if it is manual, it will be represented as 0. Next is being the CC of the car, and the doors represents how many number of doors that the car has, and the last being the weight of the car in kgs. So, this is just a description of all the variables in the Toyota csv. And we have also found out that there are two columns which serves the purpose for row labels, instead of having two columns we can remove either one of it, index is the default one. So, we can remove unnamed colon zero column.

So, how to get rid of this? Whenever you read any csv file by passing index_column = 0, the first column becomes the index column.

So now, let us see how to do that. So, whenever we read the data using the read_csv, we can just add another argument called index_column = 0. And the value 0 represents which column you should treat it as a index. I need the first column should be treated as the index. So, basically I have renamed, unnamed colon 0 to index. So, if you use 1 here, then price will be treated as row index. You will get the column name as index, but all the values will be the price column values.

So, but I do not want that since I already have a column which is in the name of unnamed, I am using that column as my index column. So, whenever I use index_column = 0. The first column will be treated as index column.

(Refer Slide Time: 08:47)

The slide has a blue header bar with the text 'Creating copy of original data'. In the top right corner, there is a logo for 'GITAA' with the tagline 'Enabling Success' and a small icon of a person. The main content area contains a table comparing 'Shallow copy' and 'Deep copy'. The table has two rows: one for 'Function' and one for 'Description'. The 'Function' row shows code examples: 'samp=cars_data.copy(deep=False)' for Shallow copy and 'cars_data1=cars_data.copy(deep=True)' for Deep copy. The 'Description' row lists characteristics for each type. The bottom left of the slide says 'Python for Data Science' and the bottom right says '8'.

	Shallow copy	Deep copy
Function	<code>samp=cars_data.copy(deep=False)</code> <code>samp = cars_data</code>	<code>cars_data1=cars_data.copy(deep=True)</code>
Description	<ul style="list-style-type: none">◦ It only creates a new variable that shares the reference of the original object◦ Any changes made to a copy of object will be reflected in the original object as well	<ul style="list-style-type: none">◦ In case of deep copy, a copy of object is copied in other object with no reference to the original◦ Any changes made to a copy of object will not be reflected in the original object

So, now we know how to import the data into Spyder. Let us see how to create the copy of original data, because there might be cases where we need to work with the copy of the data without doing any modifications to the original data. So, let us see in detail about how we can create a copy of original data. So, in python there are two ways to create copies, one is shallow copy and another one is deep copy.

First let us look at the shallow copy. The function row represents how to use the function, and the description represents what does that function means. So, in shallow

copy, you can use the `.copy` function that can be accessed whenever you have a dataframe. Since, I have `cars_data` as a dataframe, I can use `.copy`. If you want to do a shallow copy, you can use `deep = false` by default the value will be true.

So, there are two ways to do a shallow copy, one is by using the `.copy` function, another one is by just assigning the same data frame into a new object. I have assigned `cars_data` as `samp` using the assignment operator. So, this also means you are doing a shallow copy. So, what shallow copy means in the sense basically if you are doing a shallow copy, it only creates a new variable that shares the reference of the original object; it does not create a new object at all. Also any changes made to a copy of object, it will be reflected in the original object as well.

So, whenever you want to work with the mutable object, then you can do a shallow copy, where all the changes that you are making into `samp` will be reflected in your `cars_data`. Now, let us see about the deep copy. To do a deep copy, we use a same command `.copy`, but we said the `deep` as true. And by default the `deep` value will be true. So, whenever you use `.copy`, you are doing a deep copy. As you see I am doing a deep copy and by creating a new object called `cars_data1`, where `cars_data1` is the copy of the original data `cars_data`.

And what deep copy represents means in case of a deep copy, a copy of object is copied in another object like the copy of `cars_data` is being copied in another object called `cars_data` with no reference to the original. And whatever changes you are making it to the copy of object that will not be reflected in the original object at all. Whatever modifications you are doing it in `cars_data1` that will be reflected in that dataframe alone, the original dataframe will not get affected by your modifications.

So, there are two cases, you can choose any of the copies according to the requirements. Whenever you want to do any modifications and reflect back to the original data, in that case we can go for shallow copy. But if you want to keep the original data untouched and whatever changes you are making that should be reflected in the copy alone, then in that case you can use a deep copy.

(Refer Slide Time: 11:58)

The screenshot shows a Jupyter Notebook interface. At the top, there's a header with the GITAA logo and the text "Transforming careers". Below the header, the title "Attributes of data" is displayed. A section titled "DataFrame.index" is shown with a bullet point: "➤ To get the index (row labels) of the dataframe". Below this, a code cell contains the command "cars_data1.index". The output of this command, labeled "Out[8]", is a Int64Index object with values ranging from 0 to 1435, indicating the row labels for the dataframe. A cursor arrow points to the value "1435" in the output. In the bottom right corner of the slide, there's a small video thumbnail of a woman speaking.

So, now we will see how to get attributes of data, attributes in the sense getting the basic informations out of data, one of it is called getting the index from the dataframe. So, the syntax being `dataframe.index` can be used whenever you have a dataframe. So, to get the index, index means row labels here. Whenever you want to get the row labels of the data frame, you can use `data frame.index`.

Here `dataframe` being `cars_data1`, and I am using `.index` function that will give me the output for the row labels of the dataframe. If you see the row labels is ranging from 0 to 1435 where the length is 1436. So, the indexing in python starts from 0 to $n-1$ here. So, this is how we get row labels from the dataframe.

(Refer Slide Time: 12:50)

```
DataFrame.columns
> To get the column labels of the dataframe
cars_data1.columns

Out[10]:
Index(['Price', 'Age', 'KM', 'FuelType',
       'HP', 'MetColor', 'Automatic', 'CC',
       'Doors', 'Weight'],
      dtype='object')
```

Python for Data Science

Next we will see about how to get the column names of the dataframe. You can get the column labels of the dataframes using.columns. So, cars_data1.columns will give you all the column names of your dataframe. Basically the output is just an object which is a list of all the column names from the dataframe cars_data1. By getting the attributes of the data like the row labels and the column labels, you will be able to know from which range your row labels are starting from and what are all the column names, that you have in your dataframe.

(Refer Slide Time: 13:29)

```
DataFrame.size
> To get the total number of elements from the
  dataframe
cars_data1.size

Out[11]: 14360

DataFrame.shape
> To get the dimensionality of the dataframe
cars_data1.shape

Out[12]: (1436, 10) => 1436 rows & 10 columns
```

Python for Data Science

Next we can also get the size that is we can also get the total number of elements from the dataframe using the command.size. Here this is just the multiplication of 1436 into 10, where 1436 rows are there and 10 columns are there. So, when you multiply that you will get the total number of elements that is what the output represents, you can also get the shape or the dimensionality of the dataframe using the command.shape.

So, cars_data1.shape will give you how many rows are there and how many columns are there explicitly. The first value represents rows 1436 rows are there, and 10 columns are there. So, you will be able to get the total number of elements as well as how many number of rows, and how many number of columns are there separately also.

(Refer Slide Time: 14:20)

The screenshot shows a Jupyter Notebook interface. The title bar says 'GITAA' with a logo. The main area has a header 'Attributes of data'. Below it, there are two code snippets: 'DataFrame.memory_usage([index, deep])' and 'cars_data1.memory_usage()'. A callout box points to the second snippet with the text 'The memory usage of each column in bytes'. To the right, the output of the 'cars_data1.memory_usage()' command is displayed in a table:

	Out[14]:
Index	11488
Price	11488
Age	11488
KM	11488
FuelType	11488
HP	11488
MetColor	11488
Automatic	11488
CC	11488
Doors	11488
Weight	11488
	dtype: int64

So, next we will see about the memory usage of each column in bytes. So, to get the memory usage of each column in bytes, so we use the command.memory_usage, and the.memory_usage will give you the memory used by each column that is in terms of bytes. So, if you see all the variables has used the same memory, there is no precedents or there is no higher memory that is used for any particular variable. All the variable has used the same memory and the data type that you are seeing here is the data type of the output.

(Refer Slide Time: 14:57)

Attributes of data

`DataFrame.memory_usage([index, deep])`

➤ The memory usage of each column in bytes

`cars_data1.memory_usage()`

`DataFrame.ndim`

➤ The number of axes / array dimensions

`cars_data1.ndim`

`Out[15]: 2`

A two-dimensional array stores data in a format consisting of rows and columns

Python for Data Science

Next, how to get the number of axes or the array dimensions. Basically to check how many number of axes are there in your dataframe, you can get that using.ndim function. So, I have used cars_data1.ndim that will give you how many number of axes are there that is basically how many number of dimensions that are available for your dataframe. It basically the output says as to because the cars_data1 has two dimension, one dimension being rows and the other dimension being columns.

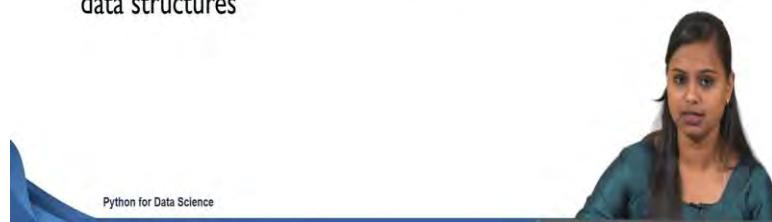
All the rows forms one dimension, and all the columns forms the other dimension. And just because we have multiple variables, it does not mean that we have multi dimension to your data. The data frame just consist of two dimensions. So, it becomes a two-dimensional data frame. And if you see a two-dimensional array stores a data in a format consisting of rows and columns, that is why our dataframes dimension is 2.

(Refer Slide Time: 16:04)

Indexing and selecting data



- Python slicing operator ‘[]’ and attribute/dot operator ‘.’ are used for indexing
- Provides quick and easy access to pandas data structures



So, next we will see how to do indexing and selecting data. The python slicing operator which is also known as the square braces and attribute called.operator which is being represented as a period, that are used for indexing. And, indexing basically provides quick and easy access to pandas data structures. Whenever you want to index or select any particular data from your dataframe, the quick and easy way to do that will be using the slicing operator and a dot operator.

(Refer Slide Time: 16:40)

Indexing and selecting data



`DataFrame.head([n])`

➤ The function `head` returns the first n rows from the dataframe

`cars_data1.head(6)`

By default, the `head()` returns first 5 rows

Out[17]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	13500	23.0	46986	Diesel	90	1.0	0	2000	three	1165
1	13750	23.0	72937	Diesel	90	1.0	0	2000	3	1165
2	13950	24.0	41711	Diesel	90	NaN	0	2000	3	1165
3	14950	26.0	48000	Diesel	90	0.0	0	2000	3	1165
4	13750	30.0	38500	Diesel	90	0.0	0	2000	3	1170
5	12950	32.0	61000	Diesel	90	0.0	0	2000	3	1170

Python for Data Science

14

So, now we will see how to index and select the data. First thing that we are going to see is about the head function. Basically the head function returns a first n rows from the dataframe. The syntax being `dataframe.head` inside the function you can just give how many number of rows it should return. And I have used 6 here that means, that the head function will return me 7 rows. If you note by default the head function returns only the first 5 rows from your dataframe.

You can also specify your desired value inside the head function. If you do not give any value to it by default, it will give the first 5 rows. So, if you see, it returns 6 rows with all the column values, so this will be useful whenever you want to get the schema of your dataframe. Just to check what are all the variables that are available in your dataframe, and what each variable value consists of. In that case, the quick method or the quick way to do is using the head function.

(Refer Slide Time: 17:43)

Indexing and selecting data

➤ The function `tail` returns the last n rows for the object based on position

```
cars_data1.tail(5)
```

Out[27]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
1431	7500	NaN	20544	Petrol	86	1.0	0	1300	3	1025
1432	10845	72.0	??	Petrol	86	0.0	0	1300	3	1015
1433	8500	NaN	17016	Petrol	86	0.0	0	1300	3	1015
1434	7250	70.0	??	NaN	86	1.0	0	1300	3	1015
1435	6950	76.0	1	Petrol	110	0.0	0	1600	5	1114

✓ It is useful for quickly verifying data
✓ Ex: after sorting or appending rows.

There is also an option where you can get the last few rows from your data frame using the tail function. Basically the function tail returns a last n rows for the dataframe that you have specified. I have used `cars_data1.tail`, and inside the function I have used 5. Even if you do not give 5, it will return the last 5 rows from your dataframe. This will be a quickest way to verify your data whenever you are doing sorting or appending rows.

(Refer Slide Time: 18:16)

Indexing and selecting data

- To access a scalar value, the fastest way is to use the `at` and `iat` methods
 - `at` provides label-based scalar lookups
 - In [29]: `cars_data1.at[4, 'FuelType']`
Out[29]: 'Diesel'
 - `iat` provides integer-based lookups
 - In [30]: `cars_data1.iat[5,6]`
Out[30]: 0

So, now we have seen how to access the first few rows and the last few rows from the dataframe. There is also a method to access a scalar value. The fastest way is to use the `at` and `iat` methods. Basically the `at` provides label based scalar look ups. Whenever you want to access a scalar value, you can either use `at` function or use `iat` methods. So, if you are using `at` function, you basically need to give the labels inside the function that is what it means as label-based scalar look ups, I am accessing a function called `at`.

And inside the function the first value should be your row label and the second value should be your column label. And I am accessing the scalar value which corresponds to 5th row and corresponds to the fuel type column. So, whatever I have given here is just the labels for rows and columns. And the value corresponds to 5th row and the fuel type is diesel. So, this is how you access a scalar value using the `at` function using just the row labels and the column labels.

There is also another method called `iat`. And the `iat` provides integer based look ups where you have to use the row index and the column index, instead of using labels you can also give row index and the column index. If you are sure about the index, then you can go for `iat`, but if you are sure about just the column names then in that case you will go for `at` function. So, if you see here I have used dot `iat` function where 5 is the 6th row, and 6 is the 7th column. And the value corresponding to 6th row and 7th column would be 0. So, this is how I access the scalar value using the row index and the column index.

(Refer Slide Time: 20:11)

Indexing and selecting data



- To access a group of rows and columns by label(s) `.loc[]` can be used

```
In [31]: cars_data1.loc[:, 'FuelType']
Out[31]:
0      Diesel
1      Diesel
2      Diesel
3      Diesel
4      Diesel
5      Diesel
6      Diesel
7        NaN
8     Petrol
```

Python for Data Science



So, now we have seen how to access a scalar value from your dataframe, there is also an option where you can access a group of rows and columns by labels that can be done using the `.loc` operator. So, now, we will see how to use the `.loc` operator to fetch few rows or columns from your dataframe. So, here also you have to use the dataframe name that is `cars_data` one. So, whenever you are using the `.loc` operator, you should be using a slicing operator followed by the function. So, inside the function, you just basically need to give two values. One should represent the row labels, and the other should represent the column labels.

So, here I want to fetch all the row values from a column called `FuelType` in that case I can use colon to represent all, but here is just the snippet of 9 rows just for explanation purpose, but in your Spyder you will be getting all the rows which are under the `FuelType` column. You can also give multiple columns. For example, you can give `FuelType` and `price` in as a list basically when we say list you can just give the values inside the square brackets, in that case you will get all the row values based on multiple columns. So, this is how we access group of rows and columns using `.loc` operator.

(Refer Slide Time: 21:35)

Summary



- Introduction to pandas
- Importing data into Spyder
- Creating copy of original data
- Attributes of data
- Indexing and selecting data

Python for Data Science



So, in this lecture, we have seen about the pandas library. We have also seen how to import data into Spyder. After importing we have also seen how to get the copy of your original dataframe; followed by that we have seen how to get the attributes of data like row labels and column labels from your data. And after that we have seen how to do indexing in selecting data using iat, at and.loc operators.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 15
Pandas Dataframes
Part - II

Hello all welcome to the lecture on Pandas Dataframes.

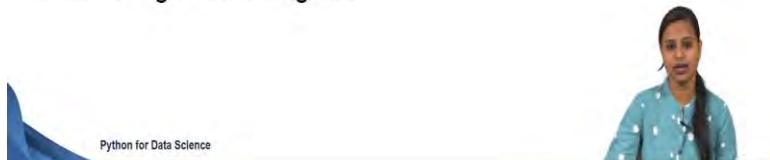
(Refer Slide Time: 00:19)

Pandas Dataframes - Recap



In the previous lecture, we have seen about

- Introduction to pandas
- Importing data into Spyder
- Creating copy of original data
- Attributes of data
- Indexing and selecting data



So, let us have a quick recap on what we have done in the previous lecture, on Pandas Dataframes. So, in the previous lecture we have seen about introduction to Pandas, where we have been introduced to the Pandas library, then we have seen about how to import the data into Spyder.

So, once we imported we have also seen, how to create a copy of original data, we have also seen how to get the attributes of data, followed by that we have also seen indexing and selecting data.

(Refer Slide Time: 00:47)

In this lecture

- Data types
 - Numeric
 - Character
- Checking data types of each column
- Count of unique data types
- Selecting data based on data types
- Concise summary of dataframe
- Checking format of each column
- Getting unique elements of each columns

Python for Data Science

So, in this lecture we are going to see about the data types of variables in a dataframe. We are going to look about numeric data types and character data types, which we are going to use often in our analysis. Once, we know about the two data types we are going to see, how to check the data types of each column in your dataframe.

Followed by that we are going to look at, how to get the count of unique data type? After that we will also see how to select the data based on particular data types. And, then we are going to look at the concise summary of the dataframe, followed by that we are going to check the format of each column just to cross verify whether the data type is of desired data type or not.

After, that we are going to get the unique elements of each column, so, in this lecture we will be looking at different topics as we have mentioned in detail. So, first we will look about data types.

(Refer Slide Time: 01:45)

Data types

- The way information gets stored in a dataframe or a python object affects the analysis and outputs of calculations
- There are two main types of data
 - numeric and character types
- Numeric data types includes integers and floats
 - For example: integer – 10, float – 10.53
- Strings are known as objects in pandas which can store values that contain numbers and / or characters
 - For example: ‘category1’

Python for Data Science

So, the way information gets stored in a dataframe or in any python object basically affects, whatever analysis that you are going to perform on your dataframe. As well as that results in a different form of outputs from the calculations that you have made. For example, you cannot perform any numerical operations on a string; similarly you cannot do any string related operations on a numerical data. So, we have to be sure of what data type you are handling in a dataframe.

So, basically there are two main types of data as discussed one is being numeric and another one is being character types. And, numeric data types basically includes all the numerical values, which are in terms of integers and floats for example, integer value being represented as 10 and float values are called whenever it has a numerical value with the decimal. So, here if we have decimal values after 10, then it will be represented as float for example, as 10.53.

This is about the numeric data types that the python can handle. Next, we are going to look about the character types. So, character data types are nothing, but all the strings are known as objects in pandas, which can store values that contains numbers as well as characters.

So, whatever value that is been enclosed inside a single or double quotes will be considered as a string and that is being represented as object in pandas. And, for example, if you have a string which is enclosed inside the single or double quotes like,

category1 that becomes a string value or an object. Even, though it has both strings and numbers whatever has been enclosed within single or double quotes will be considered as string in python. We will look in deep about, what string values represents and what numerical values represents?

(Refer Slide Time: 03:45)

The slide is titled "Numeric types". It contains a table with three columns: "Python data type", "Pandas data type", and "Description".

Python data type	Pandas data type	Description
int	int64	Numeric characters
float	float64	Numeric characters with decimals

Below the table, there is a bulleted list:

- Pandas and base Python uses different names for data types
- '64' simply refers to the memory allocated to store data in each cell which effectively relates to how many digits it can store in each "cell"
- 64 bits is equivalent to 8 bytes
- Allocating space ahead of time allows computers to optimize storage and processing efficiency

At the bottom left, it says "Python for Data Science" and at the bottom right, it says "5".

So, first we will look about the numeric types whenever we deal with the Pandas library using the python tool. It is not necessary that both the pandas and python uses the same names for data types, because pandas and base python uses different names for different data types. For example, here is a table to illustrate you, how the data type is named in python and how the data type is being named in Pandas library and followed by that we have the description as well.

First we will look at integer that is being represented as int in terms of base python and in if you look at the Pandas library; the integers will be represented in terms of int 64. And, int64 corresponds to all numeric characters; it can contain all the numeric characters. Next is a float; float is being represented as float in python whereas; in pandas it is being represented as float64.

So, float64 basically corresponds to all the numeric characters with decimal values. And, these 64 simply refers to the memory allocated to store the data in each cell, which effectively relates to how many digits it can store in each cell?

So, at the max it can store up to 64 bits which is equivalent to 8 bytes. So, why we are really concerned about the memory allocation in each cell, because allocating space ahead of time allows computers, to optimize storage and processing efficiency. Because, whenever you read any data into Spyder or into any IDE of python, it basically gets read with the data type for each and every variable according to the values that it has.

So, in that case it always allocates memory to store the data in each cell just to optimize the storage and processing efficiency. So, now we have seen about the numeric types, where we have seen about integer and float data types.

(Refer Slide Time: 05:53)

The slide has a blue header bar with the GITAA logo. The main title is 'Character types'. Below it is a bullet point: '• Difference between category & object'. A table compares the two:

category	object
- A string variable consisting of only a few different values. Converting such a string variable to a categorical variable will save some memory	- The column will be assigned as object data type when it has mixed types (numbers and strings). If a column contains 'nan' (blank cells), pandas will default to object datatype.
- A categorical variable takes on a limited, fixed number of possible values	- For strings, the length is not fixed

At the bottom left is the text 'Python for Data Science'. On the right side of the slide, there is a video player showing a woman speaking.

Next, we are going to see about the character types. In python there are two data types that can handle character types; one is category and another one is being object. So, now, there are two different data types that are available for character types. So, let us look at the difference between category and object.

So, when you look at the first point; the first point describes what the category data type would be basically any string value consisting of only a few different values, then that becomes a category. And, we have to convert such string variable to a categorical variable, which can save us some memory. Instead of keeping too many string values in a form of same representation we can always convert them to category data type to have it as a categories.

A categorical variable takes on a limited fixed number of possible values, because there is limits to the length, that is being fixed and it can always accommodate only fixed number of possible values; you cannot have 15 to 20 different values, which forms categories. So, in that case we always if you want to have too many categories in your column, then you can go for object, because, the column will be assigned as object data type where it has mixedtypes.

Basically, whenever you have numbers which is being represented as 0 1, that can also be an object, if it is being enclosed within single or double quote or even if your column has mixed numbers that is category 1, category 2, category 3, that can also be an object. And, the other point is if a column contains nan values, basically in python all the blank cells will be filled with the default nan values.

And, the next point would be, if a column contains nan that is just a blank cell. Then, the pandas will default to object data type, because by default whenever you read any blank cell in Spyder, all the blank cell will be read as nan values and by default it will default to object data type. Because all the nans will be considered as a different value so, it becomes an object data types itself.

And, here has an advantage is that for strings the length is not fixed, how many ever number of elements you can have as a string and there is no limit to the number of possible values that you can have as an object. So, this is the difference between the category and the object character types. So, now, we have seen the difference between the category and object data types.

(Refer Slide Time: 08:35)

Checking data types of each column

dtypes returns a series with the data type of each column

Syntax: `DataFrame.dtypes`

```
cars_data1.dtypes
```

	Out[37]:
Price	int64
Age	float64
KM	object
FuelType	object
HP	object
MetColor	float64
Automatic	int64
CC	int64
Doors	object
Weight	int64
	dtype: object

So, now we know about the basic data type that we are going to often work with. So, now, it is time to check the data types of each columns of the data frame that we are working with. So, basically if you want to check the data type of each column, because whenever you have been given a data, you want to really check what is the structure of the data; that means, which variable has which data type? In, that case you can use dtypes, because that returns a series with the data type of each column and the syntax would be you use dtypes along with the Data Frame name. So, Data Frame.dtypes will give you a series with the data type of each column.

So, the input would be cars_data1.dtypes, where cars the_data1 is the Data Frame, that we are looking at and the output is shown below where you have multiple variables correspondingly you have the data type of each variables. When we look at the first variable that is price, the price data type is of integer 64, which is the desired one and the age being float 64 and the kilometer has been read as object. Similarly, you will be able to check the data type for each variables using the dtypes commander.

(Refer Slide Time: 09:59)

```
Count of unique data types
```

`get_dtype_counts()` returns counts of unique data types in the dataframe

Syntax: `DataFrame.get_dtype_counts()`

```
cars_data1.get_dtype_counts()
```

Out[38]:

float64	2
int64	4
object	4
dtype:	int64

Python for Data Science

So, now we have an overall idea about what are the data types that we are going to work with using the cars_data. There is also an option where you can get the count of unique data types available in your Data Frame. So, in that case `get_dtype_counts`, returns the counts of unique data types in the data frame.

So, if you want to summarize how many in 64 variables are there and how many float 64 variables are there, then in that case you can use `get_dtype_counts` command. And, the syntax will be you will use the command along with the data frame name. So, let us see how to do that? So, here is an input where I have given `cars_data1.get_dtype_counts`. So, that will give me an output which is shown here. So, I have the summarization here, where I have different types of data type as well as I have the corresponding count also.

So, basically on the whole I have 2 variables of float64 data type and I have 4 variables of int 64 data type. As well as I have 4 variables, which consist of object data type? And `dtype int 64` represents the output data types since it has the values it is being represented as `int64`.

(Refer Slide Time: 11:25)

Selecting data based on data types

pandas.DataFrame.select_dtypes() returns a subset of the columns from dataframe based on the column dtypes

Syntax: DataFrame.select_dtypes(include=None, exclude=None)

`cars_data1.select_dtypes(exclude=[object])`

Out[39]:

	Price	Age	MetColor	Automatic	CC	Weight
0	13500	23.0	1.0	0	2000	1165
1	13750	23.0	1.0	0	2000	1165
2	13950	24.0	Nan	0	2000	1165
3	14950	26.0	0.0	0	2000	1165
4	13750	30.0	0.0	0	2000	1170

So, now we also have an overall idea about the count of unique data types that we are going to handle with. So, now, we know about how to get the data type of each variables. So, there might be cases where you want to perform the operations only on a numerical data type. Similarly, there can be cases where you are going to work with only categorical data type.

In that case, there is also a platform where you can select the data based on the data types available in your DataFrame. So, let us see how to do that? So, select_dtypes is the command that we are going to use, to select the data based on the data types. And, along with that you have to give the DataFrame name and since that is from the Pandas library this it is being represented as Pandas.DataFrame.select_dtypes. And, this command returns a subset of the columns from the DataFrame, based on the column types you have specified inside a function.

So, let us see how to use the function, here DataFrame would be the DataFrame name and here is a command that the data select_dtypes inside the function there are two arguments. One is include and another one is exclude both being nan by default, if you want to select only those columns, which are of object data type, then you can just use object inside the include argument and if you want to exclude any particular data type from your analysis. In that case you can use the data types under the exclude argument.

So, let us see an example how we are going to do that? So, here is the input the DataFrame that we are working with is cars_data one. And, the command being select_dtypes and inside the function, I have given exclude is equal to object. And, whatever data type that you are giving that should be given inside the square braces, because you can also give multiple data types object comma int 64. Basically, in that case you will be excluding all the columns which are of object and integer 64 data types.

Here, I just want to include the object data type. So, I have just given object. So, the output would be a DataFrame with the variables, which are not of object data type. So, here is the output we have price age metcolor automatic CC and weight with the data type integer and float. For example, the FuelType the doors all those have been excluded. So, this is how we basically select the data based on the data types.

(Refer Slide Time: 13:57)

Concise summary of dataframe

`info()` returns a concise summary of a DataFrame

- data type of index
- data type of columns
- count of non-null values
- memory usage

Syntax: `DataFrame.info()`

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price      1436 non-null int64
Age        1336 non-null float64
KM         1436 non-null object
FuelType   1336 non-null object
HP         1436 non-null object
MetColor   1286 non-null float64
Automatic  1436 non-null int64
CC         1436 non-null int64
Doors      1436 non-null object
Weight     1436 non-null int64
dtypes: float64(2), int64(4), object(4)
memory usage: 163.4 KB
```

So, next we are going to see about how to get the concise summary of DataFrame. So, there is a command called info that returns a concise summary of a DataFrame, the concise summary includes the data type of index; index being the row labels, the data type of row labels is what the output gives as well as it gives the data type of columns, it also gives the count of non-null values. Basically, how many filled values are there in your DataFrame. Also, it gives the memory usage of the DataFrame and the syntax would be you use the info command along with the DataFrame name.

So, let us see how to do that I have given cars_data1.info. So, the output will be similar to this, where the output starting line is Pandas.core.frame.DataFrame. So, it is of pandas core DataFrame and int 64 index being the index are being represented in terms of int 64, where you have 1400 and 36 entries, which are ranging from the 0 to 1435 row labels and totally you have 10 columns in your DataFrame. And, after that you have list of variables along with that, you have non-null values and what is the data type corresponding to that variable.

So, for example, under the price variable the total observations are 1436 and there are also 1436 non-null values and the data type of price being integer 64. In this case I have highlighted few rows. So, the purpose to check the concise summary of DataFrame is to verify, whether all the variables have been read with the proper data type or not.

If not we have to go back and convert them back to the desired data type. So, for example, price and age are of expected data type and if you see kilometer, there are no missing values actually 1400 and 56 observations are non-null and it is being read as object kilometer would be ideally be numbers. So, it should have not been read as object, but in this case it have been read as object.

Similarly, fuel type should be of object there is no problem in that and if you see HP, MetColor and Automatic, they have been read as object, float64, int64. Why, because metallic color automatic basically represents categories under that column, metallic color represents the whether the car has metallic color or not. So, it should be ideally it should be having some categories to it.

So, there is some problem that is why it has been read as float 64 and if you look at the automatic column; automatic also represents the type of gear box that the car possess. So, in that case it should ideally be categories and it should ideally be object. In this case it is int 64.

(Refer Slide Time: 16:59)

Checking format of each column



By using `info()`, we can see

- 'KM' has been read as object instead of integer
- 'HP' has been read as object instead of integer
- 'MetColor' and 'Automatic' have been read as float64 and int64 respectively since it has values 0/1
- Ideally, 'Doors' should've been read as int64 since it has values 2, 3, 4, 5. But it has been read as object
- Missing values present in few variables

Let's encounter the reason !

Python for Data Science

So, now let us just check the format of each column. So, just to summarize by using info, we can see that kilometer has been read as object instead of integer right. Next, we have horsepower that has been read as object instead of integer as well. Next, being metallic color and automatic both have been read as float 64 and int 64 respectively. Since, it just has values zeros and ones that is the reason that it has been read as float 64 and int 64.

But, it has been read as object, because since it has numbers only ideally it should have been read as integer 64, but it has been read as object. And, we have also seen there are missing values present in few variables. So, we have to encounter the reason behind all of these points.

(Refer Slide Time: 17:53)



Unique elements of columns

`unique()` is used to find the unique elements of a column

Syntax: `numpy.unique(array)`

```
print(np.unique(cars_data1['KM']))
```

`['1' '10000' '100123' ... '99865' '99971' '??']`

- 'KM' has special character to it - '??'
- Hence, it has been read as object instead of int64

Python for Data Science



So, unique function is used to get the unique elements of a column the syntax being numpy.unique of array, unique function comes from the numpy library and the input should be an array, you cannot perform the unique operation on a multiple array it can be done on a single array only. So, here I have used np.unique that is why because I have imported numpy as np.

So, the alias is just np if you have not imported the numpy library at this point of time you can import numpy library as np and then you can follow with the code here. So, I am just print np.unique and the input should be an array. So, I am accessing the kilometer variable from the cars_data1.

So, that will give me the unique values of kilometer column. So, there are so, many unique values, but only few being shown here, you have a ?? symbol that is the representation, you are not seeing all of the values. And, the special thing about the kilometer is it has a special character that is double question mark, which has been enclosed with in single quote that is the reason; it has been read as object instead of 64.

So, whenever you have a special character all the values will be converted to character or string data type. So, that is why the kilometer has been read as object instead of int 64.

(Refer Slide Time: 19:23)

Unique elements of columns

Variable 'HP':

```
print(np.unique(cars_data1['HP']))
```

['107' '110' '116' '192' '69' '71' '72'
'73' '86' '90' '97' '98' '?????']

- 'HP' has special character to it - '?????'
- Hence, it has been read as object instead of int64

Variable 'MetColor':

```
print(np.unique(cars_data1['MetColor']))
```

[0. 1. nan nan nan nan nan nan
nan nan nan nan nan nan nan]

- 'MetColor' have been read as float64 since it has values 0. & 1.

Python for Data Science

Similarly, we are going to look at the variable horsepower. So, I am going to get the unique elements of the column horsepower using the same unique command. So, you have different values under the horsepower. And, you also have a special character like 4 question marks that is the reason it has been read as the object instead of int 64.

And, when we look at the metallic color, I am using the same function dot unique to get the unique elements under the metallic color column. So, basically it has only the value 0s and 1s 0 point and 1 point. So, that is why it has been read as float 64. Since, it has values 0 and 1. Even, though it has nan values just because the value has decimals to it the whole variable have been read as float 64.

(Refer Slide Time: 20:19)

The screenshot shows a Jupyter Notebook interface. At the top right is the GITAA logo. The main content area has a blue header bar with the title "Unique elements of columns". Below this, there are two sections of code execution:

- Variable 'Automatic':**

```
print(np.unique(cars_data1['Automatic']))  
[0 1]
```

 - 'Automatic' has been read as int64 since it has values 0 & 1
- Variable 'Doors':**

```
print(np.unique(cars_data1['Doors']))  
['2' '3' '4' '5' 'five' 'four' 'three']
```

 - 'Doors' has been read as object instead of int64 because of values 'five' 'four' 'three' which are strings

At the bottom left of the slide, there is a small watermark or logo for "Python for Data Science". On the right side of the slide, there is a video frame showing a woman speaking.

So, next we are going to look at another column that is automatic. I have used the same.unique function to get the unique elements of the column automatic. So, the output being 0 and 1, as we know 0 and 1 represents category part automatic has been read as integer 64. Since, it has value 0 and 1.

Next, we are going to look at the variable door and where we tried to get the unique values out of the doors column, if you see the output there are few values 2 3 4 and 5 also you have values as strings that is being represented as 5 4 3. So, there might be a typo where all the numerical 3 values have been typed as 3 in characters, similarly for 5 and 4.

So, this might be an error while getting the data from the source. So, this is the problem where door has been read as object instead of int 64, that is because of the values 5 4 3, which are in strings data type. So, now we have an overall idea about how do we check the data type of each variable and how to cross verify whether each data type is of expected data type or not. If, not we have seen how to get the unique elements out of columns to cross verify, whether there are some problems to it. So, that we can go back and reconvert them back to the expected data type.

(Refer Slide Time: 21:51)

Summary

- Data types
 - Numeric
 - Character
- Checked data types of each column
- Count of unique data types
- Selected data based on data types
- Concise summary of dataframe
- Checked format of each column
- Got unique elements of each columns

Python for Data Science

So, next I am going to summarize whatever we have done in this lecture. Basically, we started with looking into 2 data types that is numeric and character for variables in the data frame. We have also checked the data type of each column, whether each data type is of expected data type or not. And, then we have also seen how to get the count of unique data types to get an overall idea about, what are the data types that we will be working with in our dataframe.

And, next we have seen how to select the data based on data types. For example, if you want to perform any operations that are completely related to numbers, then you would be selecting the data only with respect to numeric data types, which is like integer and float 64 data types. We have seen how to select the data based on data types?

And, we have also looked at the concise summary of dataframe to basically look at the variable and what is the data type of each variable along with that, we have also seen how to get the count of non-null values are there, which basically describes how many filled values are there in your dataframe.

After looking at the concise summary of data frame we had an idea about what each data type represents and after that we have also checked the format of each column just to cross verify, whether it is of expected data type or not, but not all the variables are of expected data type. So, we have to convert them back to the expected data type in the next lecture, we have also seen how to get the unique elements of each column. So, that

we got an idea about what each variables values are that is causing the variable which is not of expected data type.

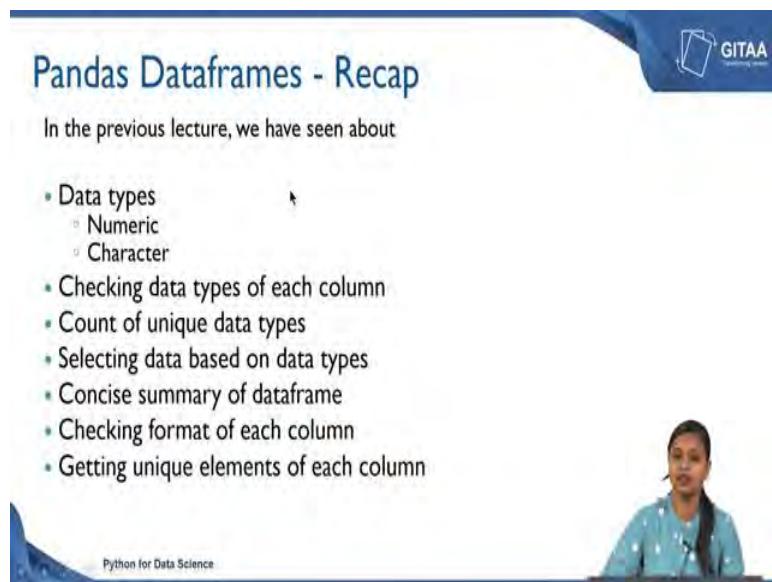
So, in the next lecture we will be looking at to resolve all the problems that we have encountered in this lecture.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 16
Pandas Dataframes Part-III

Hello all, welcome to the lecture on Pandas Dataframes.

(Refer Slide Time: 00:19)



Pandas Dataframes - Recap

In the previous lecture, we have seen about

- Data types
 - Numeric
 - Character
- Checking data types of each column
- Count of unique data types
- Selecting data based on data types
- Concise summary of dataframe
- Checking format of each column
- Getting unique elements of each column

Python for Data Science

So, we have been looking at the pandas Dataframes in the previous lectures. So, let us have a quick recap on what we have done in the previous lecture. So, in the previous lecture we have seen about the data types the two broad main data types that are numeric in character which we will be working with often and what they represents and where we can use numeric and character data types.

We have also checked the data types of each column to generally know about what are the data types we are going to work with for the analysis and we have also seen how to get the count of unique data types to get an overall idea about how many variables are there with different data types and we have also seen how to select the data based on data types.

For example, if you want to perform any operations only on a specific data type we can also do that by selecting the data based on the particular data types. And we have also looked at the concise summary of Dataframe where we have got some information about

the Dataframe with respect to what is the data type of each variables and how many non-null values are there under each column and which also gave us the memory used by the Dataframe and then we checked the format of each column.

And we found out that there are few columns that has not been read properly. So, we have gone back and looked at the unique elements of each column to get an idea about what is the reason behind variables being read with different data types which are not expected. So, these are the things we have seen in the previous lectures.

(Refer Slide Time: 01:57)

The screenshot shows a presentation slide with a blue header bar. On the left, the text 'In this lecture' is displayed in white. On the right, there is a logo for 'GITAA' with a small icon. Below the header, a list of topics is shown in a bulleted format:

- Importing data
- Concise summary of dataframe
- Converting variable's data types
- Category vs Object data type
- Cleaning column 'Doors'
- Getting count of missing values

At the bottom of the slide, there is a video player interface showing a woman speaking. The video player has a play button, a progress bar, and a volume control icon. The text 'Python for Data Science' is visible at the bottom left of the video area.

And in this lecture we are going to see how to import the data. In the previous lectures we found out that there were question marks and other special characters that are not being considered as blank while you. So, now, in this lecture we are going to import the data by considering all of those after that we are going to again look at the concise summary of data to cross verify whether there is any difference after importing the data by considering all those special characters or not.

After that we are going to convert the variable's data types because in the previous lectures we have encountered that some variables are of not expected data type. So, in this lesson we are going to convert variables to the expected data type and we are also going to look at the category which is object data type what is the effect being the category data type and what is the effect being a object data type.

Next we are going to clean the column door since it has few strings to it like 5, 4 and 3. So, we are going to clean that to make the Doors columns perfect for the future analysis after that we are going to get the count of missing values to basically arrive at a solution on how we are going to go ahead with the missing values. So, now, let us try to import the data by considering the all forms of missing values.

(Refer Slide Time: 03:20)

The slide has a blue header with the title 'Importing data'. In the top right corner, there is a logo for 'GITAA' with a small icon. The main content area contains the following text:

- We need to know how missing values are represented in the dataset in order to make reasonable decisions
- The missing values exist in the form of 'nan' '???' '????'
- Python, by default replace blank values with 'nan'
- Now, importing the data considering other forms of missing values in a dataframe

```
cars_data = pd.read_csv('Toyota.csv', index_col=0,
na_values=['??', '????'])
```

At the bottom left of the slide, it says 'Python for Data Science'.

So, now, basically before importing we need to know how missing values are being represented in the data set in order to make any reasonable decisions because you need to be really sure how the missing values are presented in your dataframe, whether it is a blank value or zeros or any question marks or with any special characters.

So, but we have encountered in our data frame the missing values exist in the form of nan which represents the blank values. Double question mark and four question mark represents the missing values wherever they do not have any records for, but we do not have to worry about the blank values because python by default replace blank values with nan. So, now, we know what are the other forms of missing values we have in our dataframe.

So, now, I am going to import the data considering the other forms of missing values in a dataframe. So, I am using the same command as earlier that is.read_csv from the pandas library and pd being the alias. So, the input being Toyota csv file and the index I have set

at to 0. So, that the first column will be treated as the index column and I have added another argument called `na_values`.

So, in the argument I have given double question mark and four question marks to basically say consider all these values as `nan` values. So, now we have seen how to import the data by considering all forms of missing values in your dataframe.

(Refer Slide Time: 04:52)

```
Summary - before replacing special characters with nan
cars_data.info()
<class 'pandas.core.frame.DataFrame'\>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price    1436 non-null int64
Age      1336 non-null float64
KM       1436 non-null object
FuelType 1336 non-null object
HP       1436 non-null object
MetColor 1286 non-null float64
Automatic 1436 non-null int64
CC       1436 non-null int64
Doors    1436 non-null object
Weight   1436 non-null int64
dtypes: float64(2), int64(4), object(4)
memory usage: 163.4+ KB

Summary - after replacing special characters with nan
cars_data.info()
<class 'pandas.core.frame.DataFrame'\>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price    1436 non-null int64
Age      1336 non-null float64
KM       1421 non-null float64
FuelType 1336 non-null object
HP       1430 non-null float64
MetColor 1286 non-null float64
Automatic 1436 non-null int64
CC       1436 non-null int64
Doors    1436 non-null object
Weight   1436 non-null int64
dtypes: float64(4), int64(4), object(4)
memory usage: 123.4+ KB
```

Now let us move ahead and get the concise summary of dataframe because I really want to see the difference after changing the question marks with `nan` values because ideally there are missing values. So, let us check that. So, this is the output that we got before replacing any special characters with `nan`.

If you see I have highlighted kilometre and the horsepower column. So, kilometre basically had no missing values 1436, but after replacing all the special characters with `nan` the kilometres non null values have been decreased from 1436 to 1421 because all rest of the values actually being question marks, but that has not been treated as blank values rather it has been treated as just categories.

So, that is why you see a drop in values; that is as 1421 and after that there is also the similar case with horsepower initially there were no missing values all the 1436 has non null values, but when we consider all the special characters with as `nan` it says 1430 non null values are there other 6 observations has missing values under horsepower column.

So, this step is really important whenever you are performing any analysis if there are any other forms of missing values that is being encountered in your DataFrame you can make sure that you are replacing all those special characters with the default value. So, now, we have sorted out the issues where we replaced all the special characters with nan. We have to solve the other issue by converting variables data type to the expected data type.

(Refer Slide Time: 06:35)

The slide has a blue header with the text 'Converting variable's data types' and the GITAA logo. Below the header, there is a text box containing the following information:

astype() method is used to explicitly convert data types from one to another

Syntax: `DataFrame.astype(dtype)`

Converting 'MetColor', 'Automatic' to object data type:

```
cars_data['MetColor'] = cars_data['MetColor'].astype('object')
cars_data['Automatic']=cars_data['Automatic'].astype('object')
```

A video player window is visible on the right side of the slide, showing a woman speaking. The video player has a play button and a progress bar. At the bottom left of the slide, it says 'Python for Data Science'.

Because in the previous lecture we have seen that some of the variables have not been read properly with the correct data type and we have encountered the reasons as well now it is time to change or convert the variables data type to the expected data type. And astype is the function which is used to explicitly convert data types from one to another and the syntax being astype of data type you can basically specify to which data type you want to convert it to and you use have to use your DataFrame name.

So, here I am interested in converting the MetColor on the Automatic variable to object data type because those values represents just the categories to it, I am going to convert them to object data type. So, how I am doing here is I am accessing the MetColor from the cars_data and using the astype function, I am converting the MetColor to object by giving object inside the function.

So, whatever I am doing here that should be reflected back to original variable that is why I have equated that to cars_data of MetColor. So, now, MetColor has been

converted to object similarly you can try that for Automatic. So, I have done it here I have converted the Automatic variables also to object data type and I have reflected the changes to the original variable as well.

Now, we have converted the MetColor in Automatic to object data type, but we have already seen there are two data types that the python can handle with respect to character data type. One is being category and another one is being object. We have already changed everything to object there is also another data type called category right. So, let us see what is the impact when we have these MetColor or any categorical variable as category or as object.

(Refer Slide Time: 08:35)

The slide has a blue header bar with the GITAA logo. The main title is "category vs object data type". Below the title, there is a text box containing the following information:
nbytes() is used to get the total bytes consumed by the elements of the columns
Syntax: ndarray.nbytes
If 'FuelType' is of object data type,
cars_data['FuelType'].nbytes
11488
If 'FuelType' is of category data type,
cars_data['FuelType'].astype('category').nbytes
1460

At the bottom left, it says "Python for Data Science". On the right side of the slide, there is a small video frame showing a woman speaking.

So, here is a slide which will illustrate the impact of variables being category and the impact of variables being object data type. So, nbytes is the command that is used to get the total bytes consumed by the elements of columns, we are going to look FuelType as object data type and FuelType as category data type.

So, I am going to get the total bytes consumed by the elements with respect to object data type as well as category data type, for that nbytes can be used and the syntax would be n dimensional array.nbytes. So, it can be n dimensional array and you can use nbytes from it. So, I am using like I am accessing the fuel type from the cars_data frame and I have used.nbytes to get the total bytes consumed by the elements where the FuelType is

being object. So, when the FuelType is being object it has consumed 11488 bytes, when it is of category.

So, here if you check I have converted FuelType to category and then I am checking the total bytes consumed by the element of the column, but it has drastically dropped down to 1460 bytes. So, this is the advantage of keeping categorical variables as category data type instead of keeping them as objects. So, this will be really useful when you are dealing with a huge amount of data and you want to reduce the space that is being allocated to each and every variable or for each and every cell.

So, in that case you would go for category data type for all the string values that you have in your dataframe, but in our case we are just dealing with a less amount of data. So, there is no need to explicitly convert the variable to category, but if you see we are going to go ahead and give the FuelType as object and the slide is just for the illustration purpose.

(Refer Slide Time: 10:46)

```
Re-checking the data type of variables
```

```
Re-checking the data type of variables after all the conversions
```

```
cars_data.info()
<class 'pandas.core.frame.DataFrame'\>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price      1436 non-null int64
Age        1336 non-null float64
KM         1421 non-null float64
FuelType    1336 non-null object
HP          1438 non-null float64
MetColor    1286 non-null object
Automatic   1436 non-null object
CC          1436 non-null int64
Doors       1436 non-null object
Weight      1436 non-null int64
dtypes: float64(3), int64(3), object(4)
memory usage: 123.4+ KB
```

Python for Data Science

And we have changed the data type for few variables, now I want to recheck the data type of variables after all the conversions that I have made. So, I am again checking the concise summary of the DataFrame cars data using the.info command, if you see now all the variables are of expected data type. For example, kilometre is of float MetColor is of object Automatic color is of object. So, now we have to clean the column doors.

(Refer Slide Time: 11:13)

Cleaning column 'Doors'

Checking unique values of variable 'Doors':

```
print(np.unique(cars_data['Doors']))  
['2' '3' '4' '5' 'five' 'four' 'three']
```

Try out !

numpy.where()

- `replace()` is used to replace a value with the desired value
- Syntax: `DataFrame.replace([to_replace, value, ...])`

```
cars_data['Doors'].replace('three',3,inplace=True)  
cars_data['Doors'].replace('four',4,inplace=True)  
cars_data['Doors'].replace('five',5,inplace=True)
```

Python for Data Science

Because in the previous lecture we have seen that under the Doors column it has some string values to it. So, I just want to recap what we have seen in the Doors column, we have checked the unique values of variable Doors it turned out to have some unique values which are in numbers as well as in strings. So, those basically represents five, four, three in numbers. So, there might be cases where they have wrongly typed the numbers in terms of strings.

So, now if you want to have a consistent values like if you want to only have numerical values like 2, 3, 4 and 5 you can always go back and change all the string values to numerical values by using a command in python. So, the command is replace that is used to replace a value with the desired value you can replace any numerical to string as well as string to numericals. So, the syntax would be `DataFrame.replace`.

Inside the function I have used the value to be replaced and after that the second argument would be what is the value that you are going to replace it with. So, here I have problem in three strings. So, using a replace command I can replace all string values to numbers. So, let us take the string three first.

So, I am accessing the Doors column from the DataFrame `cars_data` and by using `.replace` command. The first argument that I have used is the value that should be replaced that is 3 and using which value that is the number 3 and I have used `inplace` is equal to true. So,

the modifications done in the DataFrame will be reflected back in the DataFrame that you are working on. Similarly I have done it for all the other strings like four and five.

So, now, I have replaced all the string values to numbers we do not want multiple values which represents the same meaning. Now, I have a consistent values like 2, 3, 4 and 5. So, now, this is not the only way where you can replace any value with the desired value there are several function that does the same one is using where. So, you can try out where function from numpy, but you can also use the where function on pandas libraries as well. So, you can try that out to replace all the string values with the numbers.

(Refer Slide Time: 13:39)

Converting 'Doors' data type

Converting 'Doors' to int64:

```
cars_data['Doors']=cars_data['Doors'].astype('int64')
```

Now, it is time to convert the Doors to int65 because whatever levels you have added, whatever values you have replaced it with it will create it as a new set of values because python cannot differentiate between the existing set of values and the newly added values. So, it will basically create new set of categories for example, 2 3 4 5 again it will have 3 4 5 because that is the categories that we have added newly.

So, in that case let us convert them back to integer64. So, that 2, 3, 4 and 5 become the integer data type that represents the number of Doors being 2, 3, 4 and 5. So, as we know that I can convert them back to integer 64 by using the command astype and I have reflected the changes to the original variable doors. So, this is to basically have the existing levels as it is if we replace any values that will get added as a new level rather

than updating it in the original level. So, just to avoid that confusion I am reconverting them back to the integer data type.

(Refer Slide Time: 14:56)

The screenshot shows a Jupyter Notebook cell with the title "To detect missing values". The code `cars_data.isnull().sum()` is run, and the output is displayed as follows:

```
Out[108]:
```

Price	0
Age	100
KM	15
FuelType	100
HP	6
MetColor	150
Automatic	0
CC	0
Doors	0
Weight	0

dtype: int64

Python for Data Science

A woman is visible in the background of the slide.

So, now we are done with cleaning the Doors column. So, now, we have to check whether there are any missing values in your Dataframe or not. So, to check the count of missing values present in each column Dataframe.isnull.sum will be used basically isnull is a function which gives you an output in terms of Boolean values that is true or false.

So, it will consider the whole Dataframe because you are applying the isnull on the Dataframe wherever you have missing values it will be represented as true, wherever the cell does not have any missing values it will be represented as false and I have used .sum to sum up all the true values because I am interested in getting the count of missing values present in each column. So, that is why I have used .sum after isnull.

So, this is the input that I have given `cars_data.isnull.sum()` that will give me the missing values present in each column. So, under the price column I do not have any missing values, but if you see under age we have 100 missing values present under the age column. So, here is the output for the command isnull.sum.

So, under the price variable we do not have any missing values, but if you see the age variable has 100 missing values and the kilometre variable has 15 missing values, similarly FuelType has 100, horsepower has 6 and MetColor has 150 missing values. So,

now, we know that there are some missing values under each columns and the missing values are not same across all the columns because it is differing with each variables. So, we have to come up with the logic approach to see how we are going to fill those missing values or how we are going to deal with this missing values at all.

(Refer Slide Time: 16:56)

The screenshot shows a presentation slide with a blue header containing the word 'Summary'. Below the header is a bulleted list of six items:

- Imported data
- Concise summary of dataframe
- Converted variable's data types
- category vs object data type
- Cleaned column 'Doors'
- Got count of missing values

In the bottom right corner of the slide, there is a small video frame showing a woman speaking. The video frame has a blue border and the text 'Python for Data Science' at the bottom left.

So, let us summarize whatever we have seen in this lecture we have imported the data by considering all the other forms of missing values as blank values by default python takes it as null. After imported the data we have got the concise summary of data where we have checked the difference after replacing all the special characters with nan values, we have also converted the variables to the expected data types.

Next, we have seen what is the impact of having a variable with the data type category and having the variable of the data type object. And we have also cleaned that column called Doors where it had few strings to it and we have replaced all the strings to numbers. So, now, we also have an idea about how many missing values are present in each column. So, in the upcoming lectures we will be seeing about how to go ahead and deal with the missing values by imputing those.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 17
Control structures & Functions

Hello all welcome to the lecture on Control structures and Functions.

(Refer Slide Time: 00:19)

In this lecture



- Control structures
 - If elif family
 - For
 - While
- Functions



So, in this lecture we are going to see about the control structures. And under the control structures I am going to cover if elif family, for loop and while loop. And, after that we are going to also look at the functions, what do we mean by function and how do we define a function using Python?

(Refer Slide Time: 00:35)

Control Structures in Python



- Execute certain commands only when certain condition(s) is (are) satisfied (if-then-else)
- Execute certain commands repeatedly and use a certain logic to stop the iteration (for, while loops)



So, let us start with control structures in Python, whenever we mean control structures I am going to use if elif family for loops and while loops. Let us see where do we use if elif and for loops and while loops. So, whenever you want to execute certain commands only when the certain condition is satisfied.

So, in that case you can go for if else statements, the condition can also be single or you can also give multiple condition, in that case you will have multiple else statements. The other one is when to use for and while loops wherever we want to execute certain commands repeatedly and use the certain logic to stop that iteration, in that case for and while loop will be helpful.

(Refer Slide Time: 01:15)

If else family of constructs



- If , If else and If-elif - else are a family of constructs where:
 - A condition is first checked, if it is satisfied then operations are performed
 - If condition is not satisfied, code exits construct or moves on to other options

Python for Data Science



So, first we will look into the if else family of constructs, if else and If-elif-else are a family of constructs, where a condition is first checked, if it is satisfied only then the operations will be performed. If, the condition is not satisfied the code exits the construct or moves on to the other options. So, whenever we use just an if statement or with an else statement or with using multiple if's and multiple else clause.

The first check would be the condition, whenever the condition is satisfied only then the code will be executed or the statement will be executed, otherwise the code exits the construct itself and moves to the other options. So, that is how the if else family of the constructs works.

(Refer Slide Time: 02:01)

If else family of constructs

Task	Command
• If construct:	• if expression: statements
• If – else construct:	• If expression: statements else: statements
• If – elif - else construct	• If expression1: statements elif expression2: statements else: statements

Python for Data Science

Let us see different task for each construct. So, first we will look into if construct, the command would be if expression colon and statements in the next line. If is a key word, if the condition is satisfied whatever condition you have given it under the expression, then the statements will get executed. Otherwise, the code exit the construct itself. Next, we will move ahead and see what is the syntax would be for If-else construct. It forms a basis from the if construct, wherever we have given the first statement, using the if keyword and followed by if keyword you have to give the expression to be checked, that is where the condition to be specified.

And, after that you give the statements that needs to be executed, if the condition is satisfied, if the condition is not satisfied give another statement under the else clause. So, using If-elif-else construct you can basically give multiple condition that needs to be checked in order to execute a statement or in order to execute any line of code. In that case the command would be like, if expression1 is being satisfied, then execute the statement, if it is not satisfied then execute the next condition.

If that is not being satisfied, then it also comes to the next else clause, where the that statement will get executed. So, the else clause statement will get executed only when the other 2 expressions are not satisfied. Basically, the other 2 conditions are not satisfied.

(Refer Slide Time: 03:37)

For loop



- Execute certain commands repeatedly and use a certain logic to stop the iteration (for loop)

Task	Command
for	for iter in sequence: statements



So, now we will look into for loop, now we are going to look into the syntax and the different loops here, in the upcoming minutes we will be looking at in terms of an example also.

So, first we will look at the syntax for for loop. So, whenever you want to execute certain commands repeatedly and use a certain logic to stop the iteration you can go for for loop. Let us look at the syntax on how to use the for loop, the command should be like for iter in sequence colon and followed by statements in the second line. So, this is the simple command that is used to construct a for loop.

(Refer Slide Time: 04:13)

The slide has a blue header bar with the GITAA logo on the right. The main title 'while loop' is in large blue text at the top left. Below it is a bulleted list: '• A while loop is used when a set of commands are to be executed depending on a specific condition'. Underneath this is a table with two columns: 'Task' and 'Command'. The 'Task' column contains 'while loop'. The 'Command' column contains the Python code: 'while condition (is satisfied): statements'. At the bottom of the slide, there is a small video player showing a woman speaking, with the text 'Python for Data Science' visible.

Next, we will see about the while loop and the syntax for while loop: a while loop is used, when a set of commands are to be executed depending on a specific condition. Basically, a while loop will be executed as long as a condition is true, whenever the condition becomes false, whenever the condition you have given becomes false, the while loop execution will stop.

So, the task being here is the while loop and the command to construct a while loop is while is the function inside the braces, you have to give the condition. And, as long as that condition is true the statements we have given in terms of statement will get executed, whenever that condition becomes false, that is when the while loop will get stop executing.

(Refer Slide Time: 04:57)

Example: if else and for loops



- We will create 3 bins from the 'Price' variable using *If Else and For Loops*
- The binned values will be stored as classes in a new column, 'Price Class'
- Hence, inserting a new column

```
cars_data1.insert(10, "Price_Class", "")
```



So, now we are going to see an example to know how we can use if else and for loops. So, here we have been working with the data set called Toyota, where we are seeing how to read them and how to do basic pandas data frame operations? From there I have just used a single variable called price, which represents the price of the cars, price of the pre-owned cars. So, I am using the price variable from the Toyota data, where I am going to create 3 bins from the 'Price' variable using if else and for loops.

Because, the price variable is a continuous variable where it just has the values for the price of the pre-owned cars, if I want to segregate those prices into 3 buckets, then I can use if else and for loops, let us see how to we do that. The binned values as I mentioned I am going to bin the values. So, those binned values will be stored as classes in a new column as price class. So, in that case I should be creating a new column to the existing DataFrame.

So, now, I am going to create a new column to the existing Toyota data, where I have read and kept it as cars_data one. So, using the .insert function, I can give the position to which the column should be added and the column name and I have given blank. So, that all the it will create a column with the blank values. So, now a new column has been created as price class so, that we can store all the bin values as classes.

(Refer Slide Time: 06:25)

Example: if else and for loops



```
for i in range(0,len(cars_data1['Price']),1):
    if (cars_data1['Price'][i]<=8450):
        cars_data1['Price_Class'][i]="Low"
    elif ((cars_data1['Price'][i]>8450) & (cars_data1['Price'][i]<11950)):
        cars_data1['Price_Class'][i]="High"
    else: cars_data1['Price_Class'][i]="Medium"
```

- A for loop is implemented and the observations are separated into three categories:
 - Price
 - up to 8450
 - between 8450 and 11950
 - greater than 11950
- The classes have been stored in a new column 'Price Class'

Python for Data Science



So, using if else and for loops, I am going to convert all the values into 3 categories, where the categories represent the range of the price; one is being “Low” and other one is being “High” and other one is being “Medium”. So, if I want to segregate those values into 3 categories I want to basically give some condition in which it can happen. So, I am going to use that using if statement.

So, what I am starting here is I have used the for loop I have initiated the for loop here, where I have given the iter in sequence, where I is the indexing variable in the sequence. I have given the sequence using the range function, where I have given the starting value as 0 and it should have the value till the length of price. So, the value will be 1436 and I have given comma 1, then it will the iteration will happen in the steps of 1.

So, now I have given from which the iteration should happen. Now, I have given the sequence in which the iteration should happen. Now, this is time to give the condition on price. The first condition being I want to make I want to make some records, I want to bucket some records as loop by giving the condition, whenever the price is less than or equal to 8450, then keep them as in the range “Low” right.

In that case I can just give a condition, saying access the price variable from the cars_data1 dataframe and give a condition here, whenever it is less than or equal to 8450, then execute this statement that is cars_data1 price class becomes “Low”.

Basically, equate a value “Low” to the new variable price class. So, this will happen for each and every row and it will check whether the price of the car is less than or equal to 8450 or not. If it is then it basically equate them equate the basically then it will name the row as “Low”. So, there is another condition using elif statement, because I need to give 2 conditions here; one is on “Low” and one is on “High”. And, whatever is not being satisfied with “Low” and “High”, I keep them as “Medium”.

And all those rows will be kept it as “Medium”. So, in this case I have 2 conditions here; one is price less than or equal to 8450. And, the other one is being whenever the price is greater than 11950, then in that case the price will be, in that case those rows will be named as “High” and that will be stored in the column price_class. So, whenever these two conditions are not satisfied, whenever any row is greater than 8450 and less than 11950, then all those rows will be named as “Medium”. So, there is a bound here.

So, the whatever rows that are being named as “Medium” will have all the rows wherever the price is greater than 8450, and wherever the price is less than or equal to 11950. So, this is how I can give using for if elif and else. So, if you see that to just to summarize whatever we have done it in for loop and if else, a for loop is implemented and the observations are separated into 3 categories right now.

So, the price being upto 8450 and between 8450 and 11950 and greater than 11950, and we keep the price less than or equal to 8450 as “Low” and we keep between 8450 and 11950 as “Medium” and whenever the price exceeds 11950, then we keep them as “High”. And, we know that the classes have being stored in a new column called price class.

So, in each of the records of price class there will be a label called row “Low” “High” or “Medium” that is exactly based on the condition, which is being represented here. And, you have done that so, using for loop we have seen how to do the iteration and using the if and else clause we have seen how to give condition based on a variable?

(Refer Slide Time: 11:01)

Example: while loop

```
i=0
while i<len(cars_data1['Price']):
    if (cars_data1['Price'][i]<=8450):
        cars_data1['Price_Class'][i]="Low"
    elif ((cars_data1['Price'][i]>8450) & (cars_data1['Price'][i]<11950)):
        cars_data1['Price_Class'][i]="Medium"
    else:
        cars_data1['Price_Class'][i]="High"
    i=i+1
```

- A while loop is used whenever you want to execute statements until a specific condition is violated
- Here a while loop is used over the length of the column 'Price_Class' and an if else loop is used to bin the values and store it as classes

Python for Data Science

So, next let us see an example for while loop. So, a while loop is used whenever you want to execute statements until a specific condition is violated. Here, I am going to use a while loop, over the length of the column price class, and an if else loop is used to bin the values and store it as classes. So, whatever we have done in the previous slide I am going to repeat the same thing using while loop. So, you can do that using both for and while.

So, here I have initialized my indexing variable as 0, the difference between both for and while loop is in the previous slide, you would have given your iteration step in the sequence itself using your for, but in this while loop you are giving your you are just initializing your indexing variable as 0 and you will give the iteration step at the last only. Because, the first check of the while loop is the condition check. So, the while loop will be executed as long as i is less than length of cars of data that is 1436, the while loop will get stop executing whenever it exceeds that condition.

Whenever your i becomes 1437 your while loop will get stop executing. So, next I have a condition here, the same condition being represented here with the same if elif and else clause the difference being here is you give the iteration steps at the end of the loop, whenever you are using a while loop and you give the iteration steps at the beginning of the loop itself using a for loop. So, here if you recall whenever the price is less than or equal to 8450 then keep them as "Low" under the column price class.

And, whenever it exceeds 11950 then keep them as “High” under the variable price clause and whenever both the conditions are not satisfied, whenever the price is greater than 8450 and less than or equal to 11950, then in that case that observations will be named as “Medium”. So, now, we have seen the examples for both for loop and while loop to basically bucket all the price values into 3 categories as “Low” “High” and “Medium”.

There might be other functions which will do it thereby there are so, many inbuilt function that does this, but using a for loop when you have a control on whatever you are doing with the steps, then you can use a for loop and while loop. We have now 3 bins, now 3 categories now “Low” “High” and “Medium”. So, we do not know how many records fall into row, and how many records fall into “Low”, and how many records fall into “High”, and how many of them fall into “Medium”?

So, let us just see how you are observations have been categorized? So, now, we have basically used a loop to combine all the price values into three categories; one as “Low” “Medium” and “High”. Now, I want to check how the categorization has happened.

(Refer Slide Time: 14:11)

Example: while loop

• `Series.value_counts()` returns series containing count of unique values

```
cars_data1['Price_Class'].value_counts()
```

Out[14]:

Price Class	Count
Medium	751
Low	369
High	316

Name: Price_Class, dtype: int64

So, in that case I will be looking at frequencies of each categories, that is “Low”, “Medium”, and “High”, I can get the frequencies of each categories using `value_counts` function. So, whenever you have a series you can use `value_counts` that basically returns series containing the count of unique values.

I want to check the count of unique values under the price class, that is the variable name and cars_data one is the dataframe name, and.value_counts is the function, if you use that it basically gives an output which is shown here. So, you have 751 observation that falls under the “Medium” category that is basically those cars are in the range of “Medium”, those cars have the price in the range of “Medium”.

And, you have 369 observation with the “Low” range and you have only 316 observation with the “High” range. And, the name being price class and the data type of the output is being in 64. So, now, we will see how to basically convert your numerical values into a categorical variable right, because now we have converted the numerical variable price into the categories as “Low”, “Medium” and “High”, that becomes a categorical variable.

Now, that is why we have checked the unique count of each categories, see you can if you want to do that you can use either a for loop or a while loop. So, this is where all the while loop and for loop comes into play.

(Refer Slide Time: 15:43)

The screenshot shows a presentation slide with a blue header bar. On the left, the title 'Functions in Python' is displayed in white text. On the right, there is a logo for 'GITAA' with the tagline 'Transforming careers' below it. The main content area contains a bulleted list of six points about functions in Python. To the right of the list, there is a small video frame showing a woman speaking. At the bottom left of the slide, the text 'Python for Data Science' is visible.

- A function accepts input arguments and produces an output by executing valid commands present in the function
- Function name and file names need not be the same
- def function_name(parameters):
statements
- A file can have one or more function definitions
- Functions are created using the command def and a colon with the statements to be executed indented as a block
- Since statements are not demarcated explicitly, It is essential to follow correct indentation practises

Let us see about the functions in Python. Basically a function accepts input arguments and produces an output by executing the valid commands present in that function. And, the function name and the file name need not be of the same, because you can have a different file name and a different function name, that holds good in Python.

And, a file name can have one or more function definition. Say, if you have a file where you have defined a function, that function file can have one or more function definition, there would not be any issues while you are calling a function in a different file. And, the functions are created using the command def and a colon with the statements to be executed indented as a block.

So, this is how you define a function you use a def function and the function name is followed by that, say inside the parenthesis you basically need to give the parameters based on which your calculations will be done. So, using the statements here you will give the basically an equation or an expression, that should be calculated that should be solved based on the parameters, that you have given inside the function name parameters.

And, since the statements are not demarcated explicitly, it is essential to follow correct indentation practices. Because, other programming languages the Python does not support the curly braces, for any control structures or function rather it uses the indentation to explicitly show the demarcation. So, the indentation should be followed. So, your statement should be exactly slightly away from your first three letters of your function name, whenever you type a colon and give an enter, it will automatically comes with an indentation. So, it is suggested to not to change that indentation.

(Refer Slide Time: 17:33)

Example: functions



- Converting the `Age` variable from months to years by defining a function
- The converted values will be stored in a new column, '`Age_Converted`'
- Hence, inserting a new column

```
cars_data1.insert(11, "Age_Converted", 0)
```

So, now let us see an example on how to define a function? So, we are going to define a function which will allow us to convert the age variable from months to years by defining a function. I am also using the age variable from the Toyota data, that we have that we have working with. So, in that case age is being represented in terms of months, but that does not convey me the exact information or the efficient information, but that is not the exact way where I can infer the age, in rather than keeping the age in terms of months I can also keep the age in years.

So, I want to convert the age variable from months to years by defining a function. So, the converted value should be stored in a new column. So, I do not want to touch the existing column rather I am going to store all the values in the new column called ageConverted.

So, I should be creating a new column called ageConverted now. So, I have created ageConverted using the same.insert function, where I want to keep the age converted in the 11th position and initially I want to have all the values as 0, that is what I have given as 0 here. So, once you have executed this line a new column will be created as ageConverted.

(Refer Slide Time: 18:45)

Example: functions



- Here, a function `c_convert` has been defined
- The function takes arguments and returns one value

```
def c_convert(val):
    val_Converted = val/12
    return val_Converted

cars_data1["Age_Converted"] = c_convert(cars_data1['Age'])
cars_data1["Age_Converted"] = round(cars_data1["Age_Converted"],1)
```



So, now let us define a function to convert the age from months to years. So, here I am defining a function `c_convert` and the function takes arguments and returns only one value. So, `def` is the key word that is used to create a function definition or the command

that is used to function, that is used to create a function definition. And, the function name is `c_convert` and the inside the function I have given the argument called `val` and followed by the semicolon in the next line I have a variable called `valConverted` that represents value converted.

How am I going to convert I am going to convert that value by dividing, whatever value that is given here by 12. So, this is the default argument. So, whenever you called the any function you can use the `c convert` and give a variable there or give a value there, that will be divided by 12 and it will return the value converted `val`. And, it will return the value or that is stored in the `valConverted`. So, now, using this function definition, I can basically convert the age variable into months, I can basically convert the age variable from months to years.

How, I can basically use the same function here that is `c_convert` and inside the function I just basically need to pass in the arguments for `val` to divide any number by 12. So, I want to divide all the numbers from `age` by 12. So, I have given `cars_data1['Age']`. So, all the observations under the `age` column will be divided by 12. So, there I get the `age` converted to years. So, if you see I am storing that into the new column called `ageConverted` from the data frame `cars_data`.

So, this will this new variable will have all the `age` that has been converted into years. And, whenever you are doing any numerical operation Python always comes with 5 to 6 decimal points, I do not want to have a variable which has so, many decimal that also does not convey the exact information of year. So, I want to round it off to only 1 decimal. So, that it does not have 5 to 6 decimal point it will just have a digit after a decimal point.

(Refer Slide Time: 21:11)

Function with multiple inputs and outputs

Function with multiple inputs and outputs

- Functions in Python takes multiple input objects but return only one object as output
- However lists, tuples or dictionaries can be used to return multiple outputs as required

Variable 1
Variable 2

function_mimo

Output: Multiple values (list, tuple, dictionary)

Python for Data Science

GITAA Transforming Careers

So, till now we have been seeing about a function which accept a single argument or multiple arguments and which will arrive at a output value single output value. Now, we are going to move ahead and see how to define a function with multiple inputs and arrive at a multiple outputs. So, function in Python takes multiple input objects, but return only one object as output. So, you can have variable one and variable 2 as inputs to your function.

But, your output will be in a form of only single object, but that object can contain multiple values like a lists can contain multiple elements and a tuple can contain multiple elements and a dictionary can have multiple keys in values. So, in that case you will have multiple results in form of a single object. So, like I said list tuples or dictionaries can be used to return multiple outputs as required.

(Refer Slide Time: 22:11)

Example: function with multiple inputs and outputs



- Converting the **Age** variable from months to years and getting kilometers (**KM**) run per month
- The converted values of kilometer will be stored in a new column, '**km_per_month**'
- Hence, inserting a new column

```
cars_data1.insert(12, "Km_per_month", 0)
```



Let us see an example to see how function with multiple inputs and output works. So, here by defining function with multiple inputs and outputs I am going to do two things; one is converting age variable from months to years and another one is getting kilometers run per month. So, the converted values of kilometer will be stored in a new column called `km_per_month` we have already created one new variable for age as age converted.

So, I am just going to create one more for `km_per_month` that is also using the same insert function where I have set the position for the kilometer per month variable and the variable name being kilometer per month and I need to fill basically all the values with 0s initially so, I have given 0 here.

(Refer Slide Time: 23:01)

Example: function with multiple inputs and outputs



- A multiple input multiple output function `c_convert` has been defined
- The function takes in two inputs
- The output is returned in the form of a list

```
def c_convert(val1,val2):
    val_converted = val1/12
    ratio          = val2/val1
    return [val_converted,ratio]
```

Python for Data Science



So, now let us define a function which accepts multiple arguments and which will also give us multiple results as a single object. So, here is the function definition. So, as I mentioned a multiple input multiple output function `c_convert` has been defined, this is the function `c_convert` and the function also takes in 2 inputs value 1 and value 2.

And, the output is going to be returned in the form of a list that is how I have defined a function, if you see from the start I have defined a function called `c_convert` I have given 2 inputs here value 1, value 2. And, I am going to get two output; one is value converted from for the age from months to years and the other one is the ratio. So, `val1` and `val1` divided by 12 basically divides all the observations under the age variable and give you an output and that will be stored in `val_converted`.

And, now I have created another variable called `ratio` where my interest is to convert all the kilometers run per month. So, in that case I am going to divide each and every observations of the kilometer by the value 1, where I am going to divide value 2 by value 1. And, it will return both value converted and ratio in a form of list, because I have given the values inside the square brackets so, that the value will be returned in a form of list.

(Refer Slide Time: 24:31)

Example: function with multiple inputs and outputs



- Here, **Age** and **KM** columns of the data set are input to the function
- The outputs are assigned to '**Age_Converted**' and '**km_per_month**'

```
cars_data1["Age_Converted"],cars_data1["Km_per_month"] = \\\n    c_convert(cars_data1['Age'],cars_data1['KM'])
```

```
In [49]: cars_data1.head()\nOut[49]:\n      Price  Age      KM FuelType  HP MetColor Automatic  CC  Doors \\\n0  13500  23.0  46986.0  Diesel  90.0     1     0  2000     3\n1  13750  23.0  72937.0  Diesel  90.0     1     0  2000     3\n2  13950  24.0  41711.0  Diesel  90.0    NaN     0  2000     3\n3  14950  26.0  48800.0  Diesel  90.0     0     0  2000     3\n4  13750  30.0  35900.0  Diesel  90.0     0     0  2000     3
```

```
Weight  Price_Class  Age_Converted  Km_per_month\n0   1165        High       1.916667  2042.869565\n1   1165        High       1.916667  3171.173913\n2   1165        High      2.000000  1737.958333\n3   1165        High      2.166667  1846.153846\n4   1170        High      2.500000  1283.333333
```

Python for Data Science



So, let us see how do we do that? So, here age and kilometer columns of the data set are going to be the input to the function, because I am going to convert age from months to years, I am going to convert kilometers run and I am going to get the km_per_month. And, the outputs are going to be assigned to age converted and kilometer per month where we have created 2 new variables. And, here the outputs are going to be assigned to age converted and kilometer per month, because I am going to save my output simultaneously.

So, as you know in Python you can assign multiple values too by giving multiple variable names. So, that is what I am going to give in here. So, here the variable the first variable name is age converted from the dataframe cars_data1. And, the second variable is kilometer per month from the cars_data1. And, what I am going to save here is the output from the c_convert function, which I have defined it in the previous slide. So, c_convert is the function and the input would be cars_data of age.

So, age will be the first input and the kilometer would be the second input. And, I have stored whatever value I am getting it using the first argument as age converted to age converted and whatever I will get it using the second variable will be stored into the second variable called kilometer per month. So, let us look at an output after using the c convert function what is the output that we are likely to get. I have used.head function to

basically look the first 5 rows of my dataframe just to see how my variables have been populated.

So, if you see the price class this is using the for loop and while loop which we have done, where we have bucketed all the price values. If, you look at here there are price values like 13500, we have given some conditions where, if it exceeds some value, then keep it as “High” “Low” and “Medium”, then in that case the first 5 rows have been stored as “High”. And, if you see we have converted the age which were in months to years I have just divided every row with 12. So, I got 1.916667.

But, if you see here this is not the rounded off value. So, this snippet just gives you the output with 5 decimal values, you can also round that to 1 decimal point, because this value does not make sense for you can also round off your values to 1 decimal points. So, the value will be 1.9 and you can also get the kilometer run per month. Here, you have the kilometer which is like 46986 and if you want to get it for month, then we have used the function and we have also got how many kilometers that the car has travelled per month.

(Refer Slide Time: 27:33)

The screenshot shows a presentation slide with a blue header bar. On the right side of the header bar is the GITAA logo, which consists of a stylized book icon and the text 'GITAA' with the tagline 'Transforming careers.' Below the header, the word 'Summary' is written in a large, bold, blue font. The main content of the slide is a bulleted list of topics:

- Control structures
 - If elif family
 - For
 - While
- Functions

At the bottom left of the slide, there is a small watermark-like text 'Python for Data Science'. In the bottom right corner, there is a video frame showing a woman speaking, likely the instructor.

So, now we have come to the end of lectures. So, let us summarize whatever we have done till now. We have seen about the control structures where we have covered if elif family followed by that we have seen for and while loops with examples. We have also seen about the functions where the function can accept multiple inputs and give you a

single output, we have also seen how a function can accept multiple inputs and give you multiple output as a single object.

Thank you.

Python for Data Science
Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 18
Exploratory data analysis

Welcome to the lecture on Exploratory Data Analysis.

(Refer Slide Time: 00:18)



In this lecture

- Frequency tables
- Two-way tables
- Two-way table - joint probability
- Two-way table - marginal probability
- Two-way table - conditional probability
- Correlation

Python for Data Science

A video player window shows a woman with long dark hair, wearing a teal top, speaking. The video player has a blue header with the text "GITAA Learning World".

So, in this lecture we are going to explore more on the data that we were working on using frequency tables two-way tables. Followed by that, we are also going to look at how to get the joint probability out of two-way tables. We will also be looking at how to get marginal probability and conditional probability using two-way tables. So, we will see in detail about each of the topic listed here. At last we are also going to look at a measure called correlation because, all of the points which are listed above are to interpret or to check the relationship between categorical variables.

But we will also have numerical variables in our data frame, in which case we will also be looking at. So, in that case if you want to check the relationship between two numerical variables, there is a measure called correlation that is what we are going to see in this lecture. We will also be seeing in detail about what correlation measure is about.

(Refer Slide Time: 01:16)

The screenshot shows a slide titled "Importing data into Spyder". It lists two imports:

- `import os` — 'os' library to change the working directory
- `import pandas as pd` — 'pandas' library to work with dataframes

Below the imports, a code snippet is shown in a terminal window:

```
os.chdir("D:\\Pandas")
```

A woman, identified as the speaker, is visible on the right side of the slide.

So, before exploring on the data, we need to import the data into spyder to work on that. So, prior to importing data, we need to import the necessary libraries that are acquired for importing any data into spyder. So, let us do that first. First we are importing the os library we use os library to change the working directory. Next we will also be working with data frames because once we read any data into spyder that becomes a data frame. So, to deal with data frames we need to load the library called pandas.

And we have imported it as pd because pd is just analyzed to the library called pandas. So, now, we have imported the necessary libraries to change the working directory using the command `os.chdir`. `chdir` chance for changing directory and inside the command you can just give the path from which you are going to access the data from.

(Refer Slide Time: 02:09)

The slide has a blue header with the GITAA logo. The main title is "Importing data into Spyder". Below it, there's a bullet point "Importing data" followed by a code snippet:

```
cars_data = pd.read_csv('Toyota.csv', index_col=0,  
na_values=['??', '????'])
```

Below the code is a screenshot of a Jupyter Notebook cell showing a DataFrame with 4 rows and 11 columns. The columns are: Index, Price, Age, KM, FuelType, HP, Mpg, Automatic, CC, Gears, and Weight. The data looks like this:

Index	Price	Age	KM	FuelType	HP	Mpg	Automatic	CC	Gears	Weight
0	13500	23	46986	Diesel	90	1	0	2000	three	1165
1	13750	23	72937	Diesel	90	1	0	2000	3	1165
2	13950	24	41711	Diesel	90	nan	0	2000	3	1165
3	14950	26	48000	Diesel	90	0	0	2000	3	1165

Below the DataFrame, there's another bullet point "Creating copy of original data" followed by the code:

```
cars_data2 = cars_data.copy()
```

A small video thumbnail of a woman speaking is visible on the right side of the slide.

Now, let us import the data into spyder. So, we have a dataset called Toyota.csv which are nothing but the details of the cars. The details of the cars having captured in terms of various attributes like price, age, kilometre, FuelType you can look at the snipped below here. So, since this is csv data we need to use read_csv command to read in csv files and that is from the panda's library.

So, that is where we have used pd.read_csv. And inside which we have just given the filename and we have set index_col as 0; just to make sure that we are setting the first column as our index column. Since we have already worked with this Toyota data, we know that there are some missing values that are in the form of question marks. This question marks these question marks does not convey any message from the data or about the data.

So, in that case we will be considering this as missing value, but here I have given it under na_values because I am going to consider these special strings with the default none values of python. Because, python offers several functions which will allow us to deal with the default nan values.

In this case if I want to perform any operations by considering these missing values it will be a tedious process. Rather I can just consider these strings as default nan values which also the representation of not available values. While reading itself, we are considering all the question marks as nan value.

So, that we can perform all the operations that are related to missing values; and the snippet given below will give you an idea about how the data will look like. Now we have read the data into spyder we are going to create a copy of the original data, so that whatever operations or modifications we are going to do on the data frame will not be reflected in the same data set itself. Rather we can just have a copy of it so that we can cross verify with the original data at any point of analysis.

So, in that case we are going to create a copy using the dot copy function proceeded with the data frame name. So, here cars_data is a data frame name and using .copy I am creating a copy of data and I am saving it into a new object called cars_data 2. So, cars_data two becomes my new data frame which was copied from the original data frame. Now we can use this data frame to do the further analysis or to do operations so that the original data will not be modified.

(Refer Slide Time: 04:50)

The screenshot shows a Jupyter Notebook interface. At the top, there's a header 'Frequency tables' and a logo for 'GITAA'. Below the header, a green box highlights the command `pandas.crosstab()`. To its right, a blue circle labeled 'Size of data' contains two entries: '1436 - Original data' and '1336 - after dropping nan values'. The main content area displays the code: `pd.crosstab(index=cars_data2['FuelType'], columns='count', dropna=True)`. Below the code, the output 'Out[3]:' is shown in a table:

FuelType	count
CNG	15
Diesel	144
Petrol	1177

A yellow box highlights the 'Petrol' row. A text annotation next to the table says 'Most of the cars have petrol as fuel type'. In the bottom right corner of the slide, there's a small video thumbnail of a woman speaking.

So, now we are going to look how we are going to create a frequency table before creating we need to know what frequency table is about. We have multiple variables in our data frame and if you want to understand the data more, you basically want to check the relationship that exist between the variables. But we cannot just check the relationship between all the variables we can do one by one.

For example, we can check the relationship between any categorical variables using cross tabulation or if you want to do univariate analysis on a categorical variable you can

also create a frequency table. So, that you will know what is the frequency of each categories that are available under a variable. Now, let us see how to create a frequency table? So, crosstab is the function that comes from the panda's library, which is used to compute a simple cross tabulation using one two or more variables by default it creates a frequency table of the factors. So, now, let us see how to create a frequency table. So, I have used pd.crosstab that is the function that is used to create a frequency table.

And as an input to the function I have used it as index is equal to cars 2 of FuelType; that means, that I am interested in getting the frequencies of the categories that are available under the variable FuelType. And we know that that is from the data frame cars data 2 and the variable of interest should be given under the parameter called index.

We also need to have the corresponding frequencies of it. So, basically it give you the count for each categories of FuelType. And since we know that we have so many missing values in our data frame, we do not want to consider that while we are interpreting from the frequency table. In that case you can drop all those missing values by setting dropna is equal to true.

By setting that you will get rid of all the records wherever there are missing values. So, you will get the frequencies for each categories for the records where there are no missing values. By setting dropna is equal to true, if you look at the original data size and the data size after dropping nan values you will get an idea about how many records we have lost.

So, the original data size is about 1436 rows and after dropping the missing values we are left out with only 1336 rows because there were 100 records where the FuelType were missing that is why we are left out with only 1336 records. Now, we are going to create the frequency table by considering only 1336 records.

So, now let us see how the output would look like. So, if you see here you have the variable here and you have the corresponding categories under the variable FuelType. So, basically there are three categories under the variable FuelType, CNG, petrol and diesel. And you can also look at the corresponding frequencies of each of it. So, it is very evident from the output, there are only fifteen cars whose FuelType is of CNG. And there are 144 records or 144 cars have the FuelType as diesel. And if you see here petrol

has the frequency as 1177. So, in this case most of the cars have petrol type as FuelType because there are only few cars whose FuelType is CNG and diesel.

So, in this case you will have an idea about though there are so many categories that are available under FuelType most of the cars FuelType are of petrol.

(Refer Slide Time: 08:31)

Two-way tables

pandas.crosstab()

- To look at the frequency distribution of gearbox types with respect to different fuel types of the cars

```
pd.crosstab(index = cars_data2['Automatic'],
             columns = cars_data2['FuelType'],
             dropna = True)
```

Out[5]:

FuelType	CNG	Diesel	Petrol
Automatic	15	144	1104
0	0	0	73

Python for Data Science

GITAA

A woman in a teal shirt is visible on the right side of the slide.

So, in the previous example we have considered only one categorical variable just to get the frequency distribution of each categories. Now we can also have one more variable to check the relationship between those two categorical variables. If you want to check the relation between two categorical variables you can go for two-way tables.

And here we are going to use the same function to create a two-way table and in this example we are going to look at the frequency distribution of gearbox types with respect to difference FuelTypes of the car. In the previous example you have just looked at the frequency distribution of FuelType, but here we are going to look at the frequency distribution of gearbox with respect to the different FuelTypes of the car.

So, let us see how to do that and if you want to know more about gearbox types that are available for the data set that we have here is a snippet. So, gearbox type have been represented using a variable called Automatic, it has two values 0s and 1s and 0 is the representation of the car having a Manual gearbox and one is representation of the car

having an Automatic gearbox. Now let us see how to create a two-way table using crosstab function.

So, the function remains the same that is pd.crosstab, the first parameter that goes into the function is index under that I have just specified the variable as Automatic. So, under the index variable I have specified the variable as Automatic and under the columns you can specify one more variable that is FuelType because we want to look at the frequency distribution of gearbox type with respect to different FuelTypes of the car.

And I am also going to remove all the missing values from the data frame. So, by setting na is equal to True you mean that you it will consider the two-way table will be created by considering only the rows where both Automatic and FuelType are found. That is there should be no missing values in both Automatic and FuelType in that case only it will create a frequency table.

Because, if you have a missing value under a Automatic column and if you have a field value under the FuelType column it will not be able to get the count of it. Rather it will just over that particular row itself and you will be left out with the rows where there are no missing values in both Automatic and FuelType column.

So, now let us look at the output to check the relationship between those two variables. So, since I have just since I have given Automatic under the index argument and FuelType under the columns argument the output is a representation where rows correspond to Automatically and columns corresponds to FuelType. So, from the output it is very evident that. So, now, let us see how to interpret the output. So, if you see 15 here so, you can see the.

So, you can see the value 15 here; that means, that there are only 15 records where the FuelTypes of CNG and the car has a manual gearbox. And there are only 144 records where the FuelType of the car is Diesel and also the car is having a manual gearbox. And interesting and the interesting thing about output is you can see some 0s here; that means, that the cars with Automatic gearbox does not have the FuelType of either CNG or Diesel. If the car is of Automatic gearbox then it has only the FuelType as Petrol. So, low cars have the FuelType as CNG or Diesel given the gearbox is of Automatic.

So, this is very interesting about the relationship between Automatic and FuelType. So, now, this gives us the idea about what is the relationship that exist between the Automatic and FuelType variables.

(Refer Slide Time: 12:25)

Two-way table - joint probability

pandas.crosstab()

- Joint probability is the likelihood of two independent events happening at the same time

```
pd.crosstab(index      = cars_data2['Automatic'],
             columns    = cars_data2['FuelType'],
             normalize = True,
             dropna    = True)
```

	FuelType	CNG	Diesel	Petrol
Automatic	0	0.010801	0.108011	0.828083
	1	0.000000	0.000000	0.053105

Out[16]:

	FuelType	CNG	Diesel	Petrol
Automatic	0	0.010801	0.108011	0.828083
	1	0.000000	0.000000	0.053105

So, we have looked at the output in terms of numbers. There is also a way were you can convert the table values in terms of proportion and that is what we mean by joint probability. By converting the table values from numbers to proportion you will get a joint probability values you will get the joint probability values that is also using the same function crosstab. Let see how to do that, we are going to use the same function crosstab to arrive at the joint probability values.

What do you mean by joint probability first? Joint probability is the likelihood of two events if joint probability is the likelihood of two independent events happening at the same time. So, if you have two independent events happening at the same time what is the probability of it, that is what the joint probability give you let see how to do that.

The all the other quote remains the same, but you just need to add one more parameter called normalize is equal to true. If you set normalize is equal to True you are basically converting all the table values from numbers to proportion that is what normalize means. Now let us see how the output will look like. So, you have the same table here, but the values have been converted from numbers two proportions.

You can interpret the output like, the joint probability of the car having a manual gearbox and having the FuelType of CNG is only 0.01. But if you look at a value here that is 0.82 that represents that the joint probability of the car having a manual gearbox and the FuelType is also petrol the probability is really high there.

And if you see here there is no probability that you will get a car with an Automatic gearbox as well as with the FuelType CNG or diesel, but all these are from the data that we have read now. So, all these are interpretation that we have made are based on the data that we have now, there can be cases where the interpretations can be different with respect to different sets of records.

(Refer Slide Time: 14:30)

```

Two-way table - marginal probability
pandas.crosstab()

• Marginal probability is the probability of the occurrence of
the single event

pd.crosstab(index      = cars_data2['Automatic'],
            columns     = cars_data2['FuelType'],
            margins    = True,
            dropna    = True,
            normalize = True)

Out[17]:
FuelType      CNG   Diesel   Petrol    All
Automatic
0           0.010801 0.108011 0.828083 0.946895
1           0.000000 0.000000 0.053105 0.053105
All          0.010801 0.108011 0.881188 1.000000

```

probability of cars having manual gear box when the fuel type are CNG or Diesel or Petrol is 0.95

Now, we going to look at how to get the marginal probability using the two-way table. We are going to use a same function but by just tweaking or by just adding one more parameters we will be able to arrive at different types of probability values. So, here we are going to look at marginal probability. So, marginal probability is the probability of the occurrence of the single event, it will consider only the occurrence of a single event alone.

So, here is the code for that, we have used the same pd.crosstab function. So, here index and columns parameters remains the same and we have used dropna is equal to True and normalize is equal to True because we want all the table values in terms of proportions or the probability values. And I have also set margins is equal to True in order to get the

marginal probability value by setting margin is equal to True. You are basically going to get the rows sums and the column sums for your table values.

Let us see how the output will look. So, here is the output in the previous example you would have got till here you did not get the rows sum and the column sum of it. But by setting margins is equal to True you will get the row sums and the columns as well in the name of all. What is we mean by marginal probability the highlighted values are nothing, but the marginal probability values and how do interpret these values.

So, if you take the first value that is 0.946895. So, now, how can we interpret from the value 0.94 because the 0.94 value is nothing but the probability of cars having manual gearbox when the FuelType is of either CNG or Diesel or Petrol. You can infer the 0.88 value as the probability of the car having a FuelType as Petrol and when the gearbox type is of either Automatic or manual. So, this is what you can get and if you sum up everything the total probability value will be 1.

(Refer Slide Time: 16:44)

Two-way table - conditional probability

pandas.crosstab()

- Conditional probability is the probability of an event (A), given that another event (B) has already occurred
- Given the type of gear box, probability of different fuel type

```
pd.crosstab(index      = cars_data2['Automatic'],
             columns    = cars_data2['FuelType'],
             margins   = True,
             dropna   = True,
             normalize = 'index')
```

FuelType	CNG	Diesel	Petrol
Automatic	0.011487	0.114068	0.874525
Manual	0.000000	0.000000	1.000000
All	0.010801	0.108011	0.881188

Out[19]:

Row sum = 1

Python for Data Science

Now let us move on to get the conditional probability using the two-way table. So, here also we are going to use a same function that is panda's.crosstab and let us see what conditional probability is about. So, conditional probability is the probability of an event A, given that another event B has already occurred.

For example, if you want to get the probability of an event A; by considering another event has already occurred then you call it as a conditional probability. And now what is the example that we are going to look using conditional probability is that, given the type of gearbox what is the probability of different FuelType?

So, let us see how to get that, but if you see here the first four parameters remains the same, we have just tweaked the normalised parameter from we have just tweaked the normalized parameter we have initially said that as true, but we have changing it to index just to get the conditional probability values.

So, now we are going to look at the output to get the inferences. So, if you see here this is a cross tabulation of Automatic and FuelType variable and all the values are in terms of probability values. Since we have set normalize is equal to index you will get the row sum as one because that is what we mean by the conditional probability. So, given the gearbox types is of manual the probability of getting a CNG FuelType is 0.01 and the probability of getting diesel FuelType is 0.11 and the probability of getting FuelType as petrol is really high when compared to the other FuelTypes.

So, this gives you an idea about for any manual gear box petrol can be the FuelType because at the max we are getting the probability is for petrol. So, there is a really high probability value that you can get. So, from the high probability value you can say that so, from high probability value of 0.87 you can say that for any car which are of manual gearbox the probability is really high for having a petrol type for having a FuelType as petrol.

And similarly you can see here there is no property that you can get because there is no probability value that you can get for CNG and diesel. Because the probability is 0 and the probability is 1 for petrol because for all the Automatic gearbox cars the FuelType is only petrol. This we know that from the previous examples as well. So, this is how we get the conditional probability. Here we have got the rows sum to 1; we can also get the column sum to 1 in that case you will be looking at the cross table in terms of given the type of fuel being used for the car. So, given the type of fuel, you will get the probability of different gearbox types.

(Refer Slide Time: 19:42)

Two-way table - conditional probability

pandas.crosstab()

Conditional probability is the probability of an event (A), given that another event (B) has already occurred

```
pd.crosstab(index      = cars_data2['Automatic'],
             columns    = cars_data2['FuelType'],
             margins   = True, dropna = True,
             normalize = 'columns')
```

Out[20]:

FuelType	CNG	Diesel	Petrol	All
Automatic	1.0	1.0	0.939734	0.946895
0	1.0	0.0	0.060266	0.053105
1	0.0	0.0	0.939734	0.946895

Python for Data Science

So, let us see how to get that. So, we are going to use a crosstab function to arrive at a conditional probability. So, I have initially set normalize is equal to index, but here I am changing it to column just to get the column sums as 1. But all other parameters remains the same.

Now let us try interpret from the output. So, given though FuelType of the car is CNG what is a probability of the car having a manual gearbox? It is 1. So, in this case there is 0 because we know that there is no Automatic gearbox which are of CNG or diesel FuelType. But the probability of getting a car given that the FuelType is petrol and the car has also manual gearbox is 0.93.

So, now we have seen how to get the conditional probability by considering the two variables because, we have also set the normalized parameter as columns and we have also seen how to set normalize is equal to index. And we have also seen how to interpret those results.

(Refer Slide Time: 20:53)

The slide has a blue header with the word 'Correlation' in white. In the top right corner, there is a logo for 'GITAA' with a small icon. Below the title, there is a bulleted list:

- Correlation: the strength of association between two variables
- Visual representation of correlation: Scatter plots

Below the list, there are three scatter plots illustrating different trends:

- Positive trend: Points show a general upward slope.
- Negative trend: Points show a general downward slope.
- Little or no correlation: Points are scattered randomly without a clear pattern.

At the bottom left of the slide, it says 'Python for Data Science'. On the right side, there is a thumbnail of a woman with long dark hair, wearing a teal top, sitting at a desk and speaking. The background of the slide is white.

Next we are going to look at correlation because, till now we have been looking at to check the relationship between two categorical variables using cross tabulation. Now, we are going to look at how to check the relationship between two numerical variables used the measure called correlation. And what is correlation? Correlation is just to check the strength of association between two numerical variables and it need not be always numerical variables, but in this case or in this lecture we are going to look at the correlation for numerical variables.

We are going to look at a visual representation of correlation using scatter plot. So, if you see here the first plot says positive trend because, as one variable increases the other variable is also increasing. For example, as your weight of the car increases the price might go up, in that case you call it is as a positive trend when you see the points are scattered. In this case and there is another plot that you can see here is little or no correlation.

Because there is no pattern that you can find out from this scatter plot here rather all the points have been scattered all over in that case you can say that there is no correlation between those two variables at all. Theoretically or numerically if you want to interpret from the correlation measure then the correlation values bounded between -1 and +1. And closer to one in either ns is the represent the higher correlation it can be either a negative or positive sign.

Because, the correlation can be there can be high correlation negatively and there can be high positively. If you want to interpret from the correlation measured in terms of numbers then the correlation value will be bounded between - 1 to + 1. 0 represents there is no correlation at all between any two numerical variables. And closer to 1 represents there is a strong correlation between two variables positively. Theoretically, above 0.7 we can say there is a fair correlation between two numerical variables.

If you can take it to the other side of it 0 to - 1 then closer to - 1 will give you high negative correlation like this whenever the age of the car increases the price will always decrease. Because for the newer aged car the price will always be really high, in that case there can be a strong negative relationship between those two variables so the value will be closer to - 1. So, now we have got an idea about what correlation measures is about and how we can interpret visually and how we can interpret numerically.

(Refer Slide Time: 23:42)

The screenshot shows a Jupyter Notebook interface with a slide header 'Correlation' and a GITAA logo. The code cell contains:
`DataFrame.corr(self, method='pearson')`
A callout box highlights:

- To compute pairwise correlation of columns excluding NA/null values
- Excluding the categorical variables to find the Pearson's correlation

The next cell shows:
`numerical_data = cars_data2.select_dtypes(exclude=[object])`
A callout box highlights:
Let's check the no. of variables available under `numerical_data`
In [28]:
`print(numerical_data.shape)`
`(1436, 8)`
A video player window is visible on the right, showing a woman speaking. The video title is 'Python for Data Science'.

So, now we are going to see how to get the correlation using python, corr is the function that is used to calculate correlation between any variables that you can use that for a data frame. Because, by using the corr function we are going to compute the pairwise correlation of columns by excluding all the null values here. Because, we are not just going to consider only two variables rather we are going to consider all the variables at a time and the function computes the pairwise correlation. We will see what pairwise correlation is when we get the output.

But this function is used to get the pairwise correlation of columns and by default it excludes all the missing values from the data frame and then it calculates the correlation value. And the method I have specified here is Pearson's because by default it calculates the Pearson correlation. And Pearson correlation is also used for to check the strength of association between two numerical variables.

If you have ordinal variables then you can go for other measures as Kendall rank correlation and Spearman rank correlation. So, in that case we need to exclude the categorical variables to find the Pearson correlation. Now let us see how to exclude those variables which are of categorical data type. So, here cars_data two is a data frame that I am working on from the data frame I am going to select only the columns which are of numerical data type.

Since I am just going to exclude only categorical variables I have given object under exclude. I have saved that to a new data frame called numerical data. So, let us see what would be the dimension of it by checking what is the number of variables that are available under numerical data; if you see if you print and see the shape of it you can look you can see that there are 1436 observation with eight variables. Initially we had 10, we have we are left out with only 8 variables now which are of numerical data type.

(Refer Slide Time: 25:45)

```

Correlation
DataFrame.corr(self, method='pearson')
• Correlation between numerical variables
corr_matrix = numerical_data.corr()

[[{"Index": "Price", "Price": 1, "Age": -0.878407, "KM": -0.57472, "HP": 0.309902, "MetColor": 0.112041, "Automatic": 0.03380807, "CC": 0.165067, "Weight": 0.581198}, {"Index": "Age", "Price": -0.878407, "Age": 1, "KM": 0.512735, "HP": -0.157904, "MetColor": -0.099659, "Automatic": 0.0325732, "CC": -0.120706, "Weight": -0.464299}, {"Index": "KM", "Price": -0.57472, "Age": 0.512735, "KM": 1, "HP": -0.335285, "MetColor": -0.0938252, "Automatic": -0.0812477, "CC": 0.299993, "Weight": -0.0262711}, {"Index": "HP", "Price": 0.309902, "Age": -0.157904, "KM": -0.335285, "HP": 1, "MetColor": 0.0647485, "Automatic": 0.013755, "CC": 0.0537575, "Weight": 0.01373}, {"Index": "MetColor", "Price": 0.112041, "Age": -0.099659, "KM": -0.0938252, "HP": 0.0647485, "MetColor": 1, "Automatic": -0.0139728, "CC": 0.0291886, "Weight": 0.0116}, {"Index": "Automatic", "Price": 0.03380807, "Age": 0.0325732, "KM": -0.0812477, "HP": 0.013755, "MetColor": -0.0139728, "Automatic": 1, "CC": -0.0693213, "Weight": 0.085}, {"Index": "CC", "Price": 0.165067, "Age": -0.120706, "KM": 0.299993, "HP": 0.0537575, "MetColor": 0.0291886, "Automatic": -0.0693213, "CC": 1, "Weight": 0.0572485}, {"Index": "Weight", "Price": 0.581198, "Age": -0.464299, "KM": -0.0262711, "HP": 0.0867373, "MetColor": 0.0571416, "Automatic": 0.0572485, "CC": 0.0572485, "Weight": 1}]]
```

So now, let us see how to create the correlation matrix. So, we are going to look at the correlation between the numerical variables using the command corr. And the data frame

that I am applying it to is the numerical data and I am saving this input to an output called corr matrix.

Now, we are going to calculate the correlation from the data frame that we have now using the function.corr here the interest is to find the correlation between the numerical variables. And the data frame that we are applying it here is numerical data because that is where we have all the columns as numbers.

And we are also saving the output to an object called corr_matrix so that will have the output of the correlation matrix. So, now, let us look at the output and try to interpret the correlation values. So, this is a snippet that is taken from the variable explorer. And if you see here the index represents the variable names the variables in separate columns.

So, if you look at the principal diagonal which are marked in purple all are of one. Because the correlation between price and price would be 1; because the relationship is being checked against that we rebuilt itself that is why you are getting the value as 1 but if you see the value - 0.87 that represents that, there is a negative correlation between price and age. Since there is a negative symbol over there and the correlation is above 0.5; that means, that the correlation there is a strong negative correlation between age and price.

Whenever the age of the car increases, the price is decreasing that applies same to the kilometre though the correlation value is slightly lesser than kilometre it has 0.5, which is 0.57 which is equivalent to 0.6. It also have a fair negative correlation between kilometre and price whenever the car has travelled a lot the price will automatically go down. For a newer car and for the cars which have travelled really less in that case the price is always really high.

And if you look at 0.58 as the weight of the car increases the price is also increasing that is why there is a positive correlation value here that is 0.58. You can also look at the relationship between kilometre and age. There is a fair relationship positive relationship between kilometre and age because the values is 0.5 as the age of the car increases the kilometre is already increased. Similarly you can interpret from different values of different variables and here we have used the Pearson correlation. If you want to look out for other correlation measures you can specify that under the method argument.

(Refer Slide Time: 28:35)

Summary

- Frequency tables
- Two-way tables
- Two-way table - joint probability
- Two-way table - marginal probability
- Two-way table - conditional probability
- Correlation

Python for Data Science

So, now we have come to end of the session. So, let summarize whatever we have seen it in this lecture. We have started with creating frequency tables to check what is the frequency of each categories in the categorical variable. And then we were interested in looking at the relationship between two categorical variables using a two-way tables. And then we have also seen how to convert the two-way table into joint probabilities, marginal probabilities and conditional probability. And we have also seen how to check the relationship between two numerical variables using a measure called correlation.

Thank you.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 19
Data Visualization Part –I

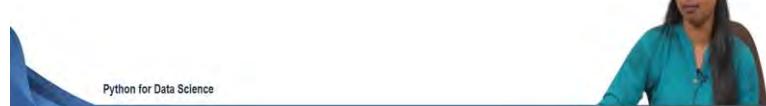
(Refer Slide Time: 00:21)

In this lecture



We will learn how to create basic plots using *matplotlib* library

- Scatter plot
- Histogram
- Bar plot



Hello all welcome to the lecture on the module Data Visualization. So, in this lecture, we are going to do data visualization, where we are going to learn how to create basic plots using the *matplotlib* library. The basic plots include scatter plot, histogram and bar plot.

(Refer Slide Time: 00:33)

Data Visualization



- Data visualization allows us to quickly interpret the data and adjust different variables to see their effect
- Technology is increasingly making it easier for us to do so

Why visualize data?

- Observe the patterns
- Identify extreme values that could be anomalies
- Easy interpretation



So, now let us see what data visualization is about data visualization, allows us to quickly interpret the data and adjust different variables to see their effect. What do we mean by quickly interpreting the data, because if we were to understand the data using numbers, then it would be difficult for us to understand the data and quickly interpret the data. But if we were to understand through graphical representation of data, it would make us; it would make the work easier in terms of interpretation and in terms of understanding.

If you see here the technology is also increasingly making it easier for us to do so because there are so many visualization techniques tools that are available to do visualization to understand the data even more better than interpreting from the numbers. But why do we need to visualize the data? We can observe the patterns. You can also identify the extreme values that could be anomalies. And if you want to interpret the data easily, then you have to go for visualization. So, these are the important points to visualize the data.

(Refer Slide Time: 01:37)

Popular plotting libraries in Python



Python offers multiple graphing libraries that offers diverse features

- | | |
|--|--|
| <ul style="list-style-type: none">• matplotlib• pandas visualization• seaborn• ggplot• plotly | <ul style="list-style-type: none">• to create 2D graphs and plots• easy to use interface, built on Matplotlib• provides a high-level interface for drawing attractive and informative statistical graphics• based on R's ggplot2, uses Grammar of Graphics• can create interactive plots |
|--|--|

Python for Data Science



So, python offers multiple graphing libraries that offers diverse features, we are going to look at few graphing libraries that are available in python. And what is the need, first one is matplotlib which is used to create 2D graphs and plots. And I have mark it as green because we are going to look at matplotlib library in this lecture. The next one is pandas visualization which has easy to use interface and that was built on top of matplotlib.

The next one is seaborn it provides a high level interface for drawing interactive and informative statistical graphics and that was also built on top of matplotlib, whatever graph and plots that you are creating using seaborn library can also be created using matplotlib library. And you can also have a control on what insights you can get of your plot using seaborn. The next one is ggplot; ggplot is used whenever you want to do advanced graphics, and this ggplot is entirely based on R's ggplot, R's another programming language that is used for analytics. And it has a package called ggplot2 which is entirely dedicated for doing visualization.

And in python we have ggplot that is based on R's ggplot2, and it basically uses grammars of graphics the next is plotly. If you want to create interactive plots, then you can go for plotly. But in this course we are going to look at two graphing libraries that are matplotlib and seaborn.

(Refer Slide Time: 03:11)

Matplotlib

- Matplotlib is a 2D plotting library which produces good quality figures
- Although it has its origins in emulating the MATLAB graphics commands, it is independent of MATLAB
- It makes heavy use of NumPy and other extension code to provide good performance even for large arrays

Python for Data Science

First we are going to look at a matplot library. So, let us see what mat plot library is about. Matplot library is a 2D plotting library which produces good quality figures. Although it has its origin in emulating the MATLAB graphic commands it is independent of MATLAB. It makes heavy use of NumPy another extension code to provide good performance even for large arrays. So, this is the overview of the matplotlib library.

(Refer Slide Time: 03:39)

The image shows a video frame of a woman with long dark hair, wearing a teal top, speaking. Above her, a white rectangular box contains the text "Scatter plot" in blue. The background is a blue gradient with some white shapes. In the top right corner, there is a logo for "GITAA Transforming careers" featuring a stylized book icon.

(Refer Slide Time: 03:42)

The image shows a video frame of a woman with long dark hair, wearing a teal top, speaking. Above her, a white rectangular box contains the text "Scatter Plot" in blue. Below the title, there are two sections with bullet points: "What is a scatter plot?" and "When to use scatter plots?". The background is a blue gradient with some white shapes. In the top right corner, there is a logo for "GITAA Transforming careers" featuring a stylized book icon. At the bottom left, there is a small text "Python for Data Science".

So, under the matplotlib library we are going to look at a plot called scatter plot. So, let us see what is a scatter plot first. A scatter plot is basically a set of points that represents the values that obtained for two different variables plotted on a horizontal and vertical axis. So, if you look at a scatter plot, you will have to access to it x and y. And the points from the data frame will be scattered all over the place of the plot or in a separate pattern. And using that and using the variables that you have given as an input, you will get the relationship out of it.

And there can be a question when to use those scatter plots. So, basically the scatter plots are used to convey the relationship between any two numerical variables, how one variable varies with respect to the other variable. And scatter plots are also sometimes called as correlation plots, because they show how two variables are correlated, whether they are negatively correlated or positively correlated, all this information you can get that by looking at the patterns of your scatter plots.

(Refer Slide Time: 04:44)

The slide has a blue header bar with the title "Importing data into Spyder". In the top right corner, there is a logo for GITAA with the tagline "Transforming careers". The main content area contains the following text and code snippets:

- Importing necessary libraries

```
import pandas as pd ← 'pandas' library to work with dataframes
```

```
import numpy as np ← 'numpy' library to do numerical operations
```

```
import matplotlib.pyplot as plt ↓ 'matplotlib' library to do visualization
```

At the bottom left, it says "Python for Data Science". On the right side, there is a video player interface showing a woman speaking, with controls for play/pause, volume, and progress.

So, now we are going to import the data into Spyder to do a scatter plot. Prior to importing the data we have to import the necessary libraries. The first library is pandas and we have imported it with the as pd and we have to import the pandas library to work with data frames. And then we will also be doing some numerical operations on it. So, we need to import numpy as np. And then since we are going to use the matplotlib library, and we have to import pyplot from the matplotlib as plt. And we use this to do visualization.

(Refer Slide Time: 05:21)

The screenshot shows a Python code editor with the following content:

```
Importing data into Spyder
• Importing data
cars_data = pd.read_csv('Toyota.csv',index_col=0,
na_values=['??','????'])
Variable explorer
Name Type Size
cars_data DataFrame (1436, 10)

• Removing missing values from the dataframe
cars_data.dropna(axis = 0, inplace=True)
Variable explorer
Name Type Size
cars_data DataFrame (1096, 10)
Python for Data Science
```

A video player window is overlaid on the bottom right, showing a woman speaking.

Now, we can import this data into Spyder using the `read_csv` command. And we have said the first column as the index column, and we have considered all the question marks as default nan values, so that we can do any operations that are related to nan values. So, `cars_data` is your data frame now and the size of it is 1436 with 10 columns. We are going to remove all the missing values from the data frame, because we are going to visualize the data. In that case I do not want to consider all the missing values rather I just want to consider all the rows where there are no missing values.

So, in that case, if you want to drop out all the records where there are missing values, then you can use the command called `dropna` preceded with the data framing. And inside the function if you set `axis` is equal to 0, then you consider all the rows where there are missing values and remove that. And I have also given `inplace` is equal to true that is because I have not assigned this to a new data frame as `cars_data` to or are in the existing data frame rather I have just used `inplace` is equal to true my setting `inplace` is equal to true, the modifications that you are making here will be reflected in the data.

If you do not give this, the records will not be dropped from your data frame. If you look at the size of your data frame after removing the missing values, it turned out to be having 1096 observations with 10 columns. So, we have left out around 400 observations.

(Refer Slide Time: 06:57)

```
Scatter plot
```

```
x           y
plt.scatter(cars_data['Age'], cars_data['Price'], c='red')
plt.title('Scatter plot of Price vs Age of the cars')
plt.xlabel('Age (months)')
plt.ylabel('Price (Euros)')
plt.show()
```

Python for Data Science

So, now let us see how to create a scatter plot. So, for creating a scatter plot, scatter is the function since we are doing it from the matplotlib library. We have saved pyplot as plt, so that is why I have used the allies called plt to access the function scatter. And inside the function I have given the x coordinate variable that is age and I have also given the y coordinate variable that is price. So, it is very evident that we are going to check the relationship between age and price through scatter plot.

And the next thing is I have given red under the parameter c that means that I am going to color the markers using the red color. So, we will see what markers when we get the output. This line will produce a scatter plot. But if you want to add title to your plot and if you want to add labels to your x-axis and y-axis, you can also give that. See if you want to add title to your plot, you can give it as plt.title. And inside that I have just given it a scatter plot of prices age of the cars.

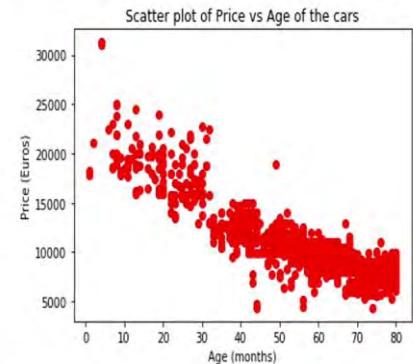
And the x label represents the age which are in months, and the y-axis represents the price which are Euros. Now, we have created a scatter plot to show the scatter plot you have to use plt.show. Even if you select the lines to plt.y label, you will be able to show the plot. Since this is a standardized code, whenever you are building your plots layer by layer by giving the type of plot first and then giving titles after that, then you have to run everything in one shot to plt.show to get all the information in a single plot.

(Refer Slide Time: 08:43)

Scatter plot



- The price of the car decreases as age of the car increases



Python for Data Science



So, now let us see what is the output by using the scatter function. Here this is the scatter plot to show the relationship between price of the car and the age of the car. So, if you see from the output, your x-axis represents the age and y-axis represents the price; if you see here, as the age increases the price decreases. The points that are shown here are called as markers. And the vertical line that is shown here are ticks, and the 60, 70, 80 are called tick labels. Since these are an x-axis, these are called xticks and x-labels, and these lines are called yticks and these values are called ytick labels.

(Refer Slide Time: 09:37)

Histogram



What is a histogram?

- It is a graphical representation of data using bars of different heights
- Histogram groups numbers into ranges and the height of each bar depicts the frequency of each range or bin

When to use histograms?

- To represent the frequency distribution of numerical variables

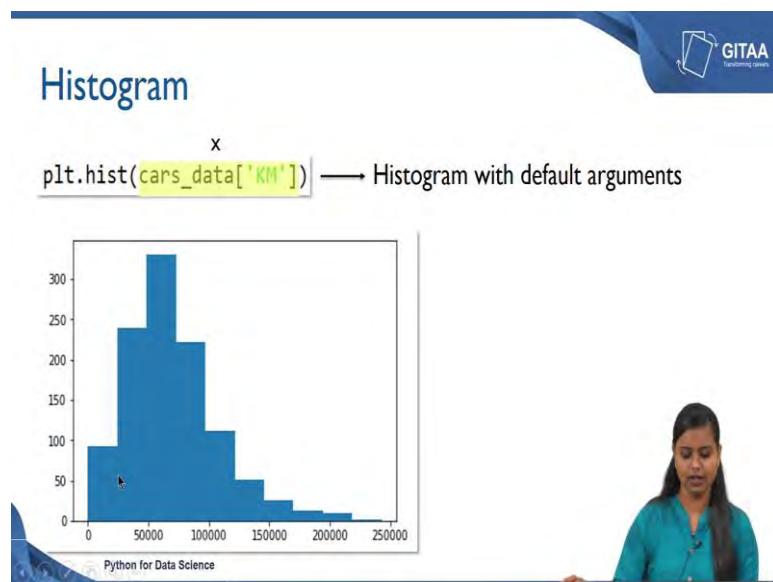
Python for Data Science



So, now we are going to see how to create the histogram. Before creating we need to know what histogram is. So, what is a histogram? It is a graphical representation of data using bars of different heights. So, whenever you have a numerical variable, and if you want to check the frequency distribution of the variables, you can go for histogram, so that each bar will give you different heights that represents the frequencies.

And histograms group numbers in two ranges, because on the x-axis you will have intervals or ranges that can also be called as bins. And for each of the bins, you will get a corresponding frequency by looking at the height of each bars. So, when to use the histogram? As I mentioned to represent the frequency distribution of any numerical variables you can look for histogram.

(Refer Slide Time: 10:27)



Now, we will see how to create a histogram using matplotlib. So, the command is hist that stands for histogram, and that is from the library called pyplot library called matplotlib, and pyplot is the sub library that is available under mat plot and plt is just an alias to the pyplot. And inside the function, I was just given kilometer from the cars_data, data frame. So, that becomes the x-axis. So, we are creating a histogram with the default arguments, because we have just given the x variable alone.

And let us look at the output. So, here is the histogram with default arguments on the x-axis you have the bins. The first bin is from 0 to 50,000, and the second bin is from 50,001 to 100,000 lakh, and the third bin is from 100,001 to 150,000. For example, using

this histogram I am not able to get the exact bin range or the exact range the corresponding frequency, because it does not give me the separation between each bars.

(Refer Slide Time: 11:36)

```
plt.hist(cars_data['KM'],
         color     = 'green',
         edgecolor = 'white',
         bins      = 5)
plt.title('Histogram of Kilometer')
plt.xlabel('Kilometer')
plt.ylabel('Frequency')
plt.show()
```

Python for Data Science

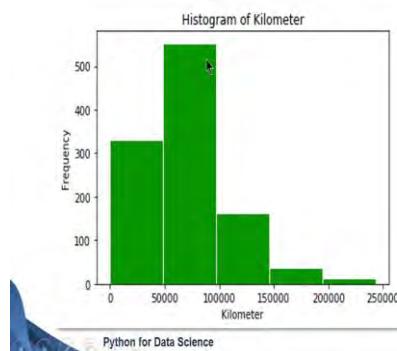
So, now, what I can do here is I can create a histogram by specifying the edge color and the number of bins I have. If I have too many bins, then it would not be easier for us to understand the frequencies for each interval, in that case you can specify or fix the number of bins as well. So, here we are using the same command, where we have fixed the x-coordinate that is kilo meter.

Since we are looking at the frequency distribution of the kilometer of the car and the color for the bars will be green, and the edge color of the bars I am giving it as white, so that there is a clear demarcation between two bars, and I have fixed the number of bins as 5. And you can also add title xlabel and the ylabel to your plot here the title for the plot is histogram of kilo meter and the xlabel is kilometer and the ylabel is frequency. And if you give plt.show and if you run from plt.hist to plt.show, you will get a histogram with those specified number of bins.

(Refer Slide Time: 12:42)

Histogram

- Frequency distribution of kilometre of the cars shows that most of the cars have travelled between 50000 – 100000 km and there are only few cars with more distance travelled



So, here is the output that we get from the previous code. So, here we are looking at the frequency distribution of kilometer of the cars. And it shows that most of the cars have traveled between 50000 to 100000 kilometer and there are only few cars with more distance travelled. Now, if you see that there is a clear demarcation between each of the bins, and you will be able to look at the frequencies of it easily.

(Refer Slide Time: 13:12)

Bar plot

What is a bar plot?

- A bar plot is a plot that presents categorical data with rectangular bars with lengths proportional to the counts that they represent

When to use bar plot?

- To represent the frequency distribution of categorical variables
- A bar diagram makes it easy to compare sets of data between different groups



Next we are going to look at a bar plot. Let us see what is a bar plot. Bar plot is a plot that represents categorical data with rectangular bars. Whenever you have a categorical data,

and if you want to look at the frequencies of each categories in a variable, then we will look at in terms of a bar plot. Bar plot is also similar to histogram, there would not be any space in between the bars for this histogram, because you are looking at in terms of continuous range.

But here you will be looking at in terms of frequencies of categories, so you will have space in between the bars that is what the difference between the bar and the histogram. And you use bar plot for a categorical variable and we use histogram for a numerical variable. And those length of the bar is basically depicts the frequencies of each categories.

And when to use a bar plot to represent the frequency distribution of any categorical variables, and the bar diagram makes it easier to compare sets of data between different groups. If you have a free categorical variable and if you have so many categories that under that variable if you want to look at distribution of each variable or how each category is distributed, then bar diagram is easier to interpret more on the categories of a variable.

(Refer Slide Time: 14:26)

The slide shows a Python script for creating a bar plot. The code is as follows:

```
counts = [979, 120, 12]
fuelType = ('Petrol', 'Diesel', 'CNG')
index = np.arange(len(fuelType))

plt.bar(index, counts, color=['red', 'blue', 'cyan'])
plt.title('Bar plot of fuel types')
plt.xlabel('Fuel Types')
plt.ylabel('Frequency')
plt.xticks(index, fuelType, rotation = 90)
plt.show()
```

Annotations explain the code:

- A double-headed vertical arrow between 'index' and 'counts' is labeled "x height of the bars".
- An arrow from 'counts' points to the first argument of plt.bar, labeled "Set the labels of the xticks".
- An arrow from 'index' points to the second argument of plt.bar, labeled "Set the location of the xticks".

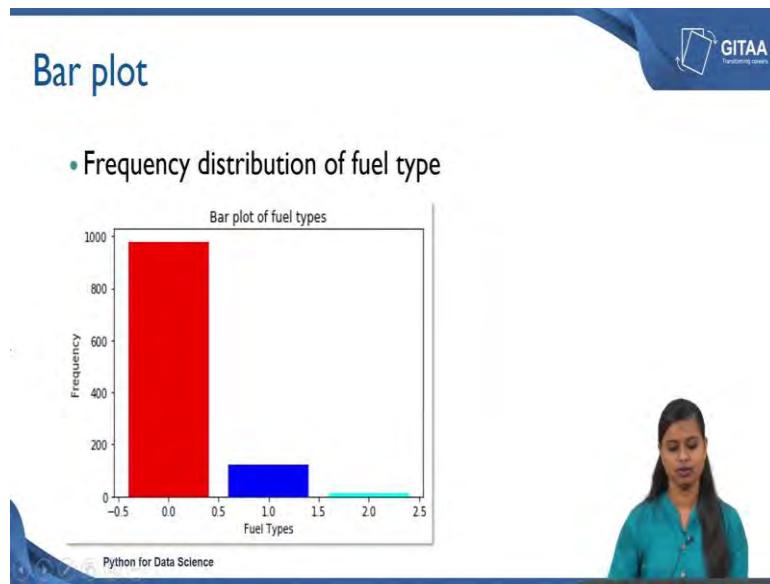
A woman in a teal shirt is visible on the right side of the slide.

Now, let us see how to create a bar plot, basically you have to have an array which shows the counts of each categories. And the next object or the next variable represents a fuelType here we have petrol diesel and CNG as fuelTypes of the cars. And since we know that apriori have just given otherwise you can also give value_counts as an input,

because that will have the category as well as the frequencies. And I have given index because on the x-axis, it should know the index the index is whatever is the length of fuelType what is the length of the fuelType it would be 3. So, the range would be 0, 1 and 2. And to create a bar plot.bar is the function that is from the pyplot sub library; and inside the function, you just basically need to give what should be there in the x-axis and what should be there in the y-axis.

And x-axis I want the index to be present, and on the y-axis I want the counts to be present. And I am going to color my bars differently red will go for petrol and diesel will be colored based on blue color and CNG will be colored based on cyan color. And I have also given the title and the labels for x and y-axis, so that I can create a bar plot which will show the frequencies of the fuelType categories. And as I have mentioned earlier index is nothing but x coordinate and count will represent the height of the bar, at height of the bars have been represented using counts.

(Refer Slide Time: 16:10)



So, this is the output that we got. But if you see on the x-axis, you have only the range. Here we are not able to find out which is petrol which is diesel or which is CNG, in that case you can also specify the labels to your x-axis by giving xticks. So, all the other codes reminds the same, but since I just want to add labels to each of the bars on the x-axis that can be given using xticks you can set the labels of the xticks using fuelType, because you have already assigned the variable called fuelType with some string values,

so that will be labels for the xticks. You can also set the location for x ticks where petrol should be present and diesel should be present and CNG should be present. And I have given rotation is equal to 90, so that all the labels will be 90 degree rotation.

So, now let us see how the plot will look like. Here if you see this the first bar is corresponding to the petrol fuelType and it is very evident that most of the cars around are 900 and odd cars have the fuelType as petrol, and there are only few cars for which the fuelType is about diesel and CNG.

(Refer Slide Time: 17:32)

Summary



We have learnt how to create basic plots using *matplotlib* library

- Scatter plot
- Histogram
- Bar plot

Python for Data Science

Now, we have come to the end of the session. In this lecture, we have learned how to create basic plots using *matplotlib* library. Those basic plots were scatter plot, histogram and bar plot.

Thank you.

Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 20
Data visualization Part – II

Welcome to the lecture on Data Visualization.

(Refer Slide Time: 00:17)

In the previous lecture



We learnt how to create basic plots using *matplotlib* library

- Scatter plot
- Histogram
- Bar plot



In the previous lecture we have been looking about how to create basic plots using *matplotlib* library. The basic plots include scatter plot, histogram and bar plot. In that lecture we have seen how to create these plots and we have also seen what *matplotlib* library is about and how to interpret each of these plots.

(Refer Slide Time: 00:39)

In this lecture



We will learn how to create basic plots using seaborn library:

- Scatter plot
- Histogram
- Bar plot
- Box and whiskers plot
- Pairwise plots

Python for Data Science



And in this lecture we are going to see how to create basic plots using seaborn library and the basic plots includes scatter plot, histogram, bar plot, box and whiskers plot and pair wise plots. We will see in detail about the creation of these plots and how to interpret them.

(Refer Slide Time: 00:57)

Seaborn



- Seaborn is a Python data visualization library based on matplotlib
- It provides a high-level interface for drawing attractive and informative statistical graphics

Python for Data Science



So, first let us see what seaborn library is about; seaborn library is a Python data visualization library which was built on top of matplotlib library that provides a high

level interface for drawing attractive and informative statistical graphs. So, using seaborn library we are going to first see how to create this scatter plot.

(Refer Slide Time: 01:17)

The slide has a blue header with the GITAA logo. The main title is 'Importing libraries'. Below it, a bullet point says 'Importing necessary libraries'. Four code snippets are shown with arrows pointing to their descriptions:

- `import pandas as pd` → 'pandas' library to work with dataframes
- `import numpy as np` → 'numpy' library to do numerical operations
- `import matplotlib.pyplot as plt` → 'matplotlib' library to do visualization
- `import seaborn as sns` → 'seaborn' library to do visualization

At the bottom left, there's a watermark 'Python for Data Science'.



So, before creating the scatter plots we need to import the basic libraries that are needed for visualization. So, let us import the necessary libraries. We are importing the pandas with alias pd because to do the visualization we need to have a data frame, here we are going to create plots using a data frame. Since we are going to deal with data frames we are importing the pandas library, if you if we want to if we wanted to do some numerical operations on it then we need a numpy library.

So, let us import the numpy as np as well, then we can import the libraries that are needed for visualization. Here even though we are going to create plots using the seaborn library we are also importing the pyplot from mat plot library with the alias plt because as we know that seaborn library has been built on top of the matplotlib library. We will be using some of the functionalities that are from the matplotlib library in that case we would be needing that library.

So, let us import matplotlib as well followed by that we can import the seaborn library as sns. Here sns will be realized for the seaborn library and matplotlib and seaborn will be used for data visualization.

(Refer Slide Time: 02:33)

The screenshot shows a Python code editor with two code snippets and their corresponding Variable Explorer outputs.

Code Snippet 1:

```
cars_data = pd.read_csv('Toyota.csv', index_col=0,
                        na_values=['??', '????'])
```

Variable Explorer Output 1:

Name	Type	Size
cars_data	DataFrame	(1436, 10)

Code Snippet 2:

```
cars_data.dropna(axis = 0, inplace=True)
```

Variable Explorer Output 2:

Name	Type	Size
cars_data	DataFrame	(1096, 10)

A woman in a yellow shirt is visible on the right side of the slide, likely the instructor.

So, now we have imported the necessary libraries into spyder. Now, its time to import the data that will be used for the visualization because, as we know we do visualization to understand the data even more better. So, on that note we are going to import a data into spyder to understand the data visually.

So, we are going to import the Toyota data which we have already worked on since it is of csv format I have used pd.read_csv and while reading itself I have considered all the missing values as the default nan values and I have ceded to an object calls cars_data, now cars_data is a DataFrame. When we looked at the size of the DataFrame through the variable explorer you can see that it has 1436 observations with 10 columns and now we are very familiar with what Toyota data is about.

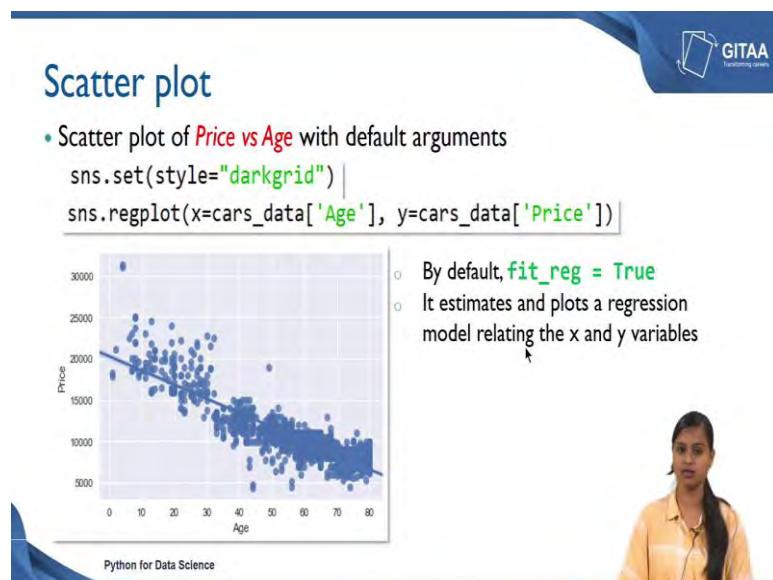
They are just the details of the cars. So, since our objective is to understand the data pictorially or visually, we need to consider only the rows wherever the records are filled because we would not be able to interpret anything from the question marks or from the nan values. In this case our objective is to basically understand the data and understand the relationship that exists between each of the variables. In that case we wanted to remove missing values, so that we will focus on understanding the data which have complete informations from the DataFrame.

So, on that note we can just remove all the missing values from the DataFrame and then proceed with the visualization. So, let us remove the missing values from the DataFrame

by using the command dropna and inside the function you can just say axis is equal to 0. So, that all the rows will be removed wherever there are nan values and if you use inplace is equal to true you are making all the modifications in the data frame that is being used here. Otherwise you have to reassign it to a new data frame called cars_data2 or you can update it in the existing data frame itself.

So, here I have updated while I am using the function itself by using in place is equal to True and after dropping all the missing values if you look at the size of the data frame it is turned out to be 1096 observations with 10 columns on the whole we have removed around 340 observations from the DataFrame and we are going to proceed with the analysis only with the 1096 observations with 10 columns.

(Refer Slide Time: 05:09)



So, now let us see how to create the scatter plot. So, here we are going to create a scatter plot of Price versus Age and that will be built using the default arguments. So, if you see the first line I have given as sns.set; that means, that we are giving a theme to the background of the plot, the theme was being dark grid; that means, that you will have a dark shade as a background of your plot and you will as well have a grid.

So, you can basically set themes to your background and dark grid is a preset seaborn team you can as well set themes that are available from the seaborn package and you can set that under the style argument and even by default if you use sns.set it will give you a dark grid theme. So, in order to do a scatter plot the function to be used from the seaborn

library is regplot; regplot stands for regression plot and inside the function you have to specify the variables for x axis and the y axis.

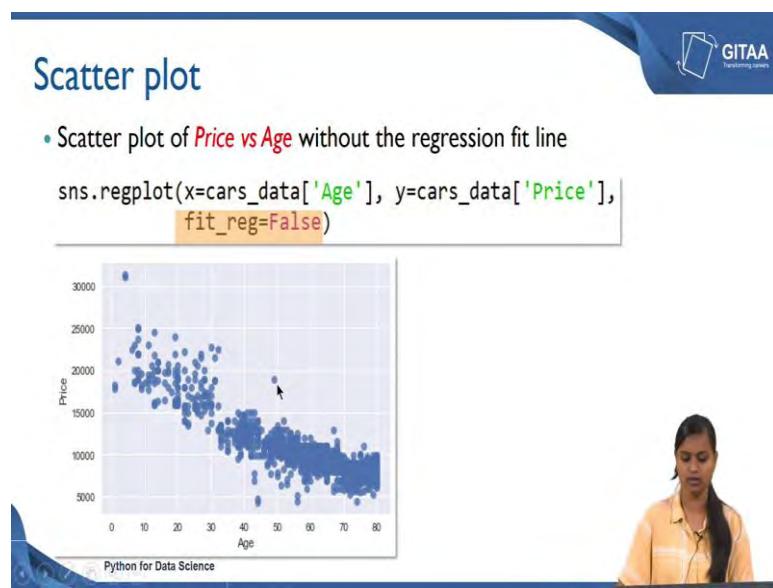
Since it is going to be the scatter plot of Price versus Age we need to give Age under the x axis and Price under the y axis we are going to check how Price varies with respect to the Age of the cars. So, here if you see I have accessed the Age variable from the data frame cars_data similarly I have accessed the price variable from the cars_data.

So, this would be the output that you will be getting by setting the style as dark grid and by giving x as Age and y as Price. And if you see here as the Age increases the Price decreases and Age that we see here is not in terms of years, but it is in terms of months. So, 70 represents the cars Ages around 5.8 years and 80 represents the cars Ages around 6.6 years.

So, in that case most of the cars or the Price is really low whenever the cars Age is beyond 5 years and the Price is really high for the new Aged car because the car has travelled a lot there could be more damage to it and since it is not new the Price is also coming down. So, it is very obvious from the logic perspective as well as from the data perspective as Age increases the Price automatically decreases. And if you also see there is a regressions line which is plotted over the scatter points that is because by default fit_reg is equal to drew in regplot function.

What it does mean it basically estimates the coefficient of x and then plots a regression line over the points that is why the function is named as regression plot and if you do not want to consider this regression fit line into your scatter plot if you just want to scatter plot of any two variables you can also do that.

(Refer Slide Time: 08:05)



Like here we are going to do a scatter plot of price versus age without the regression fit line that is very easy you can retain the same command as the previous slide, but you just need to add another parameter called `fit_reg` and you can just set them to false.

Because as I mentioned earlier by default it is true that is where you got a regression line over those scattered points, but if you said they are as false you would not be getting the regression line rather you will just get a scatter plot of Price versus Age. So, and here if you see the plot Age is called as the label to the x axis and Price is called as the label to the y axis.

Here we have not specified any labels to x and y axis it is already being shown here that is how because it takes the label from the variable name that we have already given here. And the values 0, 10, 20, 30 are the values from the variable Age and from the plotting context these are called x tick labels and 5,000, 10,000 and so on are called as y tick labels that is on the y axis the boxes are called as grids and the points the scatter points are called as markers. Here we have a solid round points.

(Refer Slide Time: 09:21)

Scatter plot

- Scatter plot of **Price** vs **Age** by customizing the appearance of markers

```
sns.regplot(x=cars_data['Age'], y=cars_data['Price'],  
            marker='*', fit_reg=False)
```

Price

Age

Python for Data Science

GITAA
Transforming careers

A woman in a yellow shirt is speaking.

So, now we are going to see how to create a scatter plot of Price versus Age by customizing the appearance of the markers. Appearance of the markers in the sense you can change the shape of your markers to differentiate between the categories, but here I have just taken only the scatter plot of Price versus Age. So, I am going to change all the markers into a different shape.

So, in that case I can do that using the argument called marker and inside the marker argument you can specify the type of marker you wanted to have it in your scatter plot. Here have since I have given the asterisks all the scatter points will be marked as stars this will be very useful whenever you want to differentiate the points with respect to different categories that are available under a variable. Other than that all the codes remains the same I have just added marker is equal to star that should be given under the single or double code.

(Refer Slide Time: 10:21)

Scatter plot

- Scatter plot of **Price vs Age** by **FuelType**
- Using **hue** parameter, including another variable to show the fuel types categories with different colors

```
sns.lmplot(x='Age', y='Price', data=cars_data,
            fit_reg=False, hue='FuelType',
            legend=True, palette="Set1")
```

So, in this example we are going to create a scatter plot of Price versus Age by FuelType; by FuelType in the sense we are going to add one more variable into the scatter plot that is the variable FuelType. Why are we going to add that? Because it is of interest to check the relationship between Price with Age by adding another variable FuelType, like how Price varies when the Age increases with respect to different FuelTypes of the car. So, that can be added by using the parameter called hue and including another variable to show the fuel type categories with different colors.

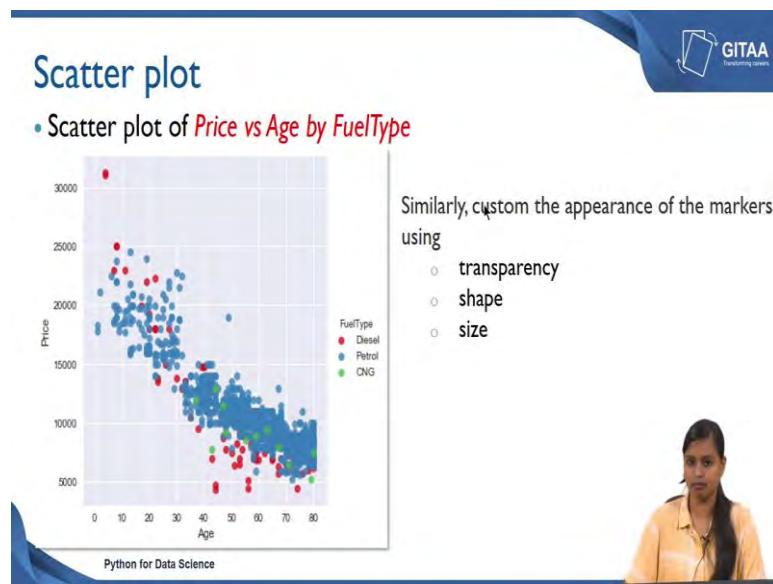
Here we are going to add fuel type under the argument or parameter called hue and we are going to represent the fuel type with respect to different colors. So, here we are going to do that by using the command called lmplot that is from the seaborn library lmplot is a function which combines regression plot and face. It is very useful whenever you want to plot a scatter plot by doing conditional subsets of data or by including another variable into the picture in that case a lmplot will be very useful.

And here the x axis is Age variable and the y axis is the Price variable and the data from which we are going to axes the cars_data and I have also said False different fit_reg. So, that we do not get a regression fit over the scatter points and here I am adding one more variable into the scatter plot using the hue parameter and since we are going to add one more variable points are going to be differentiated using colors based on the fuel

type of the car in that case we need to know which colour represents what category in that case I need to have legends as well.

So, I have set legends as True and you can also set color palettes using the different color palettes that are available in the seaborn library; one of it is called set1. So, I have just used set1 as the color palette to color the data points based on the FuelType. So, this is what the input is about.

(Refer Slide Time: 12:23)



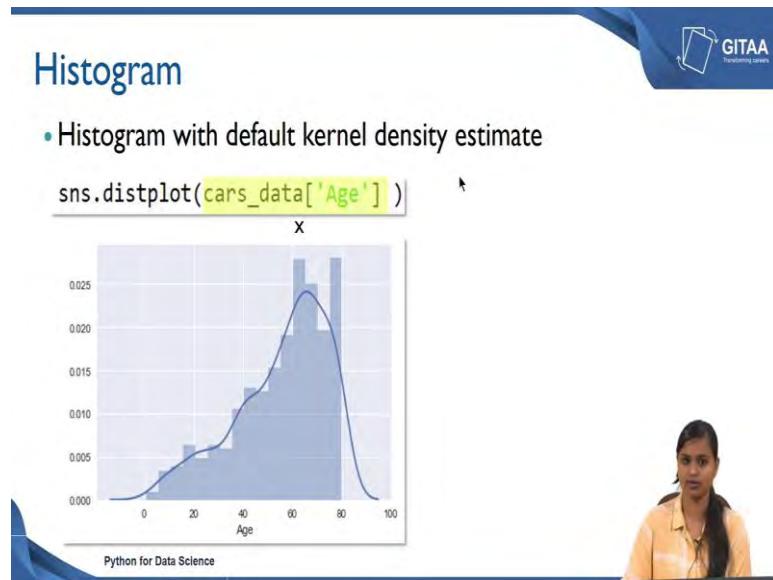
Let us see what is the output that we are getting. So, this is a scatter plot of Price versus Age by FuelType, it is the same scatter plot, but here we have just differentiated the data points using the categories that are available under the FuelType. But we have just differentiated the data points based on the FuelType categories. So, the red represents diesel and the blue represents petrol and the greens are CNG.

So, if you see here there are more data points or there are more cars that are off petrol fuel type and the price is really higher for the cars which has diesel fuel type and the price is comparatively lower for the cars which have CNG fuel type. And you can also check the relationship between price and age as we know that whenever the cars age is high the price is very low whenever the cars age is low the price will be high.

Similarly you can also custom the appearance of the markers using transparency, shape and size. Like how we differentiated the points using colors here different levels of

transparency to the points to represent the categories you can also do the same thing by giving different shapes and by giving different sizes for data points. So, this will help you include more variables into a single plot or you will be able to understand the data quickly rather by looking at the numbers.

(Refer Slide Time: 13:51)



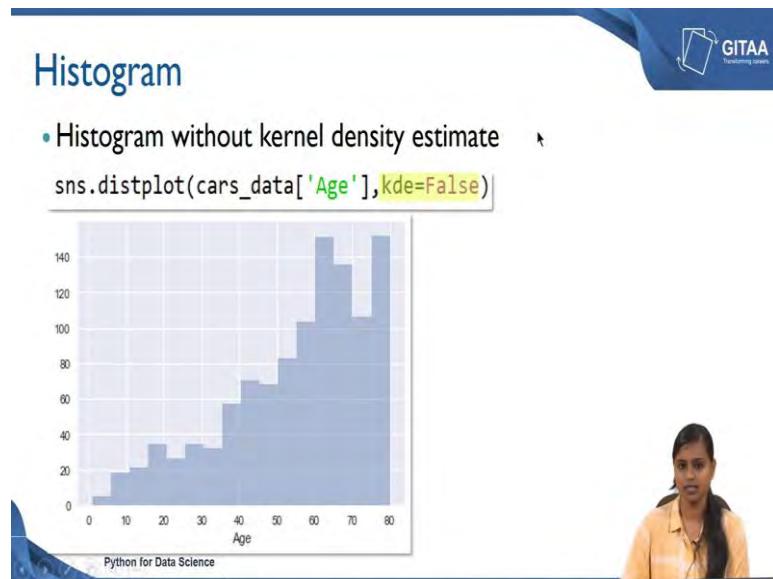
So, next we are going to look at a plot called histogram. So, we are going to plot a histogram with default kernel density estimate because using the seaborn library by default it gives you a histogram with the kernel density estimate. So, the command that needs to be used this dist plot that is the representative of distribution plot.

Since that is from the seaborn library of used sns.distplot an input should be any numerical variable for any continuous variable because histogram can be plotted to check the frequency distribution of any continuous variable. In that case Age is one of the continuous variables from the data frame, here we are interested in looking at the frequency distribution of Age how the values are distributed under the Age variable in that case you can use or you can axis the Age variable from the data frame cars_data.

So, this is being the representative of the x axis and as you know by default it gives you the kernel density estimate. It gives you work curve over the bars which is the representation of the distribution of the variable Age and on the y axis you have the kernel density estimate, but if your interest is to get the frequency distribution in terms of

counts or numbers, then you can also get rid of the kernel density estimate and then have the counts on your y axis.

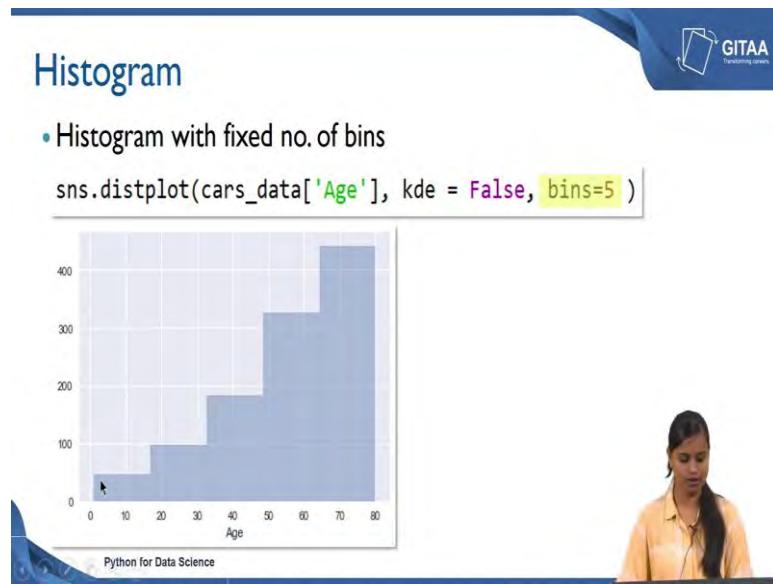
(Refer Slide Time: 15:17)



Let us see how to get that. So, here we are going to plot histogram without the kernel density estimate you can do that by setting kde is equal to False. So, if you give that the output would be similar to this on the y axis it is just a representation of frequencies or the counts on the x axis you have the range of values from the Age variable. And you have too many intervals here it is very difficult to interpret from this plot the range is about 75 to 80, where the cars age is more than 6 years the count is really high. And there are very few cars which are new that is what we can interpret because there are two extremes here; one is really high and one is really low.

So, in that case you can also have a control on how many number of bins you want to have on the plot. So, that by having the control on the bins you will be able to quickly interpret from the distribution of your age variable, let us see how to fix the number of bins.

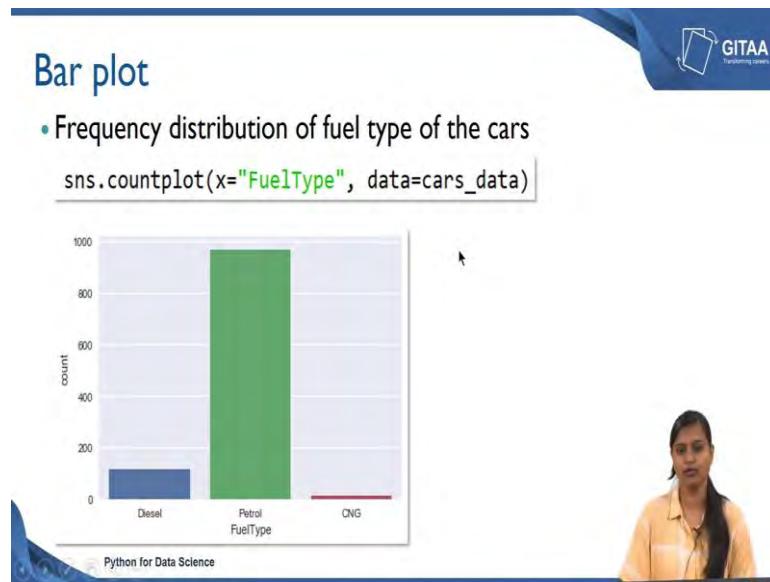
(Refer Slide Time: 16:13)



So, here we are going to plot the histogram with fixed number of bins. So, retain the same codes as in a previous slide you can just add bins is equal to 5. So, that you will have only 5 bins in your histogram.

Bins are nothing but the range or the intervals. So, from this plot by having the number of bins as 5. It is very clear that the frequency is really high for the bin value 70 to 80; that means, that wherever the cars age is above 5 years, the frequency is high. Because most of the cars detail is about the cars whose age is about 70 to 80 months, but you have only very few cars which are very new in our data frame that is why the frequency is very low in the range between 0 to 50 months.

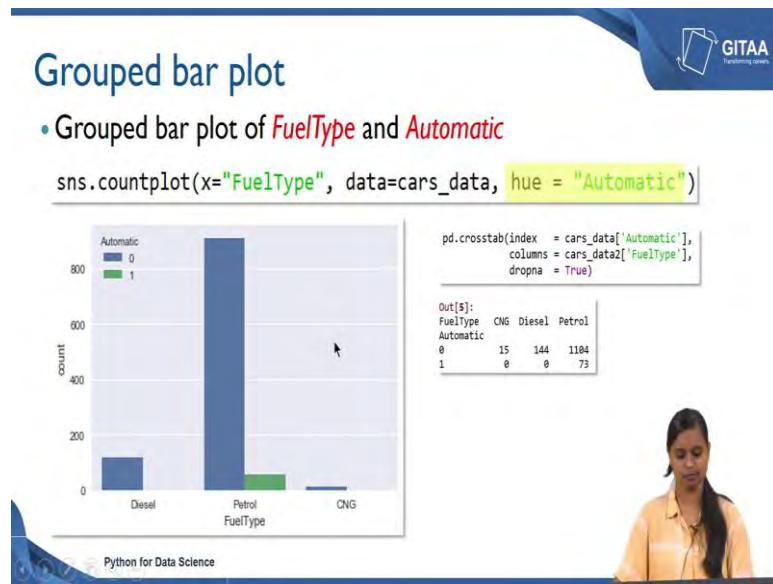
(Refer Slide Time: 17:05)



So, next we are going to look at the bar plot. So, bar plot is basically to check the frequency distribution of any categorical variable, here FuelType is one of the categorical variable in our data frame. So, let us look at the frequency distribution of the fuel type of the cars. So, to create a bar plot countplot is the function that needs to be used under the function you just need to give the variable of interest that is FuelType and you can also specify from which data frame that you are accessing it from, it is cars_data.

So, it is straightforward you will get the output, you are getting the fuel type on your x axis at the count on your y axis and the bars are the representative for the counts with respect to each FuelType. So, it is very evident from the plot that most of the cars have the FuelType as petrol and there are very few cars whose FuelType are diesel or CNG. So, now, we have seen how to create the bar plot.

(Refer Slide Time: 18:01)



So, here we are going to see how to create group bar plot. We are going to create the group bar plot of FuelType and Automatic, we mean by group bar plot. So, here we are going to have the bar plot of FuelType by adding another variable as Automatic. So, we are going to look at the frequency distribution of the FuelType of the car along with the interpretation whether the cars gearbox is of Automatic or manual.

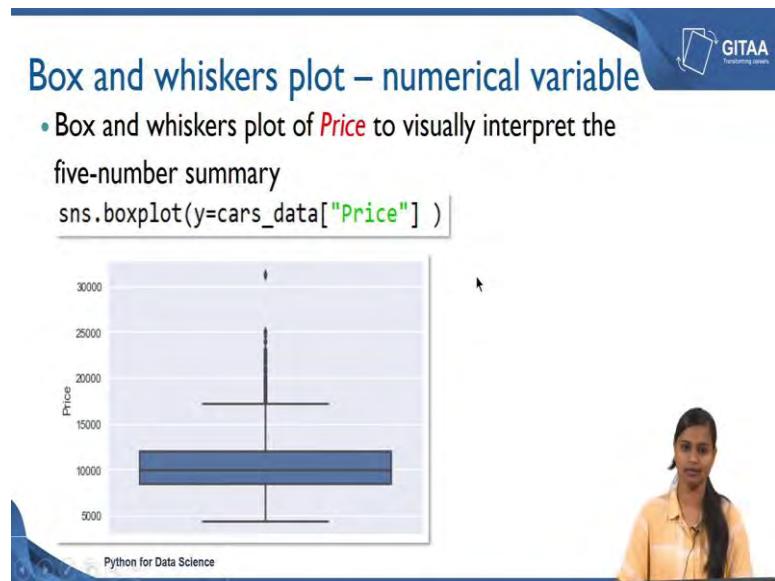
So, let us see how to do that we can use the same function as bar plot that is count plot and you can specify the FuelType under the argument x and you can also specify the data frame. So, that it knows from which it should axis the FuelType from and by adding hue is equal to Automatic we are including one more variable into the bar plot. So, that the we are checking the relationship between the FuelType and Automatic.

So, let us see what would be the output. So, here is the output on the x axis you have the FuelType you have different fuel types like diesel, petrol and CNG and on the y axis you have the count. If you see the legend here it has two value 0s and 1s and the 0s are the representative of manual gearbox and the 1s are the representative of automatic gearbox.

So, if you see here there are only automatic gearbox types whose fuel type is of petrol and there are no cars available with automatic gearbox when the fuel type is about when the fuel types are diesel or CNG. If you recall this is just the visual representation of the crosstab output that we have seen earlier in the lectures. So, we have used the crosstab function to arrive at the relationship between the Automatic on the FuelType where we

have seen that in terms of numbers and we have tried to interpret that, but here using visualization we are going to interpret the same thing, but visually.

(Refer Slide Time: 19:53)



So, next we are going to look at box and whiskers plot we are going to look at is box and whisker plot for numerical variable for if you have a numerical variable in your data frame and if you want to interpret more from the variable, then you can go for box plot. Because the box plot will give you the visual representation of the. The five number summary includes mean, median; the five number summary includes minimum maximum and the three quantiles the those are first quantiles, second quantile and third quantile.

So, let us see what the quantiles are and what the minimum and maximum values are. So, how do we do that? We can create a box plot using the command box plot that is from the seaborn library and since we are interested in getting five number summary of the Price variable I have accessed the Price variable from cars_data and I have given that under the argument called y. So, that the y will be the representation of Price, but if you give it under x the x will be the representation of Price.

So, now, let us see how the output will look like. So, as I mentioned since we have given Price under the argument y we are getting the Price value on the x axis. So, now, let us try to interpret the box plot it is called as box and whiskers plot because it has some box as well as the whiskers to it the horizontal lines are called as the whiskers, now let us try

it interpret this. This horizontal line is called as the lower whisker or also the representation of the minimum value which is excluding the outliers.

If you have outliers in your data which are nothing, but the extreme values then it excludes that values and then gives you the picture of what is the minimum value of the Price. The minimum value of the Price of the car is about 5000 Euros and the upper whiskers which represents the maximum value that is excluding the outliers.

So, the maximum value of the cars is around 16 to 17 thousand by excluding the outlier or the extreme values and these are called low whisker and the upper whisker. The lowest horizontal line of the box represents the first quartile that is 25 percentage of the cars Price is less than 8000 and the middle line represents the second quartile which is nothing, but the median. That means, that 50 percentage of the data is less than 10,000 Euros. And the upper horizontal line is nothing, but third quartile that is q_3 which represents that 75 percentage of the Price of the car is less than around 12,000 Euros this is called q_1 , q_2 and q_3 .

And whatever the points that are lying above the whiskers are called as outliers and you can also have the values below the whiskers as well. If you have points about the upper whisker then you can say that these are outliers or the extreme values where the values are really more than the 1.5 times of your q_3 . This is what the q_3 values about.

The q_3 is around 12000, so these values are 1.5 times of q_3 that is why it is called as extreme values, it is not in between the range of the price that is why it is called as the extreme values. Similarly you can ask what happens when you have some values above the lower whiskers in that case those are also called as outliers or extreme values because those values will be less than the 1.5 times of q_1 .

So, using box plot you will be able to visually look at the distribution of the Price variable and what would be the median value and what are the quantile values of the Price. And using the box plot the main key point to the box plot is you will be easily able to identify the outliers of any numerical variable. So, this is how we create a box plot and this is how we interpret the box plot as well.

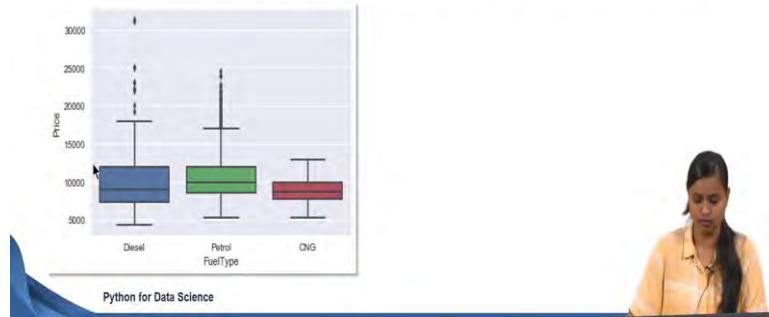
(Refer Slide Time: 23:57)

Box and whiskers plot



- Box and whiskers plot for numerical vs categorical variable
- Price of the cars for various fuel types

```
sns.boxplot(x = cars_data['FuelType'], y = cars_data["Price"])
```



So, now we are going to look at the box and whisker plot for numerical with this categorical variable because box and whisker plot is very useful when you want to check the relationship between one numerical and one categorical variable, like how the price varies with respect to the another variable. Because in the previous example we have just looked at the distribution and how do we detect the outliers using the box plot.

Here we are going to check the relationship between two variables in that case you can use box plot; you can use box plot when you have one numerical and one categorical variable here we are going to look at the price of the cars for varies fuel types of the cars. So, you can use the same command as in the previous slide that is box plot and under the x argument I have given FuelType and another y argument I have given the variable as Price.

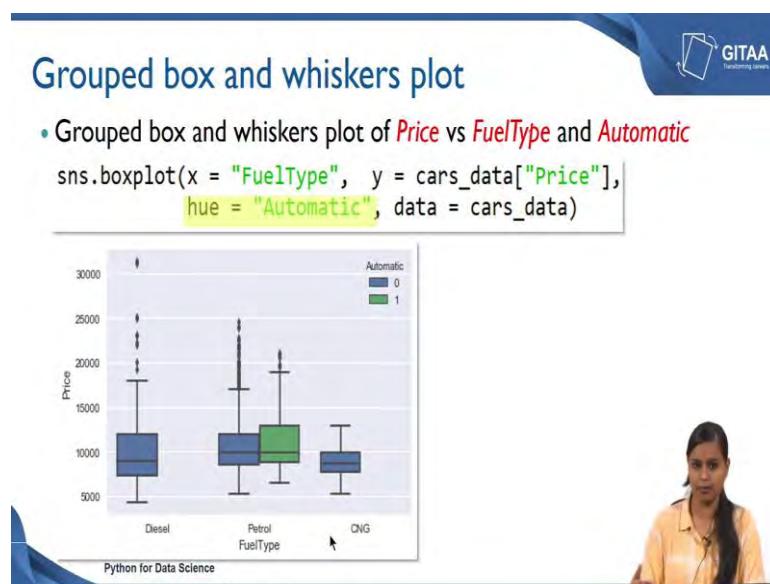
So, you have FuelType on your x axis and you have Price on your y axis and by looking and the plot it is very evident that the Price varies with respect to different FuelTypes of the cars. And as we know that the middle line is the representation of the median if you look at the middle lines of the different FuelTypes of the cars, the median Price of the car is really high when the FuelType of the car is petrol.

And the median value is really low when the FuelType is of either diesel or CNG. It is very evident that the on an average the petrol FuelType has the highest Price among the cars from the data set that we have and you can also see the maximum value of the

maximum price of the car is for the diesel FuelType and the minimum value of the car is also for the diesel FuelType.

So, these are the interpretation that you can make here and you can see there are some extreme values that are above the upper whiskers of the diesel and petrol FuelType and those are called as the extreme values. Because very few observations are there and those are about the 1.5 times of your third quantile that is 12,500 Euros. So, now, we have checked the relationship between FuelType and the Price.

(Refer Slide Time: 26:05)

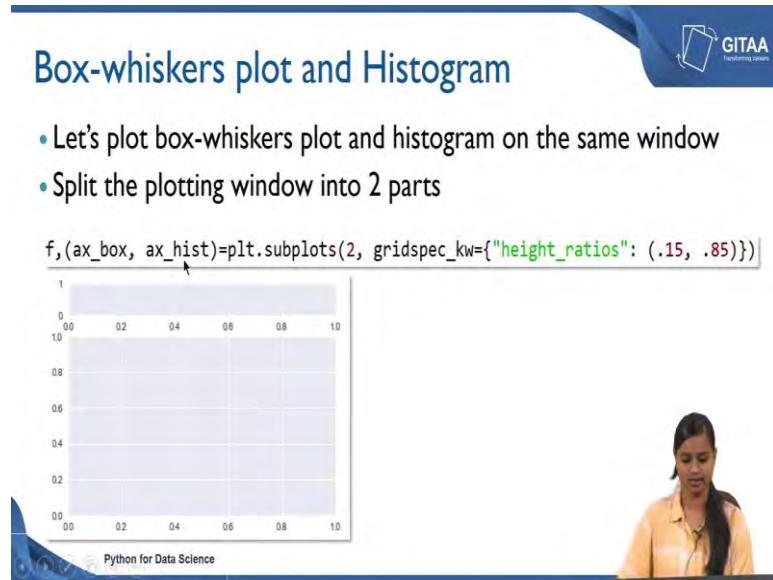


So, now we are going to look at the group box and whiskers plot. Here we are going to look at the group box and whiskers plot of Prices versus FuelType by including one more variable called Automatic. So, let us see how to do that. The codes remains the same whenever you want to have a grouped box plot you just need to add that variable including the argument called hue and I have added Automatic under hue. So, we will have a group box and whiskers plot, let us see what the output is about.

So, here you have the same box plot as the previous slide, but you have another boxes which is corresponding to the different FuelTypes and you have another boxes which are the representation of the Automatic variable. Here if you see the Automatic variable is being included here and the 0s are the representative of manual gearbox and the 1s are the representative of Automatic gearbox and those are represented by blue and green in color respectively.

So, here whenever the cars fuel type is petrol and the gearbox type is also Automatic there are no cars that are available for the Automatic gearbox when the FuelType is of either diesel or CNG, that is very evident from any of the plots that we have seen in the previous slides as well.

(Refer Slide Time: 27:25)



Till now we have been looking at having only one plot in a window, we can also have multiple plots in a window, here we are going to look at the box and whisker plot and the histogram on the single window. So, let us plot box whisker plot and histogram on the same window. So, in order to do that we have to first split the windows into two parts. What do we mean by splitting the window is into two parts? This is what we mean.

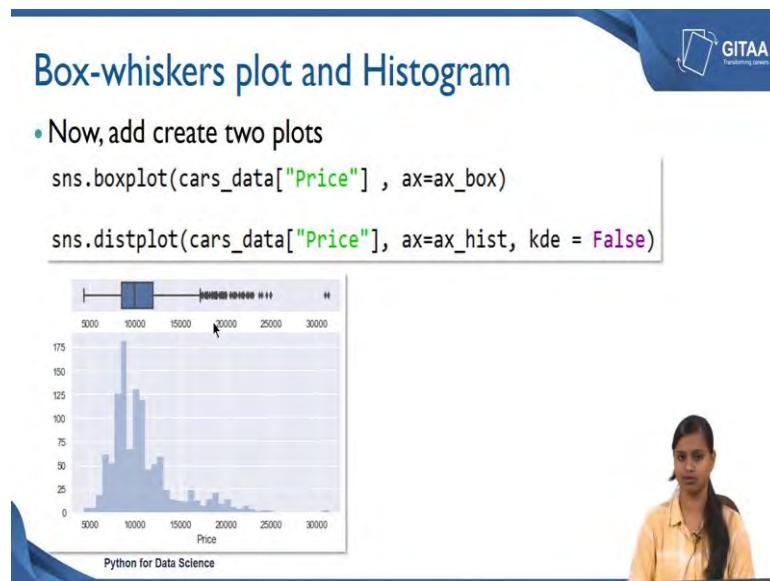
So, this is a single window I have just splitted the window into 2. So, that I can plot two plots in the same window and it will be easier for me to interpret from the two plots instead of creating different plots for the same variable I can as well do that in the same window itself. So, here if you see there is only small space for the upper part and there is a large space for the lower part that is how because I have said the aspect ratio with just because I have said the aspect ratio accordingly. Let us see how to split the window with giving; let us see how to split the window plotting window and we will also see how to customize the aspect ratio.

So, subplots is a function that is used to create the subplots and by giving 2 you mean that you need to have 2 rows. So, you want to split your window into 2 parts in row wise

and by giving specification of grids using grid spec_kw you can set the ratios of your height I have given 0.15 and 0.85. So, that there is a larger space for the second window and there is a less space for the first window.

I have just splitted my window into 2 and given the aspect ratio and I have saved that output to a and I have saved that output to two variables called f and axis box plot and axis for histogram because these are going to be same for both the box plot and the histogram. Combinely have saved it to two objects and I have also saved it to a figure. So, ax_box represents the axis for box plot and ax_hist represents the axis for the histogram.

(Refer Slide Time: 29:33)



So, let us now we can add or create two plots. So, the codes remains the same because for creating the box plot you have to use a box plot and the variable of interest is Price and using the axis ax_box. And we have also created the histogram using the dist plot function, where we have specified the variable as price and were we have also specified what is the axis for that that is using the same axis that is ax_hist. And we wanted just the histogram with not the kernel density estimate; so that is why we have set kb is equal to false.

So, let us see how the output will look like. So, this is the output for the first command using the boxplot you are getting the boxplot for the variable price as well as you are getting the histogram for the variable Price. So, here this is the histogram of Price as well

as the boxplot. So, you will be able to look at the frequency distribution of any continuous variable as well as the five number as well as you will be able to look at the visual representation of five number summary of it.

If you want to look how the outliers are present in a variable and what is the minimum and maximum value of a variable. So, all these information can be get using a single plot when you create plots using subplots.

(Refer Slide Time: 30:57)

Pairwise plots



- It is used to plot pairwise relationships in a dataset
 - Creates scatterplots for joint relationships and histograms for univariate distributions
- ```
sns.pairplot(cars_data, kind="scatter", hue="FuelType")
plt.show()
```



So, now we will see how to create pairwise plots pairwise plots are used to plot pairwise relationships in the data and basically creates the scatter plots for joint relationships and histograms for univariate relationships you will understand what do we mean by joint relationships and univariate relationship when we look at the output of it.

So, to create the pairwise plot the command that is used this pair plot that is also from the seaborn library and you can also specify the data frame because on the whole we are going to look at the relationship between all the variables considering all the variables. And the kind of plot that we are going to plot for any numerical variable is scatter and we are going to colour the data points using another variable called FuelType.

(Refer Slide Time: 31:41)



So, now let us look at the output of the plot. So, this would be the output of your pairwise plot you basically have the all possible relationship using all the variables. So, the first plot is like the distribution of the price because on the x axis you have price and on the y axis also you have the price variable. So, this is called as joint relationship whenever you have same variable on both x and y axis, then you call it as joint relationship in that case you will just have the histogram of it like to indicate the frequency distribution of the variable price.

But here the next plot if you see the y axis is price and if you look at the x axis it is age. So, this is actually; so when you move from left to right whatever you have in your x axis represent is the representative of the x axis and whatever you have it in your y axis is the y axis variable. So, this is the scatter plot of price versus age, it is very evident that as the age increases the price decreases and all the colours are the representative of various fuel types of the cars.

So, similarly you will be able to interpret it for the other variables relationship and if you see all the diagonal boxes are histograms because all the diagonals values represents all the diagonal boxes represent the univariate distribution because this is just the distribution of the variable age and this is just the distribution of the variable kilometer and the distribution of the variable is hp is being represented here.

So, the pairwise plot is used to get the plots for all possible combination of variables and you will be able to look at in terms of scatter plot and histogram. This would give you an easier way to understand the relationship that exists between different pairs of variables. You can also see there is a legend which is shown here for the fuel type like how the markers have been colored.

(Refer Slide Time: 33:47)

The screenshot shows a presentation slide with a blue header bar. On the left, the word 'Summary' is written in white. On the right, there is a logo for 'GITAA Transforming careers' featuring a stylized book icon. Below the header, the text reads: 'We have learnt how to create basic plots using seaborn library:' followed by a bulleted list of plot types. The list includes: Scatter plot, Histogram, Bar plot (with a bullet point for Grouped bar plot), Box and whiskers plot (with a bullet point for Grouped box and whiskers plot), and Pairwise plots. At the bottom left of the slide, it says 'Python for Data Science'. On the right side of the slide, there is a small video frame showing a person speaking.

- Scatter plot
- Histogram
- Bar plot
  - Grouped bar plot
- Box and whiskers plot
  - Grouped box and whiskers plot
- Pairwise plots

So, now we have come to the end of the session, let us summarize whatever we have learned till here. So, we have learned how to create the basic plots using the seaborn library; the first thing that we have seen is how to create scatter plot and then we have seen how to create histogram using the seaborn library. And we have also looked at how to create simple bar plot and then we have seen how to create grouped bar plot.

Followed by that we have seen how to create box and whiskers plot and also, we have seen how to create group box and whiskers plot. At last we have seen how to create the pairwise plot from the seaborn library which was very useful to look at the relationship of all the variables in a single window.

Thank you.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 21**  
**Dealing with missing values**

Welcome to the lecture on Dealing with missing values.

(Refer Slide Time: 00:19)

**In this lecture**



- Identifying missing values
- Approaches to fill the missing values



In the previous lecture we have analyzed the data with complete cases. The complete cases in the sense, we have not considered the missing values in the analysis; wherever they were missing values we have omitted those missing values and then we will proceed with the analysis. That would not be the real case scenario, because in a real time scenario you would be having lot of missing values which you will be dealing with, in that case they need to be your way or an approach on how to deal with those missing values.

So, in this lecture we are going to see how to identify the missing values and some of the approaches to fill in those missing values. So, these are the major concepts that we are going to see in this lecture.

(Refer Slide Time: 01:01)

The screenshot shows a Python code editor with two lines of code:

```
import os
import pandas as pd
```

Annotations explain the imports:

- `import os` → 'os' library to change the working directory
- `import pandas as pd` → 'pandas' library to work with dataframes

A video player interface is visible at the bottom, showing a thumbnail of a person speaking and the text "Python for Data Science".

So, prior to importing your data into Spyder, we need to import some of the necessary libraries that are required for the analysis. So, we are going to import the os library that has to change the working directory. So, that you will be able to access the file, then we are going to import the pandas library as pd and this is to work with data frames, so that we can use any functionalities that are from the pandas library.

Next we can now change your working directory by using the chdir command, inside the function you just need to give the file path of it. So, I given the file path; once you set your working directory you will be able to access the file in your current working directory.

(Refer Slide Time: 01:42)

## Importing data into Spyder



- Importing data

```
cars_data = pd.read_csv('Toyota.csv',index_col=0,
na_values=['??','????'])
```

- Creating copies of original data

```
cars_data2 = cars_data.copy()
cars_data3 = cars_data2.copy()
```

Python for Data Science



Next let us import the data into Spyder, I am importing the Toyota.csv which we were walking on, and I have said the first column as my index column and I have also given some set of string values that needs to be considered as python NaN values; and I have save my output on object called cars\_data. Now, cars\_data becomes the data frame, whatever I am going to do that should not be reflected in the original data that I have now; rather I can just create multiple copies of data, so that I can work with the duplicated data.

So, let us create copies of original data. So, using the.copy command, so I am creating the copy from the original data cars\_data; and saving that as a new data frame called cars\_data2. So, this is one copy and I have need also another copy called cars\_data3 from the cars\_data2 data frames. So, I have now two copies of data from the original one; that is cars\_data2, and cars\_data3. Now, we are going to see how to identify the missing values.

(Refer Slide Time: 02:48)

## Identifying missing values



- In Pandas dataframes, missing data is represented by **NaN** (an acronym for Not a Number)
- To check null values in Pandas dataframes, **isnull()** and **isna()** are used
- These functions returns a dataframe of Boolean values which are True for NaN values

Python for Data Science



In Pandas data frames, missing data is represented by NaN. And NaN is just an acronym for not a number. So, whenever you have a blank cell in your csv file, the python will automatically read it as NaN value. So, all those NaN values will be treated as missing values in Pandas data frames. And if you want to check that null values in Pandas data frames, there are several functions that are available to do any operations that are related with dealing with missing values. We are going to consider only the two functions that are used to check the null values in Pandas data frames, two of it has been shown here.

The first one is `isnull` and the second one is `isna`; both of the functions will account for the python default NaN values. These functions can be used to check the null value city of Pandas data frames. Because before proceeding with how to deal with your missing values, you need to identify whether there are any missing values or not or how many of them are there. In that case these functions would be really helpful to do that; these functions basically returns a data frame of Boolean values which are true for NaN values. And, these functions returns data frame of Boolean values which are true for NaN values.

Basically, you will get an output with only true or false values. So, wherever you have NaN values it will you will see it as true, and wherever you do not have NaN values rather you have other values then you will have the value as false. We will see now how to use these two functions.

(Refer Slide Time: 04:20)

**Identifying missing values**

**Dataframe.isna.sum(), Dataframe.isnull.sum()**

- To check the count of missing values present in each column

```
cars_data2.isna().sum() (or) cars_data2.isnull().sum()
```

Out[38]:

| Price     | 0   |
|-----------|-----|
| Age       | 100 |
| KM        | 15  |
| FuelType  | 100 |
| HP        | 6   |
| MetColor  | 150 |
| Automatic | 0   |
| CC        | 0   |
| Doors     | 0   |
| Weight    | 0   |

dtype: int64

Python for Data Science

So, as I have mentioned these isna and isnull are the functions that are used to check the missing values in the Pandas data frames. So, we are going to get the count of missing values present in each column. Let us see how to do that, because I do not want to look at how many missing values are there in a complete data frame rather I will just want to look at how many number of rows are missing under each column.

So, that it will give me an idea on how to proceed with the missing value, so that it will give me an idea how to deal with that missing values. So, the syntax is Dataframe.isna.sum, in order to get the count of missing values present in each column. And you can also use isnull function with the same syntax. So, I have shown the code here; that is cars data2isna.sum. We will see what the dot sum is used for. Similarly I have used it for isnull, though both of the functions give you the same output and gives you the same information you can choose any one of it.

So, now let us look at the output. So, the output shown is, the output shown here is for the function isna. And you are getting the column names correspondingly you are also getting how many number of rows are missing under that particular column. For example, age has 100 rows where there are missing values, kilometer has 15 rows where there are missing values; similarly FuelType has 100 rows with missing values, horsepower has only 6 rows with missing values, and the MetColor has 150 rows with

missing values none of the other variables have missing values. So, now we have got an idea about how many number of rows are missing under each of the columns.

Now, we got an idea about how many number of rows are missing under each columns. Now it is not enough to know that, because we need to check whether 100 rows missing under age is completely different from the 100 rows that are missing under FuelType, or there are any sequence, or there are any combination of missing values that are available in our data frame. In a same row, you want to see whether only one column is missing or multiple column values are missing; because whenever you have missing values you need to have two types of approach.

One is to completely get rid of all the rows wherever there are missing values. And the other one is, to have an approach to logically fill up those missing values. So, these two cases depends upon the problem that you have. So, if you have a row when all the column values are missing or most of the column values are missing; then there is no point in refilling the missing values with any substitute value right, you can just get rid of that particular row alone.

In that case, removing the rows will help you and even if there are; and the second thing is if there is no pattern to your missing values like if it is just randomly missing all over the cells, then you should go and look at an approach to fill in those missing values. In that case we this numbers will not give us an idea about that, we need the subset all the rows and see where the missing values are there and then we can decide on what we have to do with the missing values.

(Refer Slide Time: 07:37)

The screenshot shows a Jupyter Notebook interface. At the top, there is a header with the text "Identifying missing values" and the GITAA logo. Below the header, a bullet point says "Subsetting the rows that have one or more missing values". Underneath this, a code snippet is shown:

```
missing = cars_data2[cars_data2.isnull().any(axis=1)]
```

To the right of the code, there is a "Variable explorer" window. The table in the explorer shows the following data:

| Name       | Type      | Size       |
|------------|-----------|------------|
| cars_data  | DataFrame | (1436, 10) |
| cars_data2 | DataFrame | (1436, 10) |
| missing    | DataFrame | (340, 10)  |

A small video player window in the bottom right corner shows a woman speaking.

So, now we are going to subset the rows that have one or more missing values, because I need to consider a row wherever there are only one column is missing and as well as more than one column is missing. In that case, the code shown here will do that, from the cars data2, I am subsetting the rows wherever there are missing values using the function isnull and I have given.any of axis = 1; the 1 represents column.

I am telling the function give me all the rows wherever at least one column value is missing. And we have save that to an object call missing, so it becomes a data frame. Let see how the missing values are or now let us see where the missing values are there. So, once you read that, you will get that in your variable explore. Let us check the dimension of it, it is a data frame now; and the dimension of it is 340 rows with 10 columns. So, there are 340 rows, where at least one column value is missing. So, now let us see where the missing values are there.

(Refer Slide Time: 08:43)

The screenshot shows a Jupyter Notebook cell with the title "Identifying missing values". The cell contains a Pandas DataFrame with 1436 rows and 11 columns. The columns are: Indx, Price, Age, KM, FuelType, HP, MfrColor, Automatic, CC, Dmnl, and Weight. The "Age" column is highlighted in pink, indicating missing values (NaN). The DataFrame looks like this:

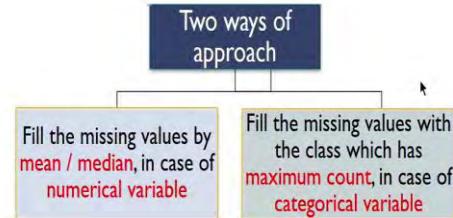
| Indx | Price | Age   | KM     | FuelType | HP  | MfrColor | Automatic | CC   | Dmnl | Weight |
|------|-------|-------|--------|----------|-----|----------|-----------|------|------|--------|
| 247  | 12900 | nan   | 55678  | Petrol   | 110 | 1        | 0         | 1600 | 4    | 1030   |
| 896  | 8250  | nan   | 60000  | Petrol   | 86  | 0        | 1         | 1300 | 4    | 1030   |
| 581  | 10500 | 31579 | 35230  | Petrol   | 97  | 0        | 0         | 1400 | 3    | 1025   |
| 572  | 10950 | nan   | 63451  | Petrol   | 97  | 0        | 0         | 1400 | 3    | 1025   |
| 230  | 11925 | nan   | 104885 | Petrol   | 97  | 0        | 0         | 1400 | 3    | 1025   |
| 404  | 9450  | nan   | 104885 | Petrol   | 97  | 1        | 0         | 1400 | 3    | 1025   |
| 1431 | 7500  | nan   | 20544  | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1025   |
| 586  | 9550  | nan   | 29650  | Petrol   | 86  | nan      | 0         | 1300 | 3    | 1025   |
| 1433 | 8500  | nan   | 17015  | Petrol   | 86  | 0        | 0         | 1300 | 3    | 1015   |
| 988  | 9995  | nan   | 44458  | Petrol   | 86  | 0        | 0         | 1300 | 3    | 1015   |
| 948  | 7750  | nan   | 53000  | Petrol   | 86  | 0        | 0         | 1300 | 3    | 1015   |
| 1236 | 7450  | nan   | 82675  | Petrol   | 86  | 0        | 0         | 1300 | 3    | 1015   |
| 1198 | 7450  | nan   | 89587  | Petrol   | 86  | 0        | 0         | 1300 | 3    | 1015   |
| 1040 | 9500  | nan   | 22178  | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1015   |
| 804  | 8900  | nan   | 73380  | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1015   |
| 1273 | 5950  | nan   | 74567  | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1015   |
| 1210 | 7950  | nan   | 87000  | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1015   |
| 712  | 8750  | nan   | 91246  | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1015   |
| 674  | 6900  | nan   | 104800 | Petrol   | 86  | 1        | 0         | 1300 | 3    | 1015   |
| 1375 | 7750  | nan   | 57000  | Petrol   | 86  | 0        | 0         | 1300 | 4    | 1000   |
| 850  | 8100  | nan   | 65400  | Petrol   | 86  | 1        | 0         | 1300 | 4    | 1000   |

Right click your missing data frame, you will get this window. So, the snippet shown here is the missing data frame and I have flitted out all the rows wherever there are missing values under the age variable. And, if we look at the corresponding columns, there are no missing values accept only one. There is no pattern to the missing values, it is just missing completely in the random case; because it is completely randomly missing under each variable. If you were to subset all the rows wherever there are missing values under kilometer, then that is also the case.

You do not have any pattern to it that holds good for the other columns. I have just shown you for an example, to illustrate that there are no pattern to the missing values that are there here, rather the missing values are completely random in the variable age. So, now, and in that case, we cannot just go ahead and drop all the missing values that are there in your data frame, because if you consider removing a row wherever there are at least one missing value; then you will get rid of 340 observation from 1436 observations, right; that is a huge loss of information that you are going to have. Now, we should have an approach to fill in those missing values.

(Refer Slide Time: 10:00)

## Approaches to fill the missing values



What can be the approaches to fill the missing values? There are several ways to fill in the missing values. In this lecture we are going to look at two ways of approach; the first way is filling the missing values by mean or median, in case of a numerical variable that is one of the standard way or that is one of the simplest way based on which you can fill in all the values. So, for any numerical variable you can look at the average of it or median of it, then you can have that value to fill in all the missing values.

Similarly, you can look at a model value for the categorical variable to fill in all the missing values in your data frame. The model value is nothing, but whichever category of the variable has the highest frequency; if that category occurs most frequently, than you can replace all the missing values with that category itself. So, these are the two ways of approaches that we going to see to fill the missing values here. As I mentioned there are several ways through which you will be able to fill in the missing values as well.

(Refer Slide Time: 10:59)

The slide has a blue header with the GITAA logo and 'Transforming Careers'. The title 'Imputing missing values' is in bold blue text. A grey box contains a bullet point: '• Look at the description to know whether numerical variables should be imputed with mean or median'. Below this is a green box containing the text 'DataFrame.describe()'. Another grey box contains a bullet point: '• Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values'. At the bottom, there is a code snippet 'cars\_data2.describe()' with a cursor arrow pointing to it. The background of the slide shows a person in a yellow shirt speaking. The footer of the slide says 'Python for Data Science'.

So, now we are going to impute the missing values; prior to that we have to look at the description to know whether numerical variable should be imputed with the mean value or the median value right. Because if you are going ahead and filling it filling the missing values with the mean value; that could also be a problem, because if you have any extreme values in your data that might cause you the mean value very high or very low. If you have very low value in your data that can tweak your mean value to the lowest value; but if you have an extreme value in the other end, like higher end then that can also mislead your mean value.

In that case you should always go for median, because while calculating the median you basically sort all the values in ascending order and you will be taking exactly the middle value, if you have odd number of observation. But if you have an even number of observation, you will take the middle two; and then you take an average of it and then you will use that value as the median. We have to know about the distribution of the variables to basically decide on whether we are going to imputed with median or mean. So, dataframe.describe is the function that is used to get the descriptive statistics that summarizes the central .describe function basically.

So, there is a function called describe, you can use that for a data frame, so the syntax is the data frame.describe. The describe function basically generate descriptive statistics that summarize the central tendency dispersion and shape of data sets distribution; and

that excludes all the NaN values and give you those statistics. So, now, we are going to check that using the describe function. So, this is how we use the function, we use the describe function followed by the data frame name.

(Refer Slide Time: 12:51)

```
In [8]: cars_data2.describe()
Out[8]:
 Price Age KM HP MetColor \
count 1436.000000 1336.000000 1421.000000 1430.000000 1286.000000 \
mean 10730.824513 55.672156 68647.239972 101.478322 0.674961 \
std 3626.964585 18.589804 37333.023589 14.768255 0.468572 \
min 4350.000000 1.000000 1.000000 69.000000 0.000000 \
25% 8450.000000 43.000000 43210.000000 90.000000 0.000000 \
50% 9900.000000 60.000000 63634.000000 110.000000 1.000000 \
75% 11950.000000 70.000000 87000.000000 110.000000 1.000000 \
max 32500.000000 88.000000 243000.000000 192.000000 1.000000 \
 Automatic CC Weight \
count 1436.000000 1436.000000 1436.000000 \
mean 0.055710 1566.827994 1072.45961 \
std 0.229441 187.182436 52.64112 \
min 0.000000 1300.000000 1000.00000 \
25% 0.000000 1400.000000 1040.00000 \
50% 0.000000 1600.000000 1070.00000 \
75% 0.000000 1600.000000 1085.00000 \
max 1.000000 2000.000000 1615.00000
```

So, let us look at the output of it; the input being shown here and the output being shown here. So, if you look at the price variable, it basically gives you count, mean, standard deviation; from minimum to maximum it is called as the five number summary, because it gives you minimum maximum and the three quantiles of it. So, the count gives you how many number of observations are there under price; and the mean is nothing, but the average of the price.

The average of the price of the car is 10730 Euros, and the standard deviation is around 3626, the minimum price of the car is 4350, and 25 percent represent that; 25 percentage of the price of the car is less than 8450, and 50 percentage of the cars price is less than 9900 Euros. And if you look at 75 percentage, then you can say that 75 percent of the cars price is less than 11950, and the maximum price of the car from the data frame that we have here is 32500.

So, when you look at the numerical variables the missing values are there under age, kilometer, and horsepower. So, if you look at the mean of the age, on an average the age of the car is around 55 months; and if you look at the median you can see that, median is also represented by 50 percent. So, you can say that and the value being here is 60; the

mean and the median are closer, there is no much deviation from the mean. So, according to the age variable we can go ahead and imputed with the average age itself, instead of going ahead with the median value.

Next when we look at the kilometer, the average kilometer that the car has travelled is 68647 kilometers; and the median kilometer when the car has travelled is 63634 kilometers. But if you see there is a huge difference in the kilometers travelled, there is about 5 thousand and odd difference. So, in this case the mean is far away from the median. So, in this case you cannot go ahead and impute it with the mean, because that gives you a higher number that can be a reason because of any extreme values under the kilometer variable; rather we can go ahead with the imputation of median for the kilometer, because it basically gives us the exactly the middle value of it.

So, we can use median for kilometer. And then we can look at horsepower, the average horsepower of the car is 1000, the average horsepower of the car is 101; and the median horsepower of the car is 110. There is no much difference between 110 and 101, we can fill in the missing values using the mean value.

(Refer Slide Time: 15:54)

Imputing missing values of 'Age'

GITAA  
Transforming careers

- Calculating the mean value of the `Age` variable  
`cars_data2['Age'].mean()`  
Out[11]: 55.67215568862275
- To fill NA/Nan values using the specified value  
`DataFrame.fillna()`  
`cars_data2['Age'].fillna(cars_data2['Age'].mean(),  
inplace = True)`

Python for Data Science

So, now, let us try to impute the missing values of the age variable, so for that, as we know that since there is no much difference between the mean and the median, we are going to impute the age variable with the mean value of it.

So, let us just try to see, what the mean values for age variable. So, let us calculate the mean value of the age variable by using the mean function. And you can use the mean function along with those specified variables from a data frame. I am accessing the age variable from cars\_data2; average age of the car is turned out to be 56 months old.

Now we are going to use this value to replace all the missing values under the variable age. So, to fill any NA or NaN values using a specified value, there is a function called fillna. Again there are so many functions that can be used to replace the missing values with a given value; but here I am illustrating you the example with fillna function, you can use that on a data frame.

So, the syntax being data frame.fillna, we will see how to use this function. Basically the function is used to fill in all the blank or NaN values with the given value. So, basically inside the fillna function you can specify the value with which you are going to replace all the missing values with. So, here I have given the function that needs to be used to calculate a value, and that value can be used to fill in all the missing values.

I have not given the exact value here, rather I have just journalize the function saying; so calculate the Age mean, calculate the mean of the age variable and then use that value to fill in all the missing values of the age variable. And what is the value, I have shown here that is 55.67.

Basically I wanted to fill in all the missing values on the data frame cars\_data2. So, I have given inplace = true. I break the line here for the presentation purpose, you can continue in the same line also. So, now, we have replace the missing values under the age variable. So, now, we are going to impute the missing values of the variable kilometer.

(Refer Slide Time: 18:00)

The slide shows a Jupyter Notebook interface. The title bar says 'Imputing missing values of 'KM''. The notebook contains the following code:

```
In [16]: cars_data2['KM'].median()
Out[16]: 63634.0
```

Below this, another code block is shown:

```
cars_data2['KM'].fillna(cars_data2['KM'].median(),
 inplace = True)
```

A small video player window in the bottom right corner shows a woman speaking. The video player has a blue border and the text 'Python for Data Science' at the bottom.

So, here as we saw from the kilometer variable the mean values really deviating from the median value that might be the reason, because they can be in extreme values under the kilometer variable. So, to get rid of that confusion we are going to use the median function. So, that it will give us the exact middle value of it. So, we are going to calculate the median value of kilometer to see what the median value is?

The median value you can calculated using the function median followed by the variable from a data frame and then you got a value called 63634; that means, that the median kilometer that the car has traveled is 63634 kilometers. So, now, we will use this value to replace all the missing values. So, we are going to use the same function that is fillna. So, I am going to apply the fillna function on to data frame called cars2 under the variable kilometer, and inside the function I have just given cars data2kilometer.median.

So, that the median value will be calculated from the kilometer and then that value will be used to replace all the missing values. And I have given inplace is equal true, so that all the missing values will be replaced in the existing data frame itself. So, now, we are going to impute the values, now we are going to impute the missing values of the horsepower variable.

(Refer Slide Time: 19:30)

The screenshot shows a Jupyter Notebook interface. At the top, there's a header with the title "Imputing missing values of 'HP'" and the GITAA logo. Below the header, there are two code snippets in a code cell:

- Calculating the mean value of the **HP** variable

```
In [19]: cars_data2['HP'].mean()
Out[19]: 101.47832167832168
```

---

- To fill NA/Nan values using the specified value

```
DataFrame.fillna()
cars_data2['HP'].fillna(cars_data2['HP'].mean(),
 inplace = True)
```

At the bottom left of the slide, it says "Python for Data Science". On the right side of the slide, there is a small video player showing a person speaking.

So, if you look at the horsepower variable, we have seen that the mean and median are not very far away. The mean was closer to the median, so in that case we can go ahead and impute the missing values of the horsepower with the mean value. So, here I am calculating the mean of horsepower by accessing the horsepower variable from the data frame `cars_data2`.

And if you look at the output, the average horsepower of the car is 101.47 and odd. So, now, we are going to use this value to replace all the missing values of horsepower. So, using the same function `fillna`, I am replacing all the missing values using the mean of horsepower and we are filling the missing values in the existing data frame by giving `inplace = true`. Now, we have replaced the missing values under all the numerical variables like age, kilometer, and horsepower.

(Refer Slide Time: 20:25)

## Imputing missing values of 'HP'



- Check for missing data after filling values

```
In [56]: cars_data2.isnull().sum()
Out[56]:
Price 0
Age 0
KM 0
FuelType 100
HP 0
MetColor 150
Automatic 0
CC 0
Doors 0
Weight 0
dtype: int64
```

Python for Data Science



So, now let us check for the missing data after filling those values. So, now, we have replaced the missing values under age, kilometer, and the horsepower; let us see whether there are any missing values under that or not, yes there are no missing values. But if you can see there are still missing values under FuelType and MetColor, because we have not still touched on that variables. So, now we are going to impute the missing values of the categorical variables.

(Refer Slide Time: 20:52)

## Imputing missing values of 'FuelType'



### Series.value\_counts()

- Returns a Series containing counts of unique values
- The values will be in descending order so that the first element is the most frequently-occurring element
- Excludes NA values by default

```
cars_data2['FuelType'].value_counts()
```

```
Out[28]:
Petrol 1177
Diesel 144
CNG 15
Name: FuelType, dtype: int64
```

Python for Data Science



So, as I mentioned earlier for any categorical variable, the simplest way to fill in the missing values of categorical variable is to check for the most frequently occurring category and then replacing that with the missing values. So, for that we need to understand what are the categories are there under the FuelType, right; and we need to understand the corresponding frequencies as well. So, for that there is a function called value\_counts that can be applied on to a series.

And the value\_counts basically returns a series containing counts of unique values. The output of the function will give you the values in the descending order, so that the first element is the most frequently occurring element and it gives you the frequencies of each category by excluding all the NA values by default. So, we do not need to remove the missing values and then get the frequencies of it, by default it excludes all the missing values.

So, this is how we use the function. cars\_data2 is the data frame name, under that I am accessing the variable called FuelType; by giving.value\_counts you will get an output which is shown here. For example, the petrol has the highest frequency that is 1177 observations are of petrol FuelType, and the only 144 cars have diesel FuelType, and only 15 cars has CNG FuelType. In this case it is very clear that the mode value would be petrol; if you want to get the mode value separately, then you can also do that, you can also use the same function you can give the index as 0.

(Refer Slide Time: 22:25)

## Imputing missing values of 'FuelType'



### Series.value\_counts()

- To get the mode value of FuelType

```
cars_data2['FuelType'].value_counts().index[0]
Out[29]: 'Petrol'
```

- To fill NA/Nan values using the specified value

### DataFrame.fillna()

```
cars_data2['FuelType'].fillna(cars_data2['FuelType']\n .value_counts().index[0],\n inplace = True)
```



Since in python the indexing starts from 0, the 0 will give you the first value of the output of value\_count. What would be the first value of the value\_counts output? It would be petrol, because since it has the highest number of observation it was shown at the top. So, when you access it using the index 0, you will get a value called petrol.

Now you will be able to use this value to fill all the missing values of FuelType variable. Let us see how to do that; you can use the same function that is.fillna. I am using the same function from the cars data2data frame; I am accessing the variable FuelType. And I am applying the fillna function on it, saying that and inside the function we need to give a value, so that the missing value will get replaced.

Here I am not giving petrol directly, rather I am giving the whole function here, so that the code will be generalized; it is you do not need to tweak it whenever you want it, because just to write a generalized code. So, here I am filling all the missing values with the most frequently occurring category of FuelType. So, petrol will be used to fill in all the missing values of the variable FuelType. And you know here, we have used inplace = true, so that we are replacing the existing data frame itself.

Now, by using fillna we have replace the FuelType missing values with the petrol FuelType, we are going to now consider the MetColor variable.

(Refer Slide Time: 23:58)

## Imputing missing values of 'MetColor'

`mode()`

- To get the mode value of **MetColor**

```
In [39]: cars_data2['MetColor'].mode()
Out[39]:
0 1.0
dtype: float64
```

`DataFrame.fillna()`

```
cars_data2['MetColor'].fillna(cars_data2['MetColor']\n .mode()[0], inplace = True)
```



Here also we are going to impute all the missing values of MetColor with a model value of the variable MetColor. MetColor basically the metallic color of the car, basically it will have a two values. As we know the MetColor has two values; 0 represents the car does not have a metallic colour, and 1 represents the car has a metallic color. So, there are several ways using which you will be able to fill in the model value. In the previous example, we have seen how to fill in the missing values using value\_counts and accessing it from the index.

Now, we are going to use the mode function to calculate the model value of the categorical variable. So, to get the mode value of the MetColor, we can use it as data frame and access the respect to variable and then use the function.mode. The.mode function gives you an output which is shown here, it basically gives the value along with the index. The mode value of the MetColor is 1.0, since it has missing values it is giving you an floating points that is 1.0, ideally it should be 1. And the index of the value is 0. So, why you are also getting in index, while calculating the median or mean you would not be able to get the index.

Because median or mean can only be a scalar value; but mode cannot be a scalar value or mode need not be a scalar value. Mode can either be bimodal, there are many cases where a variable can have bimodal or more than two models, so in that case you will have continuous index. In this case, since our variable MetColor has only one value as more you are getting only 0. So, now, to fill in all the missing values using the specified value, we use the function.fillna. So, now, I have use the fillna function on to a variable metallic color that is from the data frame cars\_data2; and inside the function I have specified that, calculate the mode from metallic color variable and replace all the missing values.

So, here I have also given the index to it, that is just because the output always comes with the index; if you have bimodal, then you will always have another index as 1 and you will have another value, that will be the case where you have a both the categories have the same frequency. In that case you can choose any one of the value which is required for the analysis; but here which is more important for the analysis. But here, since I am very clear that I have only one value and that the model values index is 0, I have given it has 0. And I am making all the modifications onto the same data frame called data2, so I have given inplace = true.

So, now we have replaced both FuelType and MetColor variable. Now we have replaced combinedly both numerical as well as categorical variables using different approaches.

(Refer Slide Time: 27:00)

The screenshot shows a Jupyter Notebook cell with the title "Checking for missing values". The code `cars\_data2.isnull().sum()` is run, resulting in the output:

```
In [59]: cars_data2.isnull().sum()
Out[59]:
Price 0
Age 0
KM 0
FuelType 0
HP 0
MetColor 0
Automatic 0
CC 0
Doors 0
Weight 0
dtype: int64
```

A woman in a yellow jacket is visible in the bottom right corner of the slide.

So, let us check for the missing data after filling in all the values, using the `isnull.sum` function that has to be applied on to the data frame. In a data frame `cars_2` check whether there are any missing values. If there are any missing values get the sum of it under each column that is what the function describes. And if you see here none of the columns have missing values now; since we have already imputed all the missing values with the logical approach that we have imputed for all of the variable separately using a same logic, right.

For example for a numerical variable we have imputed with the mean or median, and for all categorical variables we have imputed with a model value. So, this can be the case where you have only missing values in 6 to 7 columns. What will you do whenever you have missing values in multiple columns like 50 to 60? In that case there can be a simple function which will do the imputation in one shot; we are going to use that.

(Refer Slide Time: 28:00)

The slide has a blue header with the title "Imputing missing values using lambda functions" and the GITAA logo. Below the header, there are two bullet points:

- To fill the NA/ NaN values in both numerical and categorial variables at one stretch
- Check for missing data after filling values

The code block contains the following Python code:

```
cars_data3 = cars_data3.apply(lambda x:x.fillna(x.mean()) \n if x.dtype=='float' else \n x.fillna(x.value_counts().index[0]))
```

The Jupyter Notebook output shows the result of running the code:

```
In [52]: cars_data3.isnull().sum()
Out[52]:
Price 0
Age 0
KM 0
FuelType 0
HP 0
MfgColor 0
Automatic 0
CC 0
Doors 0
Weight 0
dtype: int64
```

A woman in a yellow jacket is visible in the bottom right corner of the slide.

So, to fill the missing values that is NaN values in both numerical and categorical variables at one shot we are going to use a lambda function. Apply functions can be used whenever you want to perform any operations column wise or row wise, it can be used any other cases; but here, this apply function will be used across columns.

And if you see here, I am not using the cars\_data2; because the cars\_data2 does not have any missing values now, since we have already replaced everything. If you can recall, I have created another copy from the cars\_data while reading itself. In that case the cars\_data3 is still have missing values, because we have not touch this data at all. So, I am using that data set and I am using and apply function, so that whatever function that I am giving inside the apply function that will be applied across all the columns.

And the function that I am using inside the apply function is a lambda function; lambda function is an anonymous function and a powerful function whenever you use it inside a function. And here I am using it inside an apply function; and I am defining a function call x, and I am using a predefined function called fillna to apply it across all the columns. So, what it basically does means? fillna we know that, wherever there are missing values it will replaced with the given value that we are giving it inside the function.

So, what value that we are giving here? We are not giving a value, rather we are giving conditions to it; we are giving if else conditions, like basically our target is to fill in all

the missing values of numerical variable with mean, and to fill in all the missing values of categorical variable with mode. In that case, give a condition saying that calculate the mean of each and every variable, if the various data type is a float; else calculate the mode value of each and every variable and then use that value to replace with.

So, that is what the function does. So, using this function we have just replaced all the missing values at one shot; wherever there are missing values in numerical variable everything has been replaced with the mean value of each and every variable. And similarly wherever there are missing values under any categorical variable, for each and every categorical variable separately the model value will be calculated and then that will be used for filling those missing values.

So, once we do that, let us check for missing data after filling all the values. So, I am checking whether there are any missing values under the data frame cars\_data3. The output is also shown here, if you see all the values are 0 here; because we have already imputed all the missing values that is why you are seeing all zeros here.

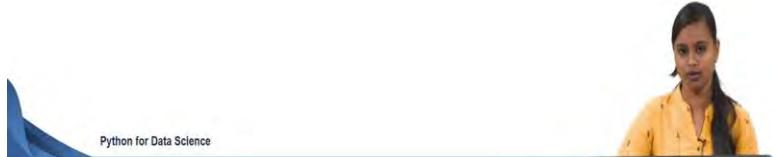
So, now we do not have any missing values under any variable; but as I mention this is not the only way that you can go ahead with the imputing with missing values. There are several approaches that are available to impute all the missing values, the way that we have shown here is the simplest of all by just imputing it with mean or the model value.

(Refer Slide Time: 31:16)

## Summary



- Identifying missing values
- Approaches to fill the missing values



Let us summarize whatever we seen till here; we have seen how to identify the missing values, you also seen how you can identify the pattern of your missing values by subsetting all the rows and seeing whether one row has only one column value missing or multiple columns are missing; and then by arriving at a decision that the missing values are there randomly under each column; then we found out an approach to fill in all the missing values.

We have seen two approaches that is mean or median imputation and mode imputation; mean or median imputation is only for a numerical variable and you use mode imputation for a categorical variable.

Thank you.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 22**  
**Introduction to Classification Case Study**

Welcome to the case study descriptions for this course. I hope till now you have practiced and learned enough Python, to be able to do programming comfortably. Our aim now is to basically help you understand, how you could use Python as a tool in solving actual data analytics or data science problems.

And we are going to give you two case studies that we will teach; and one of them is on classification problem, and the other one is a function approximation or a regression problem. So, what I will do, is in this lecture I will explain or Introduce this Classification Problem that you are going to solve. And then we will go through the data and actual solution to the problem in another lecture.

(Refer Slide Time: 01:15)

**Problem statement**

- "Subsidy Inc. delivers subsidies to individuals based on their income
- Accurate income data is one of the hardest piece of data to obtain across the world
- Subsidy Inc. has obtained a large data set of authenticated data on individual income, demographic parameters, and a few financial parameters.
- Subsidy Inc. wishes us to :  
    Develop an income classifier system for individuals.

The Objective is to:

Simplify the data system by reducing the number of variables to be studied, without sacrificing too much of accuracy. Such a system would help Subsidy Inc. in planning subsidy outlay, monitoring and preventing misuse.

So, the problem statement is as follows; let us say there is a company called Subsidy Inc. which delivers subsidies to individuals based on their income. So, if your income levels are different, you get different subsidies and so on. However and whenever someone new comes in, it is very difficult to get information about personal income. So, that is one of the most difficult pieces of data to get. So, here is an idea in terms of using an already

existing database, where we have various attributes of these people and wherever personal income has been disclosed we also have that as a data.

So, what we are trying to do here is, we are trying to see, if we can somehow classify an income level based on the attributes that we have for individuals. So, basically this becomes then a classification problem, where we are going to say the individual has an income level beyond a certain value or below a certain value. In this case we have this income level as 50000; and what we want to do is we want to classify individuals with less than 50000 personal income and greater than 50000 personal incomes.

So, this is basically a simple classification technique. And you one might ask, if you were able to do this how can you use it; if you have multiple customers that come in, then we can kind of identify the proportion of customers who are likely to be having an income less than 50000 and the percentage of people more than 50000; and that could allow us to be able to plan an outlay of resources and so on. So, that is one use case that you can think of.

Another use case is really, if someone actually discloses an income and if it is an outlier in terms of, let us say someone says they are earning less than 50000; but our classifier says with all these attributes it is very very likely the income is greater than 50000, then it might allow us to look at those particular cases in more detail, so that there is no misuse of these schemes.

(Refer Slide Time: 03:43)

| Variable description |         |                                                        |                                                        |                         |
|----------------------|---------|--------------------------------------------------------|--------------------------------------------------------|-------------------------|
| Variables            |         | Data Type                                              | Description                                            | Categories of variables |
| age                  | integer | The age of the individual in years                     | --                                                     |                         |
| JobType              | string  | Working status of person, which sector does he work in | Federal-gov, Local-gov, Private & 5 more...            |                         |
| EdType               | string  | The level of education                                 | 10 <sup>th</sup> , Masters, Doctorate & 13 more...     |                         |
| maritalstatus        | string  | The marital status of the individual                   | Divorced, Never-married, Married-AF-spouse & 4 more... |                         |
| occupation           | string  | The type of work the individual does                   | Armed-Forces, Sales, Tech-support & 11 more...         |                         |

So, whenever we have a problem, data science problem all of this starts with data. So, we first need to understand the data that we have. So, in this case you will be given a data set, which was about 31978 samples; that is 31978 individuals for whom this data is available including their personal income. And this is the data set that is going to be used in developing this classifier. Now I talked about features for this individual. So, we are going to look at multiple features. In this case we have 12 features that we are going to look at, and one variable which is personal income which makes the total number of variables 13.

So, if you want to think of this data in a matrix form, then we have about a matrix of the size 31978 times 13. Now let me quickly go through the variable definitions and what type of variable they are and their description. And this will help you to kind of visualize this data once you actually start working on this case study with the instructors. So, one variable is JobType. So, this is a string and this basically tells us about the working status of the person is he in and works which sector he or she works in, it is Federal government, Local government, Private and 5 more such identifiers for JobType.

Education type is again string, which tells you the level of education and the multiple possibilities or 10th, Masters, Doctorate and 13 more. Marital status is again a string, which tells you the marital status of the individual which could be Divorced, Never married and so on. Occupation is again another string and the type of work that the individual does it, could be Armed Forces, Sales, Tech support and 11 more categories in this case. And as we talked about this you can start getting an idea of why these variables are relevant for this problem; and you can really see that all of these variables are relevant, because the pay depends on the sector that you work in, the level of education, the type of work you do and so on.

So, as we go along you will see that there is this notion of picking relevant data that is useful for this problem.

(Refer Slide Time: 06:27)

| Variable description |         |           |                                                                                                                                |                                   |
|----------------------|---------|-----------|--------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| Variables            |         | Data Type | Description                                                                                                                    | Categories of variables           |
| relationship         | string  |           | Relationship of individual to his/her household                                                                                | Husband, wife, own-child & 3 more |
| race                 | string  |           | The individual's race                                                                                                          | Black, White & 3 more             |
| gender               | string  |           | The individual's gender                                                                                                        | Male, Female                      |
| capitalgain          | integer |           | The capital gains of the individual (from selling an asset such as a stock or bond for more than the original purchase price)  | --                                |
| capitalloss          | integer |           | The capital losses of the individual (from selling an asset such as a stock or bond for less than the original purchase price) | --                                |
| hoursperweek         | integer |           | The number of hours the individual works per week                                                                              | --                                |

So, moving on, there is a variable called relationship which tells you the relationship of an individual to his or her household, so this could have value such as husband, wife and so on. The race of the individual black white and 3 more; gender of the individual male, female. And then there are other features that talk to the real financial aspects of the individual.

So, there is one variable called capital gain which is the capitalgain of the individual which is a rounded off integer in this case; capitalloss another integer again a rounded off integer which basically as the name suggests is a capital loss. And the hoursperweek that the individual works again it is rounded off and the type is as integer.

(Refer Slide Time: 07:25)

The screenshot shows a software interface with a dark blue header bar. The main title is 'Variable description'. Below it, there's a status bar with icons and the text 'Data Analytics'. The central part is a table titled 'Variable description' with four columns: 'Variables', 'Data Type', 'Description', and 'Categories of variables'. The table has two rows. The first row contains 'nativecountry' (string), 'The native country of the individual', and 'United-States, Cambodia, Canada & 39 more...'. The second row contains 'SalStat' (string), 'The outcome variable indicating whether a person's salary status', and 'less than or equal to 50,000, greater than 50,000'. The bottom right corner of the table area has a small number '5'.

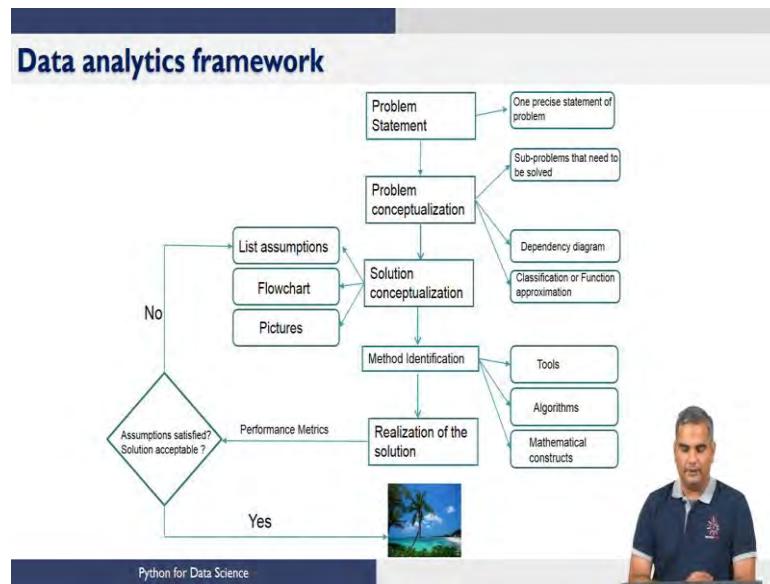
| Variables     | Data Type | Description                                                      | Categories of variables                           |
|---------------|-----------|------------------------------------------------------------------|---------------------------------------------------|
| nativecountry | string    | The native country of the individual                             | United-States, Cambodia, Canada & 39 more...      |
| SalStat       | string    | The outcome variable indicating whether a person's salary status | less than or equal to 50,000, greater than 50,000 |

And also there is a variable which says which nativecountry that this person comes from, which could be United States, Cambodia, Canada and so on. And then finally, the output variable which is what we are trying to build a classifier for which is salary status, which is outcome variable, which is basically in this case just split into 2 categories less than or equal to 50000 and greater than 50000; those are just the two categories.

So, basically what we have here is a binary classification problem; where there are two categories less than or equal to 50000 and greater than 50000. And based on the feature set that we have, we have to basically build a classifier that will be able to tell us once we get a new customer. And we put the features of that new customer into the system; get an information about salary status. Now, notice that some of this input features themselves might be difficult to get such as capital gains and capital loss and so on.

Nonetheless what we will try to do is, we will try to see what best in terms of a classifier we can build and then one of the ideas as I will point out later is also to reduce the number of input variables and themselves. So, instead of 12, let us say we build a classifier with these 12 variables and then we drop a few and then see the performance does not change much. But from a data collection effort in the future it might be much easier to get information about certain things and not other things, then we can see whether we can use those of ideas to actually build a classifier that is practical and useful in the future.

(Refer Slide Time: 09:13)



Now, whenever you do a data science problem, you basically have to think about it in some flow process. This flow process I have explained in the other courses that I have taught; but just quickly as an overview in this course on python for data science. I just want to go through this, so that we can see how we kind of map the solution that we build in this case to this flow chart.

The very first thing is to basically say, what the process statement or the problem is, right. So, now, in this case we have already done this, but if someone just came and gave you this data; and then says you know we are collecting a lot of data and this is the data that we have, is there something worthwhile or useful we can do with this data. Then one problem statement might be to say, look can I figure out the salary status of the person based on just the features in terms of that individual several attributes.

And that is a precise problem statement that you can post here; of course, with this data itself you can post other problem statements, but here we are going to stick to this. And typically when it is a very large data science case study, clearly this is not, this is in terms of data still 30000 odd data is not small by any means; but it is still not large by any means either. More importantly this is a very simple binary classification problem.

In many cases when you are presented with a problem, it is really not obvious exactly what data science problem you need to solve. So, in those cases, you have to think about some notion of problem conceptualization. Breaking down a very loosely worded

problem statement into multiple smaller problem statements; and then under identifying these smaller problem statements as what type problems they are either classification or function approximation problems. And then once you are able to solve the smaller problems, how do you put them in some logical arrangement of solution. So, that you can solve the larger loosely frame problem.

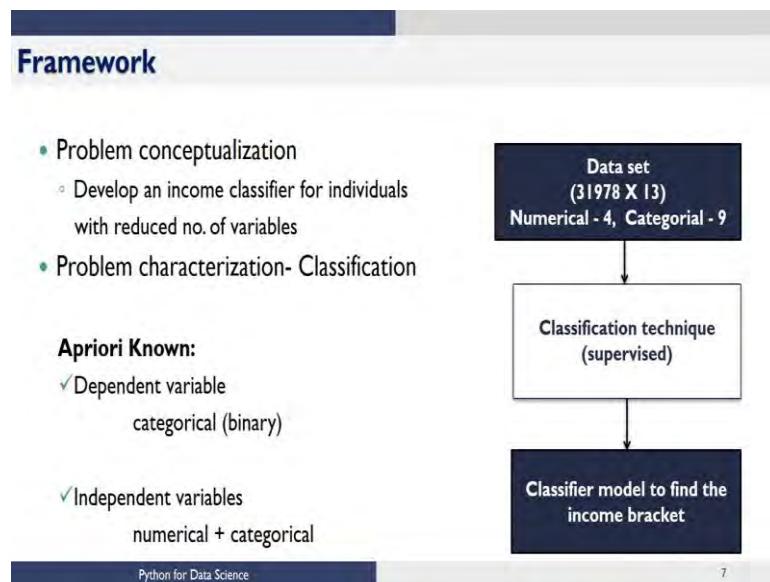
So, that is actually where the intellectual exercise of data science comes in. But in this case, in this first course this is a simple problem you directly understand that this problem can be conceptualized as a binary classification problem, so there is nothing much more to do here in this case. Then once you have a binary classification problem, you know there are several tools that exist some of which we have taught to you.

So, how do you use a certain tool or how do you pick one tool out of these many tools that are available. Again there is a very logical way in which you can do this, you can make some assumptions about the underlying data distributions and model types and so on. And then choose a tool that is relevant for those assumptions and then see whether the tool works and all that. That is little more sophisticated data science thinking.

Again in this course what we are going to do is, we are going to simply look at a couple of tools and then run those tools on this data set; and then basically we are going to judge which tool is better based on just outcomes in terms of the prediction error or the confusion matrix and so on. So, once you do that method identification, then basically you have to realize that solution in some platform of choice.

As I mentioned before the two most popular platforms of choice are python and R. And in this case we are going to use python to solve this problem. And, once you solve this problem and you are happy with the results; then that is the end of this whole data analytics solution, strategy or methodology.

(Refer Slide Time: 13:30)



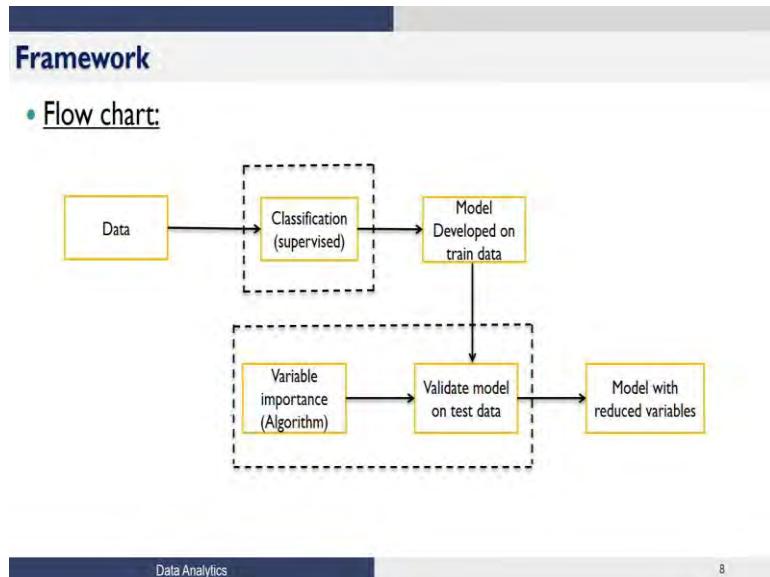
So, now, let us look at this. So, as we said before, problem conceptualization is pretty simple; here we are developing an income classifier for individuals. We have also added another small aspect to this problem which is not really obvious unless you start thinking about it carefully; like I said before you could say I will use all the features and build the model, but we would also like to see if we can reduce the number of features and get comparable performance of the classifier.

And as I mentioned before the reason is, then it makes this solution, practical to be used in the future. So, for example, if you are totally tied to knowing everything in the input future and without those you cannot find a solution, then you might not be able to use it as effectively. Whereas if you were able to reduce the number of feature, so that there are a few things that you need to know to be able to make a prediction, those kinds of systems are much more useful.

So, that is another additional wrinkle I would say, that we have introduced into this problem. So, again we had the data set of 13 columns; one of them is an outcome variable, the other 12 are independent or input variables from our viewpoint. And of those 13, 4 are numerical and 9 are categorical; and this is a classification problem. In fact, this is a binary classification problem and this is also a supervised classification problem; because in the data set that we are going to use to build a classification model, for every data sample we know the outcome variable value.

So, basically we are going to supervise our learning algorithm. So, that it learns the given characteristics of the data. So, Apriori known all the dependent variable, dependent variable which is categorical binary we have already talked about it and also the independent variables which are basically a combination of numerical and categorical variables.

(Refer Slide Time: 15:47)



So, if you think about putting all of this into a flow chart based on the extra wrinkle that we threw into the problem in trying to reduce the number of input variables that we need to solve the problem. So, we start with data, then we build a classification model supervised classification model; and then once we build the model we validate the model on test data. So, this is another important idea in all of data science and data analytics; if you use all the data that is given to you and build a model, then the model is very likely to do well because it has seen all the data.

So, basically all of data science or data analytics is useful, only when we can use that really in predictions as we see more samples or more data that comes in. So, we want to have some notion of how well our algorithm is likely to do when I get new data. So, to identify that what we do is, we basically split the data into what we call as train and test data; and when we build a model, we basically build a model using the train data, and then we test this model based on the test data we have left out within the data that we had.

Now, if our model does quite well on the test data which is not seen before; then we can be comfortable that this model is likely to work in the future also. So, that is what we have here as validate model on test data. And once we have this validated model where we are happy with the performance on the test data, then we can do something called a variable importance step; which is, if I have these 12 variables that I am using to predict the salary status, is there some way in which I can look at these variables and order them in terms of importance.

So, what do we mean by importance? So, if I were to ask you, pick one variable which is the most important for predicting this, what would that variable be right; then if I ask you pick two variables which are important for predicting the salary level as less than 50000 or 50000; what would those two variables be from these 12 variables input variables? So, if you keep asking this question, at some point once we have a certain set of variables which you can use to figure out, whether the salary level is less than equal to 50000 or greater than 50000.

Then adding more variables, if it is not really improving the performance tremendously, particularly on the test set; then what you do is, you basically say you know maybe 5 or 6 or 4 or whatever the number turns out to be those many number of variables are good enough to predict the outcome variable. Again as I mentioned before, what this does is that, it makes it usable in future.

(Refer Slide Time: 18:59)

## Framework

- Solution conceptualization
  - Identify if data is clean
  - Look for missing values
  - Identify variables influencing salary status and look for possible relationships between variables
    - Correlation, chi-square test, box plots, scatter plots etc.
  - Identify if categories can be combined
  - Build a model with reduced number of variables to classify the individual's salary status to plan subsidy outlay, monitor and prevent misuse



So, this is what we will do in terms of the steps and you will see python code and python exercises, which does each one of these steps and you will be taught this based on the other things that you have been taught in python. The first step is to identify is data is clean. What that means, is that if there are certain categories for each of these variables, all the data are they in the same category.

So, for example, if you are expecting to see a string, do you see a string or in some case some other number and so on. So, basically some kind of consistency check in terms of your understanding of the data and does the data confirm to your understanding is the first step. And when you do this test of whether data is clean or not, there might be cases where there are certain data points that are missing.

So, for example, if there 12 features that we are talking about, not all individuals might have disclosed all the 12 features. So, there might be rows where certain columns might be missing. So, for example, some people might not have disclosed capital gains or capital loss and so on. So, those are rows with missing values, those are samples that are not complete in themselves. So, how do you handle this, how do you look for missing values; there are two things you can do, one is to remove all the samples which have incomplete data.

The upside is you are not going to introduce any artificiality into this problem; but the downside is you will lose a lot of data points. And it is important, because when we have 12 features, let us say there is only one feature that is missing; if you throw that data, you are also throwing the 11 other feature information with that sample. So, you lose some information through this loss of data. The other approach is to say somehow I am going to fill that missing value, and there are multiple ways of filling missing values.

Then the upside is we retain the whole data set, but the downside is we introduce some artificiality and which we do not know whether it is right or wrong. Because whatever procedure that you come up with to fill your missing data, it is based on some assumptions you make. And if those assumptions are not truly satisfied or they are violated, you might be imputing or putting in values which might not be close to the true value. So, that is the downside. So, we have to think about how we do this and basically depending on what percentage of data is missing and so on we make some judgments.

And basically, then we will start looking at some descriptive statistics to think about; how to identify variables, which impacts salary status tremendously and are there relationships internally between these input variables and so on. And if there are these relationships between different variables; so for example, if a particular category variable when it takes a value, another category variable takes some value and they are correlated all the time. Let us say if this is taking some value, I can predict the other categorical variable will take some other value; if you have situation like that, then using both the categorical variables in your classification is not very useful at all, because it is redundant information.

In those cases can be combined in this category, so that we do better; better could mean better in terms of performance or better in terms of future use. And finally, we want to build a model with this reduced number of variables, to classify the individual salary status; and basic idea or the use case for this is, you can plan subsidy outlay, monitor and prevent misuse.

(Refer Slide Time: 23:06)

## Framework

- Method identification
  - Logistic Regression
  - Random Forest
  - K Nearest Neighbors
- Realization of solution
  - Evaluate performance metrics
  - If assumptions are satisfied and solutions are acceptable then model is good



Python for Data Science

So, as I said before, there are very rational ways of choosing techniques for different problems, but; that means, that you have to have some information, some knowledge about the underlying data and so on. If you are going to have really not much information, you are just going to try different things; one standard thing that everyone does nowadays, is to take a number of these techniques and then apply all of them on the

same problem, and simply pick the technique that does the best, right. So, that is a very practical and utilitarian viewpoint of data science or data analytics algorithms. And it works in many cases of course, the upside is you do not have to really know too much about what the algorithm does even; you just need to know if I have a binary classification problem, there is this laundry list of techniques that I can use. So, if you have understanding at that level, then you can start using it.

The downside is subtle and in most cases it might not be a problem, but in some cases it can be a serious problem; which is that, if the data is you know biased in terms of how the sampling has been done and so on, and it is not truly representative of what is happening. Then sometimes you might have a technique which works really well with this data, but when new data comes in it might do very poorly. So, always it is a good idea to understand the data and then kind of pick a technique; but in this case what we are going to do is, we are going to pick all of these techniques and then run them.

And once you run those techniques, this is where python comes in; you whatever you have learned in your coding and so on, you will use that to realize the solution. And basically you will evaluate performance metrics. So, in this case how well it does on the test data, is a good performance metric to check. And typically if we had made some assumptions about the data, then you will kind of tie in this with the assumptions that you made and then see whether everything checks out.

But in this case, if you are just picking a list of techniques and then trying them on; all you are going to look at is, how well is it doing on my test data and then simply pick the best method in terms of the results on the test data. So, this is the introduction to the problem. This will be followed by a lecture on actually a solution strategy with python code, in terms of loading the data, doing all of this. So, that you get the feel of doing a complete data science project in this course.

Thank you.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 23**  
**Classifying Personal Income**

Hello all, welcome to the lecture on the case study. So, in this lecture, we are going to look at a case study called Classifying Personal Income, where we are going to see how to solve that problem using python.

(Refer Slide Time: 00:27)

**Problem statement**

- Subsidy Inc. delivers subsidies to individuals based on their income
- Accurate income data is one of the hardest piece of data to obtain across the world
- Subsidy Inc. has obtained a large data set of authenticated data on individual income, demographic parameters, and a few financial parameters
- Subsidy Inc. wishes us to :
  - Develop an income classifier system for individuals

**The Objective is to:**

Simplify the data system by reducing the number of variables to be studied, without sacrificing too much of accuracy. Such a system would help Subsidy Inc. in planning subsidy outlay, monitoring and preventing misuse.

Python for Data Science

So, let us look at the problem statement of the case study that we are going to solve. Subsidy inc. is a company which delivers subsidies to individuals based on their income. And accurate income data is one of the hardest piece of data to obtain across the world. So, they have obtain the large set of data on individual income, demographic parameters, and based on few financial parameters. So, they wish us to develop an income classify system for individuals.

The main objective of the case study is to simplify the data system by reducing the number of variables to be studied without sacrificing too much of accuracy, so that the data collection part would be easier. If we have very few parameters that needs to be measure, then the data collection part will be easier. So, that such a system would help subsidy income in planning subsidy outlay monitoring and preventing misuse. Let us

quickly look at the variables that are available in the case study that we are going to solve.

(Refer Slide Time: 01:34)

| Variable description |           |                                                        |                                                        |
|----------------------|-----------|--------------------------------------------------------|--------------------------------------------------------|
| Variables            | Data Type | Description                                            | Categories of variables                                |
| age                  | integer   | The age of the individual in years                     | --                                                     |
| JobType              | string    | Working status of person, which sector does he work in | Federal-gov, Local-gov, Private & 5 more...            |
| EdType               | string    | The level of education                                 | 10 <sup>th</sup> , Masters, Doctorate & 13 more...     |
| maritalstatus        | string    | The marital status of the individual                   | Divorced, Never-married, Married-AF-spouse & 4 more... |
| occupation           | string    | The type of work the individual does                   | Armed-Forces, Sales, Tech-support & 11 more...         |

Python for Data Science

The first variable is age which represents the age of the individual in years. Next one is a JobType which represents the working status of a person which sector does he work in like federal government local government and so on and so forth. Next is the educational type the level of education like 10<sup>th</sup>, masters, doctorate and so on. Next one is the maritalstatus the maritalstatus of the individual, whether a person is married the worst or never married and so on.

Next one is the occupation the type of work the individual does whether the individual is working in an armed forces or sales or technical support and you have eleven more categories like that.

(Refer Slide Time: 02:20)

The screenshot shows a table titled "Variable description" with a header row containing "Variables", "Data Type", "Description", and "Categories of variables". The table has six rows of data. The "Variables" column lists "relationship", "race", "gender", "capitalgain", "capitalloss", and "hoursperweek". The "Data Type" column lists "string", "string", "string", "integer", "integer", and "integer". The "Description" column provides details for each variable, and the "Categories of variables" column lists the possible values or categories for each variable. The table is set against a background of a presentation slide with the title "Variable description" and the text "Size: 31,978 x 13" and "Data file: income.csv".

| Variables    | Data Type | Description                                                                                                                    | Categories of variables           |
|--------------|-----------|--------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| relationship | string    | Relationship of individual to his/her household                                                                                | Husband, wife, own-child & 3 more |
| race         | string    | The individual's race                                                                                                          | Black, White & 3 more             |
| gender       | string    | The individual's gender                                                                                                        | Male, Female                      |
| capitalgain  | integer   | The capital gains of the individual (from selling an asset such as a stock or bond for more than the original purchase price)  | --                                |
| capitalloss  | integer   | The capital losses of the individual (from selling an asset such as a stock or bond for less than the original purchase price) | --                                |
| hoursperweek | integer   | The number of hours the individual works per week                                                                              | --                                |

Next one is relationship of the individual to his or her household. The next one is the race; it is being represented by black, white and 3 more. Next one is the gender the individual's gender being represented as male and female. The next one is the capitalgain; the capital gains of the individual basically from selling an asset such a stock or bond for more than the purchase price.

Next one is the capitalloss the converse of that, the capital losses of the individual from selling an asset such as a stock or bond for less than the original purchase price. Next one is the hoursperweek, the number of hours that the individual works per week in a company.

(Refer Slide Time: 03:03)

The screenshot shows a table titled "Variable description" with a header row containing "Variables", "Data Type", "Description", and "Categories of variables". Below the header, there are two rows of data. The first row corresponds to the variable "nativecountry" (string type), which is described as "The native country of the individual" and has categories including United States, Cambodia, Canada, and 39 more. The second row corresponds to the variable "SalStat" (string type), which is described as "The outcome variable indicating whether a person's salary status" and has categories less than or equal to 50,000 and greater than 50,000. The table is displayed on a slide with a blue header and footer.

| Variables     | Data Type | Description                                                      | Categories of variables                           |
|---------------|-----------|------------------------------------------------------------------|---------------------------------------------------|
| nativecountry | string    | The native country of the individual                             | United-States, Cambodia, Canada & 39 more...      |
| SalStat       | string    | The outcome variable indicating whether a person's salary status | less than or equal to 50,000, greater than 50,000 |

And we have nativecountry of the individual being represented with like categories like United States, with countries like United States, Cambodia, Canada and so on and so forth. Next one is the last one is the salary status which is the outcome variable. The outcome variable indicating whether a person salary status, whether it is a less than or equal to 50,000.

So, it has basically two categories less than or equal to 50,000 and greater than 50,000. So, this is the overall idea about the problem and the description of the variables. Now, we are going to go back to spyder and we are going to see how to solve this problem.

(Refer Slide Time: 03:45)

The screenshot shows the Spyder IDE interface. On the left, a code editor window displays a Python script named 'temp.py'. The code imports various packages like pandas, numpy, and seaborn, and performs operations such as data partitioning and model selection. On the right, an 'IPython' console window titled 'Usage' shows the execution of the script. It lists the Python version (3.6.4), the date (Jan 16 2018), and the IPython version (6.2.1). It also shows the first three command-line inputs: 'import pandas as pd', 'import numpy as np', and 'In [3]:'. The bottom status bar indicates the current line (Line 16), column (Column 18), and memory usage (42%).

```
=====#
CLASSIFYING PERSONAL INCOME
=====#
Required packages
To work with dataframes
import pandas as pd
To perform numerical operations
import numpy as np
To visualize data
import seaborn as sns
To partition the data
from sklearn.model_selection import train_test_split
Importing Library for Logistic regression
from sklearn.linear_model import LogisticRegression
=====#
```

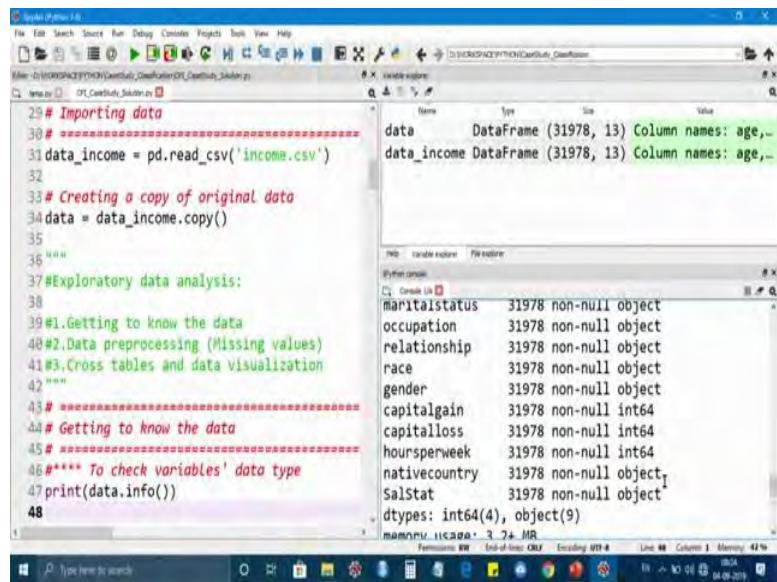
In [1]: import pandas as pd  
In [2]: import numpy as np  
In [3]:

So, now, let us start by importing required packages. To work with data frames, we are going to import pandas as pd. And to perform any numerical operations on it, we are going to import numpy as np. Followed by that, we will be visualizing the data using the seaborn library. So, we are going to import seaborn as sns. Followed by that, we are going to import train test split from the package called sklearn and the model\_selection is a sub package under sklearn.

So, we are going to import train\_test\_split from sklearn package. After that we are going to import LogisticRegression from sklearn linear\_model. And to look for performance matrix, we are going to import accuracy and confusion matrix from sklearn matrix. So, now, we have imported the required packages. Now, let us import the data into spyder. So, income is the filename and it is in csv format.

So, now let us import the data into spyder. Income is the filename and it is in csv format, we going to read that and we have storing it onto a data frame call data\_income. So, once we read the data, we can see it under the variable explorer.

(Refer Slide Time: 05:24)



The screenshot shows a Jupyter Notebook interface. On the left, a code cell contains the following Python script:

```
29 # Importing data
30 # *****
31 data_income = pd.read_csv('income.csv')
32
33 # Creating a copy of original data
34 data = data_income.copy()
35
36 """
37 #Exploratory data analysis:
38
39 #1.Getting to know the data
40 #2.Data preprocessing (Missing values)
41 #3.Cross tables and data visualization
42 """
43 # *****
44 # Getting to know the data
45 # *****
46 #*** To check variables' data type
47 print(data.info())
48
```

On the right, a 'Variable Explorer' window displays the data frame structure:

| Name        | Type      | Size        | Value                 |
|-------------|-----------|-------------|-----------------------|
| data        | DataFrame | (31978, 13) | Column names: age,... |
| data_income | DataFrame | (31978, 13) | Column names: age,... |

Below the variable explorer, a 'Console' window shows the output of the `data.info()` command:

|  | maritalstatus         | occupation            | relationship          | race                  | gender                | capitalgain          | capitalloss          | hoursperweek         | nativecountry         | SalStat               | dtypes              |
|--|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|----------------------|----------------------|----------------------|-----------------------|-----------------------|---------------------|
|  | 31978 non-null object | 31978 non-null int64 | 31978 non-null int64 | 31978 non-null int64 | 31978 non-null object | 31978 non-null object | int64(4), object(9) |

At the bottom, the status bar indicates: Permissions RW End-of-line CRLF Encoding UTF-8 Line 48 Column 1 Memory 42%

So, the data underscore income is of data frame and the size of the data frame is 31978 observations with 13 columns. After reading data, we are going to create a copy of the original data, so that original data frame will not be touched. And further analysis will be made on the copy data frame. So, we are going to create a copy using ‘.copy’ function and we are storing it on to a data frame called data new data frame now. So, now, we have read the data.

Now, it is time to explore the data to understand the data even more better. So, under exploratory data analysis, we are going to broadly look into three topics the first one is getting to know the data, where we basically see what type of; what type of variables that we are going to deal with. The next thing that we are going to see here is data pre processing, where are we going to deal with missing values like how to identify the missing values and how are we going to deal with that.

And after that we are going to understand the data through visualization and cross tables, because we will be looking into checking the relationship between the variables using cross tables and data visualization. So, basically once we write the data we would be interested in getting the data type of each variable, so that it gives an idea about what type of data that we are going to deal with. So, ‘.info’ command on a data frame gives you the data type of each variables.

But if you check here all the variables are read with expected data type, because age is read as integer, JobType, from JobType to gender, it has been read as object because those have only categories to with and capitalloss, capitalgain, hoursperweek have been read as integer, because those variables have numerical variables. And the SalStat nativecountry has been read with object data type, because we know that their exist categories under that variable. So, it has been read as object.

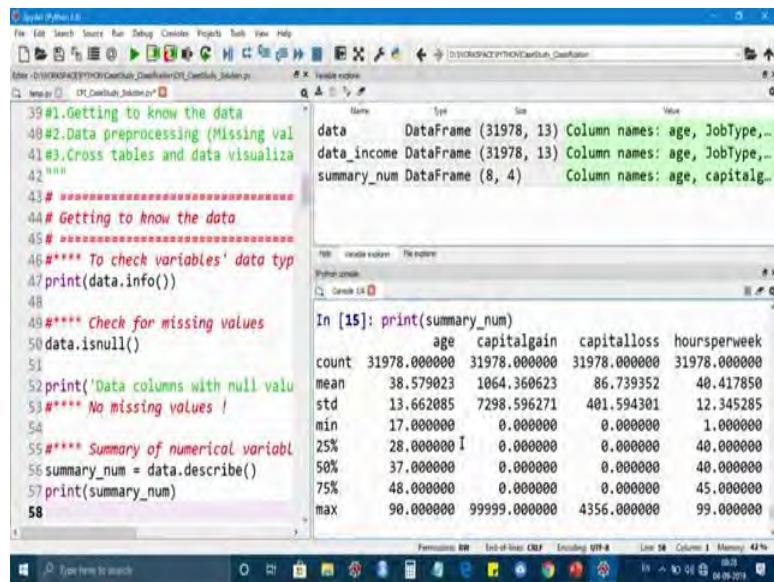
So, now, we have seen what are the data types of each variables and we know that all the variables are read with the proper unexpected data type. So, now, we will check whether there are any missing values in the data frame using a isnull function. So, let us check whether there are any missing values. So, basically a isnull returns Boolean values that is true and false true indicates the missing data and the false indicates there are no missing data in a particular cell.

But if you see the output, it will be difficult to skim through the whole output. So, let us take the sum of missing values in each column, so that we will get an idea about how many values or how many cells are missing under each variable. So, this can be done by adding dot sum to the existing command. So, let us see how to do that. So, this is just the print statement, but the actual command is here data.isnull.sum.

So, if we execute this line, you will get an output which says no columns have missing values, because the corresponding values of each variable are 0. So, it represent that there are no missing values under any of the variables. So, now we have checked whether there any missing values in a data frame or not. Now, it is time to understand the data by getting the descriptive statistics out of it.

In our data frame, both numerical and categorical variables are there. So, the descriptive statistic can give you great insights into the shape of each variable, so that describe function can be used to get the basic descriptive statistics out of data. So, I am going to use the dot described function on a data frame data. And I am storing that output to an object call summary\_num, because by default the describe function gives you eight statistical properties of numerical variables.

(Refer Slide Time: 09:47)



The screenshot shows a Jupyter Notebook interface with several code cells and their corresponding outputs.

Code cells (39-58):

```
39 #1.Getting to know the data
40 #2.Data preprocessing (Missing val
41 #3.Cross tables and data visualiza
42 """
43 # *****
44 # Getting to know the data
45 #
46 #**** To check variables' data typ
47 print(data.info())
48
49 #**** Check for missing values
50 data.isnull()
51
52 print('Data columns with null valu
53 #**** No missing values !
54
55 #**** Summary of numerical variabl
56 summary_num = data.describe()
57 print(summary_num)
58
```

Output:

|       | age          | capitalgain  | capitalloss  | hoursperweek |
|-------|--------------|--------------|--------------|--------------|
| count | 31978.000000 | 31978.000000 | 31978.000000 | 31978.000000 |
| mean  | 38.579023    | 1064.360623  | 86.739352    | 40.417850    |
| std   | 13.662085    | 7298.596271  | 401.594301   | 12.345285    |
| min   | 17.000000    | 0.000000     | 0.000000     | 1.000000     |
| 25%   | 28.000000    | 0.000000     | 0.000000     | 40.000000    |
| 50%   | 37.000000    | 0.000000     | 0.000000     | 40.000000    |
| 75%   | 48.000000    | 0.000000     | 0.000000     | 45.000000    |
| max   | 90.000000    | 99999.000000 | 4356.000000  | 99.000000    |

The first one being count mean standard deviation minimum 25 percent, 50 percent, 75 percent and the maximum value. Basically the count their count represent the count of observations under other particular variable that is age. When we look at the mean of age, the average age of the individual is turned out to be 39 years. And the standard deviation is around 14 and the minimum age of the individual is 17 years; the next is 25 percent that is 25 percent of the individuals age is less than 28 years.

The next is 50 percent; the median age is 37 years here. Next is 75 percent, 75 percent of the individual age is less than 48 years, at last you have maximum value we know that at the max the age of the individual is 90 years. Similarly, if you look at the capitalgain, it is a profit from the sale of property or an investment, so whether the sale price exceeds the purchase price.

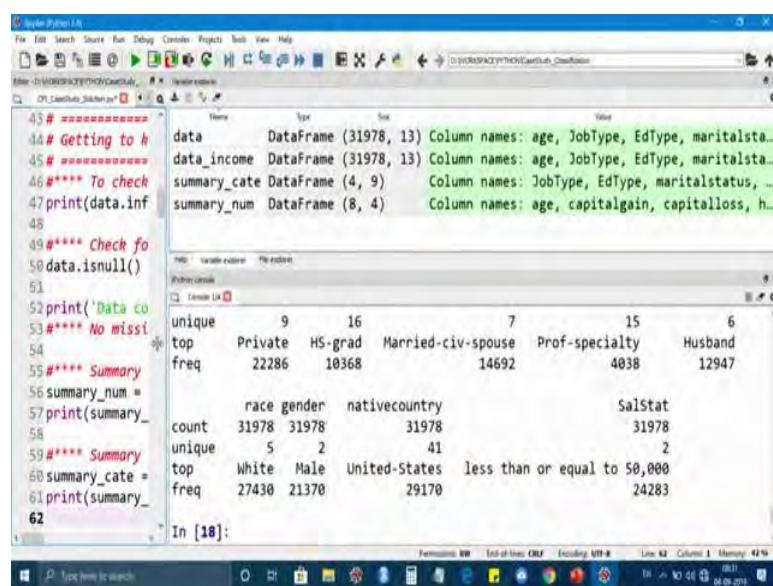
So, if you look at the capitalgain, it is a profit from the sale of property or an investment, where the sale price exceeds the purchase price then only we call it as a capitalgain. If you see on an average the capitalgain of the individual is 1069 and the standard deviation is really high that is 7298. And if you look at the maximum value, it is 99999, but the minimum value is 0.

So, in this case, it turned out that only 25 percentage of the capitalgain is greater than 0 that is why you have nothing under from minimum to 75 percent age, because it is very evident that very few people will be investing in stocks or any other investment, so that

they get some profit out of it. So, conversely a capitalloss arises, if the proceeds from the sale of a capital or a asset are less than the purchase price, it is turned out that only 25 percent of the capitalloss is greater than 0, though the maximum values 4356, the minimum value is still 0.

So, similarly you can look at the spread of the hoursperweek, like on an average individual spends 40 hoursperweek in a company. Similarly, you can look at the other statistics. And we also have categorical variables in our data frame. On that note, we would be interested and getting the frequencies of each categories under a variable. So, the same command is being used by setting include is equal to o, o represents object. So, let us see what the output gives us. And we are storing that onto a object call summary\_cate that represents summary for categorical variables.

(Refer Slide Time: 12:59)



The screenshot shows a Jupyter Notebook interface with the following code and its output:

```

43 # *****
44 # Getting to know the data
45 # *****
46 #**** To check
47 print(data.info)
48
49 #**** Check for missing values
50 data.isnull()
51
52 print('Data contains', data.isnull().sum().sum(), 'missing values')
53 #**** No missing values
54
55 #**** Summary
56 summary_num = pd.DataFrame.describe(data)
57 print(summary_num)
58
59 #**** Summary for categorical variables
60 summary_cate = pd.DataFrame.describe(data, include='object')
61 print(summary_cate)
62

```

The output shows the data types and counts for numerical columns, and then provides detailed summaries for categorical columns:

|      | unique  | 9     | 16      | 7     | 15                 | 6     |                |      |         |       |
|------|---------|-------|---------|-------|--------------------|-------|----------------|------|---------|-------|
| top  | Private | 22286 | HS-grad | 10368 | Married-civ-spouse | 14692 | Prof-specialty | 4038 | Husband | 12947 |
| freq |         |       |         |       |                    |       |                |      |         |       |

|        | race  | gender | nativecountry | SalStat                      |
|--------|-------|--------|---------------|------------------------------|
| count  | 31978 | 31978  | 31978         | 31978                        |
| unique | 5     | 2      | 41            | 2                            |
| top    | White | Male   | United-States | less than or equal to 50,000 |
| freq   | 27430 | 21370  | 29170         | 24283                        |

So, basically when you set when you set include is equal to o, it gives you four measures starting from count, count basically gives you how many observations are considered while giving you the summary and then unique, unique represents, how many unique categories are available under that particular variable. For example, under JobType, we have nine unique categories. And next is the top which gives you the model value of the variable that is the most frequently occurring category.

So, for JobType the most frequently occurring category is private. And you can also get the corresponding frequency. Here 22286 observations correspond to private category.

Similarly, you can look at the frequencies of the categories that are available under other variables. But from this output, we now just know how many unique values are there, but we would be interested in getting the unique categories under a variable, so that it gives us an idea what are the categories that are there in a particular variable.

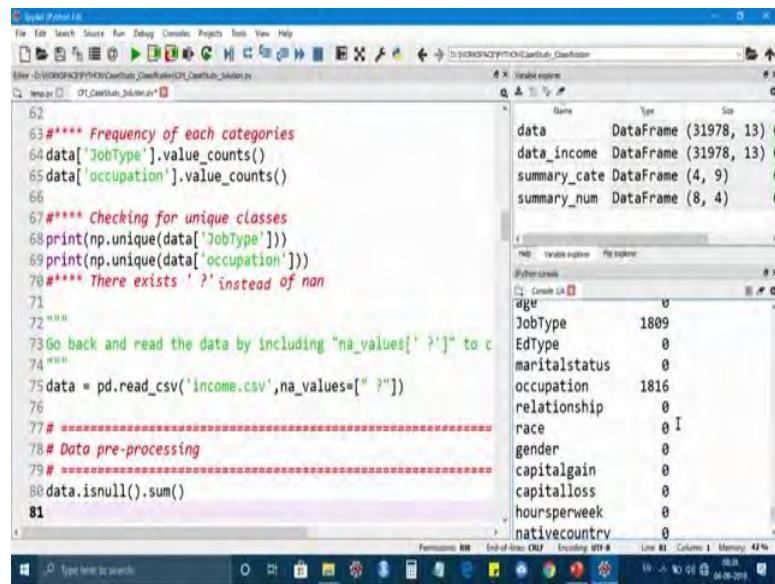
So, for that case, in that case, the value\_counts gives us the better representation. So, let us see how to use the value underscore function. So, I am just going to take the first variable that is JobType. I am going to get the frequencies of each categories that are available under JobType. So, basically this gives us the better representation by listing all the available categories under frequencies. If you look at here, it basically gives you the frequencies of each categories under JobType. You can also see there exist question mark instead of nun.

So, there exist a question mark instead of nan, because by default python only reads the blank cells as nan not other special characters. In this case, we have a special character, but it is not being read as nan, but we have encounter that there are some missing values in the form of question marks. Similarly, we can also get the frequencies of categories using the value\_counts for other variables, but it turned out that occupation also has question marks here. But when we looked at the data, there were only question marks under two variables that is occupation and JobType.

So, to basically see how exactly the special characters are, we can use the unique function. So, let us check the unique classes of JobType. So, basically unique function is from numpy library and inside the function as specify the JobType from data frame data. So, if you see that there exists the special character which is question mark there is the space before the question mark, this is how exactly the levels of the JobType are being entered.

So, if you see here before the starting letter of the every level, there is a space in front of it. So, similarly we have a space in front of question marks also, this is how exactly the special character is been entered here. So, now, we know that there are some special characters which is just the representation of missing values. So, what we are going to do here is, we are going to go back and read the data by including the na values to consider the special character as nan. By adding na\_values with a list of a string values we are going to consider this special character as nan values.

(Refer Slide Time: 16:59)



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for data analysis. The code includes comments for frequency counts, unique classes, and handling missing values ('?'). It reads a CSV file named 'income.csv' with 'na\_values' set to '?'. A summary DataFrame is then created. The right side of the interface shows the variable explorer with a list of variables and their types, and the output pane showing the summary DataFrame.

```
62
63 ##### Frequency of each categories
64 data['JobType'].value_counts()
65 data['occupation'].value_counts()
66
67 ##### Checking for unique classes
68 print(np.unique(data['JobType']))
69 print(np.unique(data['occupation']))
70 ##### There exists ' ?' instead of nan
71
72 """
73 Go back and read the data by including "na_values[' ?']" to c
74 """
75 data = pd.read_csv('income.csv',na_values=[' ?'])
76
77 # **** Data pre-processing
78 # ****
79 # ****
80 data.isnull().sum()
81
```

| Name         | Type      | Size        |
|--------------|-----------|-------------|
| data         | DataFrame | (31978, 13) |
| data_income  | DataFrame | (31978, 13) |
| summary_cate | DataFrame | (4, 9)      |
| summary_num  | DataFrame | (8, 4)      |

| age           | ?    |
|---------------|------|
| JobType       | 1809 |
| EdType        | 0    |
| maritalstatus | 0    |
| occupation    | 1816 |
| relationship  | 0    |
| race          | 0 I  |
| gender        | 0    |
| capitalgain   | 0    |
| capitalloss   | 0    |
| hoursperweek  | 0    |
| nativecountry | 0    |

So, once we have executed this data, you will be able to see the data frame size is 31978 observation with 13 variables. Now, we have seen how to consider other special characters as nan values. Now, it is time to pre process the data. Basically now we know that there are some missing values in a data frame to deal with the missing data, we need to first identify the missing data and then we can examine the missing value pattern.

So, by using the same isnull function, we found out that 1809 cells are missing under JobType and 1816 cells are missing under occupation and no other variables have missing values except JobType and occupation. So, before deciding on how to deal with the missing data, let us see in a particular row either one of the column is missing or both the column values are missing. For that let us subset the rows at least one column is missing in a row.

So, here we going to subset the rows by giving .any\_axis is equal to 1, so that it considered at least one missing column in a particular row. Now, we have subset the rows with missing values. If you look at the size of the data frame, it is 1816. It is with 1816 rows with 13 variables.

(Refer Slide Time: 18:32)

| Index | age | inh-type | ed-type   | maritalstatus | occupation | relationship | sex    | gender | capitalgain | capitalloss | hoursperweek | nativecountry |
|-------|-----|----------|-----------|---------------|------------|--------------|--------|--------|-------------|-------------|--------------|---------------|
| 8     | 17  | nan      | 11th      | Never_ nan    | Own-c_     | White        | Female | 0      | 0           | 5           |              | Unite...      |
| 17    | 32  | nan      | Some-...  | Marri_ ...    | Husba...   | White        | Male   | 0      | 0           | 40          |              | Unite...      |
| 29    | 22  | nan      | Some-...  | Never_ ...    | Own-c...   | White        | Male   | 0      | 0           | 40          |              | Unite...      |
| 42    | 52  | nan      | 12th      | Never_ ...    | Other_ ... | Black        | Male   | 594    | 0           | 40          |              | Unite...      |
| 44    | 63  | nan      | 1st-4...  | Marri_ ...    | Husba...   | White        | Male   | 0      | 0           | 35          |              | Unite...      |
| 57    | 72  | nan      | HS-gr...  | Marri_ ...    | Husba...   | White        | Male   | 0      | 0           | 20          |              | Unite...      |
| 69    | 53  | nan      | 5th-6...  | Widow_ ...    | Unmar...   | Black        | Female | 0      | 0           | 30          |              | Unite...      |
| 73    | 57  | nan      | Assoc-... | Widow_ ...    | Unmar...   | White        | Female | 0      | 0           | 38          |              | Unite...      |
| 75    | 28  | nan      | Some-...  | Never_ ...    | Own-c...   | White        | Male   | 0      | 0           | 24          |              | Unite...      |
| 76    | 21  | nan      | Some-...  | Never_ ...    | Unmar...   | White        | Female | 0      | 0           | 35          |              | Unite...      |
| 97    | 34  | nan      | HS-gr...  | Never_ ...    | Unmar...   | Black        | Female | 0      | 0           | 40          |              | Unite...      |
| 133   | 18  | nan      | 12th      | Never_ ...    | Own-c...   | White        | Male   | 0      | 0           | 25          |              | Unite...      |
| 137   | 65  | nan      | Some-...  | Marri_ ...    | Husba...   | White        | Male   | 0      | 0           | 38          |              | Unite...      |
| 147   | 42  | nan      | HS-gr...  | Marri_ ...    | Husba...   | White        | Male   | 0      | 0           | 40          |              | Unite...      |
| 148   | 55  | nan      | HS-gr...  | Marri_ ...    | Wife       | Asian...     | Female | 0      | 0           | 40          |              | Unite...      |
| 153   | 23  | nan      | Some-...  | Never_ ...    | Unmar...   | Amer...      | Female | 0      | 0           | 25          |              | Unite...      |
| 205   | 58  | nan      | HS-gr...  | Marri_ ...    | Husba...   | White        | Male   | 0      | 0           | 50          |              | Unite...      |
| 213   | 78  | nan      | 9th       | Widow_ ...    | Unmar...   | White        | Female | 1111   | 0           | 15          |              | Unite...      |
| 225   | 28  | nan      | 11th      | Marri_ ...    | Own-c...   | Asian...     | Female | 0      | 1762        | 40          |              | South         |
| 228   | 17  | nan      | 11th      | Never_ ...    | Own-c...   | White        | Female | 0      | 0           | 20          |              | Unite...      |

So, now let us open the missing data frame and inspect the missing data. So, it is very clear that whenever JobType is missing, the occupation is also missing, but the total number of missing values is 1816 rows. So, there is a special category called never worked and the corresponding value of the occupation is nan. Since the JobType is never worked.

So, if you saw the JobType, so if you saw the JobType there is a category called never worked. If the individual has never worked, then occupation could not be filled that is the reason you have nan values under occupation. So, now let us go back to the script in window and see and what we are going to do with it.

(Refer Slide Time: 19:27)

The screenshot shows a Jupyter Notebook interface with several code cells and output cells. The code is as follows:

```
81
82
83
84 missing = data[data.isnull().any(axis=1)]
85 # axis=1 => to consider at least one column value is missing
86
87 """ Points to note:
88 1. Missing values in Jobtype = 1809
89 2. Missing values in Occupation = 1816
90 3. There are 1809 rows where two specific
91 columns i.e. occupation & Jobtype have missing values
92 4. (1816-1809) = 7 => You still have occupation unfilled for
93 these 7 rows. Because, jobtype is Never worked
94 """
95
96 data2 = data.dropna(axis=0)
97
98 # Relationship between independent variables
99 correlation = data2.corr()
100
```

The output cells show the results of the last three code cells:

```
In [27]: missing =
data[data.isnull().any(axis=1)]
```

```
In [28]: data2 =
data.dropna(axis=0)
```

```
In [29]: correlation =
data2.corr()
```

```
In [30]:
```

So, let us quickly note some points that we have got from the missing data. So, missing values in JobType is equal 1809 rows. So, there are 1816 rows that are missing under occupation and there are 1809 rows where two specific column that is occupation and JobType have missing values. So, there is the difference between 1816 and 1809 that is 7 rows. You still have occupation and fill for those seven rows because JobType is never worked that is why the total number of missing values is 1816.

Now, in this case we can delete the cases containing the missing data or replace the missing values with reasonable alternative data values. If the data are not missing at random, we have to model the mechanisms that produce missing values as well as the relationship of interest which is very complex in these contexts. So, here we are going to remove all the rows with missing values in consider only the complete cases alone. So, in that case, we are going to use a command called dropna. And by setting axis is equal to 0, we are dropping out all the rows wherever there are missing values.

So, if you look at the size of the data to data frame that is 30162 rows with 13 variables; that means, that we have gotten read of 1816 rows wherever they were missing values, why because we could not find out the mechanism that goes behind which produce the missing values and we do not know the relationship of interest as well. So, we are going in this session, we are going to remove all the missing values and we are going to continue with the further analysis.

So, with complete cases let us look at the relationship between numerical variables that is between independent variables using the correlation measure. So, you can get the correlation, you can get the pair wise correlation using the .corr function using the corr function. So, and I save my output to a variable called correlation onto an object check called correlation.

(Refer Slide Time: 21:34)

| Index      | age    | capita_hh  | capita_per | hoursperwk |
|------------|--------|------------|------------|------------|
| age        | 1      | 0.0801542  | 0.0601655  | 0.101599   |
| capita_hh  | 0.0801 | 1          | -0.0322293 | 0.0004318  |
| capita_per | 0.0601 | -0.0322293 | 1          | 0.052417   |
| hoursperwk | 0.1015 | -0.0804318 | 0.052417   | 1          |

So, let us look at the correlation values. So, if you look at the correlation values, none of the variables correlation, so none of the values are nearer to 1; most of the values are nearer to 0. It represents that none of the variables are correlated with each other because the correlation values lies from -1 to +1. And if it is closer to 1, we say that there is a strong relationship between two variables; and if it is closer to 0, then we say that there is a there are no a little correlation between variables.

So, in our case, none of the variables are correlated with each other. So, now, we will consider the categories categorical variables to look at the relationship offered.

(Refer Slide Time: 22:36)

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The code uses the pd.crosstab function to create a gender proportion table and a gender vs salary status table. The output pane displays the resulting DataFrames.

```
Extracting the column names
data2.columns
Gender proportion table:
gender = pd.crosstab(index = data2["gender"],
 columns = 'count',
 normalize = True)
print(gender)
Gender vs Salary Status:
gender_salstat = pd.crosstab(index = data2["gender"],
 columns = data2['SalStat'],
 margins = True,
 normalize = 'index') # Include row totals
print(gender_salstat)
```

| SalStat | greater than 50,000 | less than or equal to 50,000 |
|---------|---------------------|------------------------------|
| Female  | 0.113678            | 0.886322                     |
| Male    | 0.313837            | 0.686163                     |
| All     | 0.248922            | 0.751078                     |

So, now we will look at the gender proportion using the cross table function. So, here I am accessing the columns of data2 basically it gives you the column names of each variable, so that you can use the column names in the further analysis. So, here we are going to look at the gender proportion using the cross table function. And by setting normalise=true, you will get the proportion table.

And you will give the variable of interest under the index column. And let us look at the output. Now, if you see here male are high in frequency that is 67 percent corresponds to male and only 33 percent corresponds to female. So, now we have got an idea about what is the proportion of gender that we have. Now, it is time to check how salary status varies across the gender.

So, we are going to look at the two-way table where we are going to check the relationship between gender and salary status. And in a row I want gender, and in columns I want; and in rows I want gender, and in columns I want salary status. And by setting normalised is equal to index, I am going to get the row proportion equals to 1.

(Refer Slide Time: 23:57)

The screenshot shows a Jupyter Notebook interface with the following content:

```
184 # Extracting the column name
185 data2.columns
186
187 # **** Gender proportion table:
188 # **** Gender vs Salary Status:
189 #
190
191 gender = pd.crosstab(index
192 columns='normali
193 print(gender)
194 #
195 # **** Gender vs Salary Status:
196 #
197 gender_salstat = pd.crosstab(
198
199
200 print(gender_salstat)
201
202 #
203
```

In [33]:

|   | name            | type                  | size                                                            | value |
|---|-----------------|-----------------------|-----------------------------------------------------------------|-------|
| 1 | correlation     | DataFrame (4, 4)      | Column names: age, capitalgai...                                |       |
| 2 | data            | DataFrame (31978, 13) | Column names: age, JobType, E...                                |       |
| 3 | data2           | DataFrame (30162, 13) | Column names: age, JobType, E...                                |       |
| 4 | data_income     | DataFrame (31978, 13) | Column names: age, JobType, E...                                |       |
| 5 | gender          | DataFrame (2, 1)      | Column names: count                                             |       |
| 6 | gender_csalstat | DataFrame (3, 2)      | Column names: greater than 50,000, less than or equal to 50,000 |       |

...  
Include row and column totals  
...: print(gender\_salstat)  
...:  
Salstat greater than 50,000 less than or equal to 50,000  
gender  
Female 0.113678 0.886322 I  
Male 0.313837 0.686163 II  
All 0.248922 0.751078 III

So, from the output, it is very clear that only 11 percent of the female earn greater than 50000 US dollars and 89 percentage earn less than 50000 US dollars, but men are more likely to earn more than when compared with women. On classification problems, you need to know how balance the class values are. For example, you need to know the proportion of your output variable. So, let us visually look at the frequency distribution of the salary status using the count plot function from seaborn and the variable of interest is SalStat, because we are going to look at the bai plot of salary status.

(Refer Slide Time: 24:40)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - D:\WORKSPACE\Python\CaseStudies\CR\_Credit\_Card\_Sklearn.py

File Explorer Variables Explorer

```
109 # *****
110 gender = pd.crosstab(index = data2["gend",
111 columns = 'count',
112 normalize = True)
113 print(gender)
114 # *****
115 # Gender vs Salary Status:
116 # *****
117 gender_salstat = pd.crosstab(index = da,
118 columns = da,
119 margins = Tr,
120 normalize = 'i'
121 print(gender_salstat)
122
123 # *****
124 # Frequency distribution of 'Salary status'
125 # *****
126 SalStat = sns.countplot(data2['SalStat'])
127 """
128 75 % of people's salary status is <=50,
```

correlation DataFrame (4, 4) Column names:...

data DataFrame (31978, 13) Column names:...

data2 DataFrame (30162, 13) Column names:...

data\_income DataFrame (31978, 13) Column names:...

gender DataFrame (2, 1) Column names:...

gender\_salstat DataFrame (3, 2) Column names:...

Help Variables Explorer File Explorer

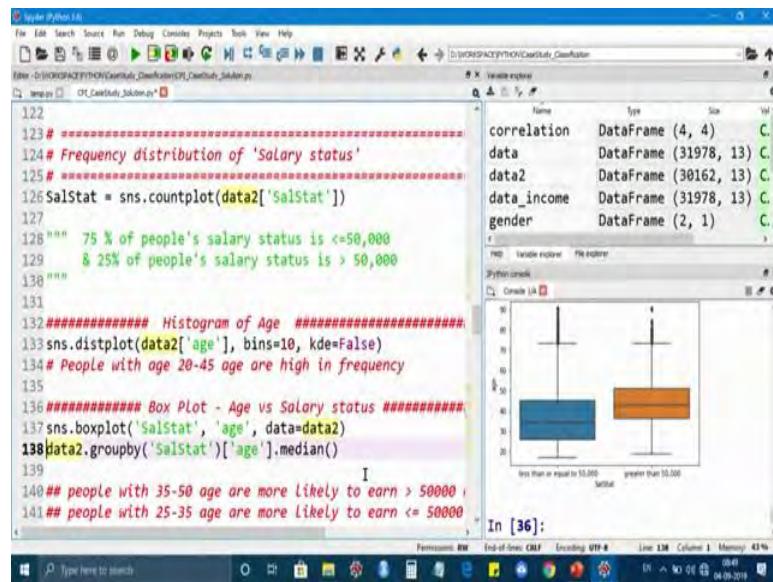
Create (A)

In [34]:

It is very clear that around 75 percentage of the observation corresponds to less than or equal to 50000 and only 25 percent corresponds to greater than 50000. So, now, let us plot the histogram to check the underlined frequency distribution of the age variable. So, I am going to plot histogram using this plot from the sea born library and for the age variable. And by setting bins is equal to 10, I will have only ten bins to interpret from and I am setting kde is equal to false, so that I will have the frequency is on the y-axis.

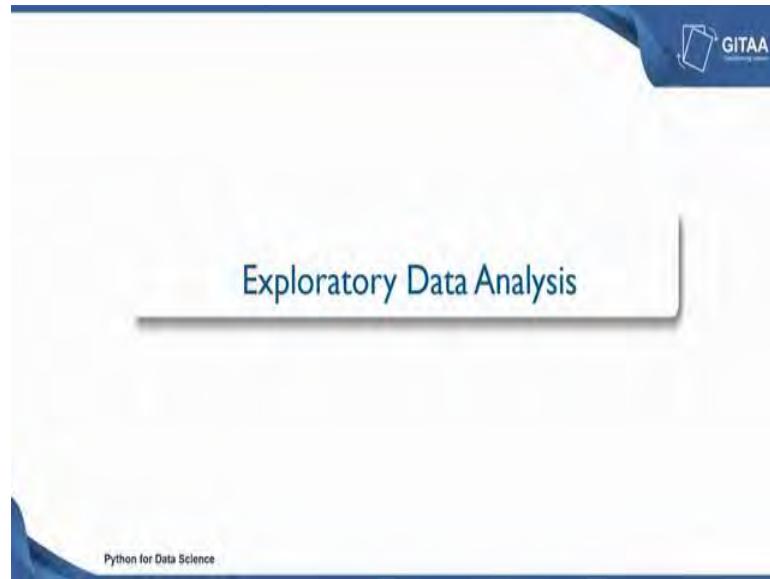
So, from the histogram, it is very clear that people with age 20 to 45 are high in frequency and we also have the records for the other ages, but between 20 to 45, the frequency is high. Similarly, we can generate cross tables and plots to understand the relationship between other variables. So, if you want to do a bivariate analysis to check how age is affecting the salary status, then we can do a box plot to check how salary is wearing across age.

(Refer Slide Time: 26:05)



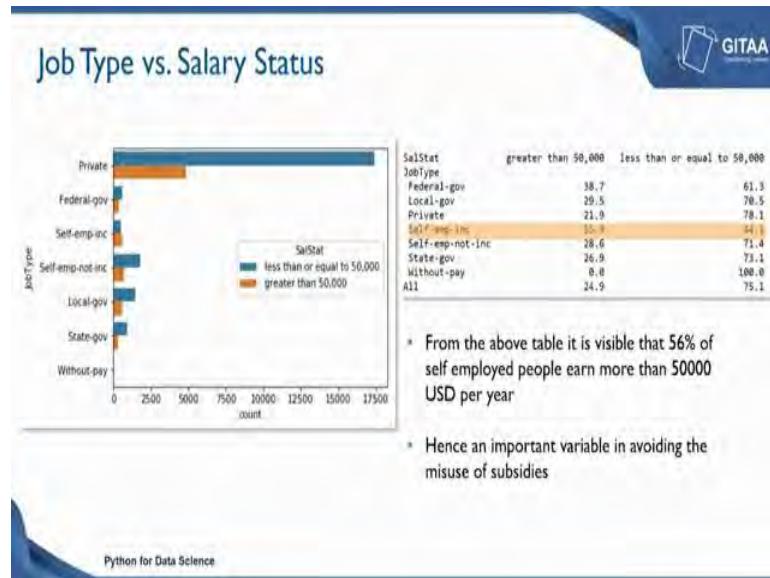
So, people with 32, so people with 35 to 50 age or more likely to earn more than 50000 US dollars, but people with 25 to 35 age are more likely to earn less than 50000 dollars. So, similarly you can generate the cross tables and plots to understand the relationship between other variables.

(Refer Slide Time: 26:27)



Using python we will look at the relationship between the variables using cross tables and a visualization. So, now, I am going to quickly take you through the exploratory data analysis through slides, where I am going to show you the relationship between variables using the same commands that we have used in the python.

(Refer Slide Time: 26:50)



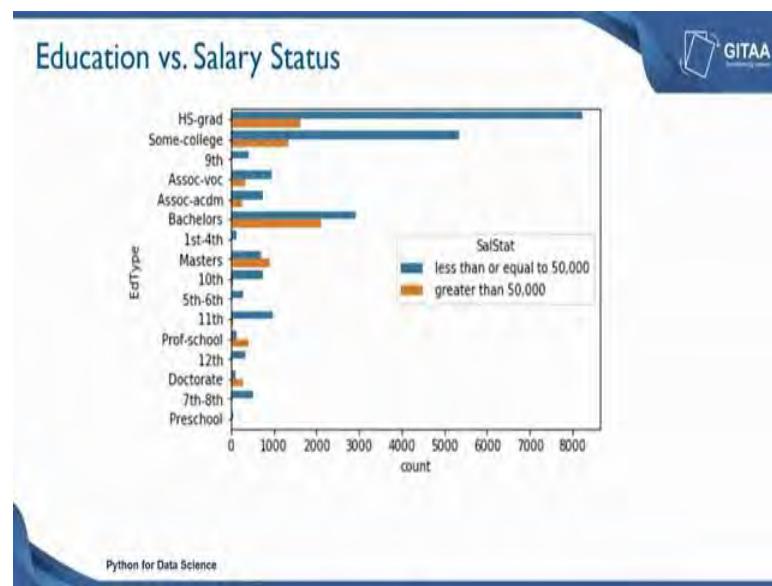
So, first we are going to look at the JobType versus salary status. So, first we are going to look at the relationship between JobType and SalStat. So, here is the bar plot which gives you an idea about what is the count, what is the frequency of each category it is

available under a JobType. So, for example, if you see here most of the individuals work in a private organisation and very few work in the state government or federal government organisation.

This is the frequency distribution of the JobType. But if you want to see how salary is wearing across the JobType, then we can look at the cross table. And I have highlighted only a single row which will give you an idea about what is the percentage splits for an individual. If he is if he is self employed then what is the percentage spilt that, he will earn greater than 50000 or less than or equal to 50000. For the greater than 50000 it turned out to be 56 percent and for the less than or equal to 50000 it is 44 percent.

And for the rest of the JobType more or less 70 more or less 70 percentage of the individuals earned less than or equal to 50 and the rest of the percentage belongs to greater than 50. Only for the self employed people there is an equal proportion more or less equal proportion for both the categories. And on the whole from the above table, it is visible that one whole if you look at the JobType, it will be an important variable in avoiding the misuse of subsidies.

(Refer Slide Time: 28:23)



So, next we will look at a variable call education. Let us just look at the frequency distribution of education. Education basically has too many levels to it starting from preschool to doctorate. And if you look at the frequencies of each categories only the

higher secondary graduation has the highest frequency. People most of the people have finished the higher secondary graduation, and followed by some college.

And if you see here the blue bar is the representative of less than or equal to 50 and greater than 50. So, now let us look at the relationship that exists between the salary status in education to see how the salary state is varying across the education type.

(Refer Slide Time: 29:10)

The slide features a title 'Education vs. Salary Status' and a logo for 'GITAA'. Below the title is a cross-table with two columns: 'greater than 50,000' and 'less than or equal to 50,000'. The rows represent different education levels. The table shows that higher education levels are associated with higher salaries. The last row, 'Some-college', includes a note about Python for Data Science.

| SalStat      | greater than 50,000 | less than or equal to 50,000 |
|--------------|---------------------|------------------------------|
| EdType       |                     |                              |
| 10th         | 7.2                 | 92.8                         |
| 11th         | 5.6                 | 94.4                         |
| 12th         | 7.7                 | 92.3                         |
| 1st-4th      | 4.0                 | 96.0                         |
| 5th-6th      | 4.2                 | 95.8                         |
| 7th-8th      | 6.3                 | 93.7                         |
| 9th          | 5.5                 | 94.5                         |
| Assoc-acdm   | 25.4                | 74.6                         |
| Assoc-voc    | 26.3                | 73.7                         |
| Bachelors    | 42.1                | 57.9                         |
| Doctorate    | 74.7                | 25.3                         |
| HS-grad      | 16.4                | 83.6                         |
| Masters      | 56.4                | 43.6                         |
| Preschool    | 0.0                 | 100.0                        |
| Prof-school  | 74.9                | 25.1                         |
| Some-college | 28.0                | 80.0                         |

From the above table we can see that people who have done Doctorate, Masters , Prof-school are more likely to earn above 50000 USD per year when compared with others. Hence an influencing variable in avoiding the misuse of subsidies.

Python for Data Science

So, we are going to use the cross table for that. So, from the table it is, so from the table we can see that people who have done doctorate, masters and professional school are more likely to earn above 50000 US dollars per year when compared with the other education type. So, that it and hence it can be an influencing variable in avoiding the misuse of the subsidies.

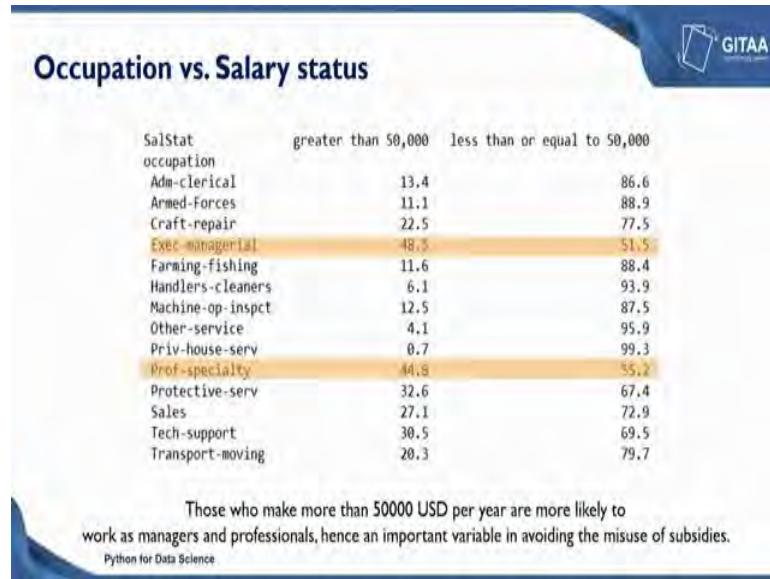
(Refer Slide Time: 29:36)



Next we will look at a variable called occupation. This is the bar diagram of the occupation variable. And if you look at the categories under the occupation, it has so many to it starting from administrative, clerical and it has so many categories like armed forces, farm, fishing, technical support and other services.

If you try to interpret how salary status is dependent on occupation, then there is a clear demarcation because there is no equal length of bars for both the colours like for less than or equal to 50 or greater than 50 for each and every level of the occupation the salary status, the proportion or the frequency of salary status is differing. So, let us get an a clear idea using the cross table.

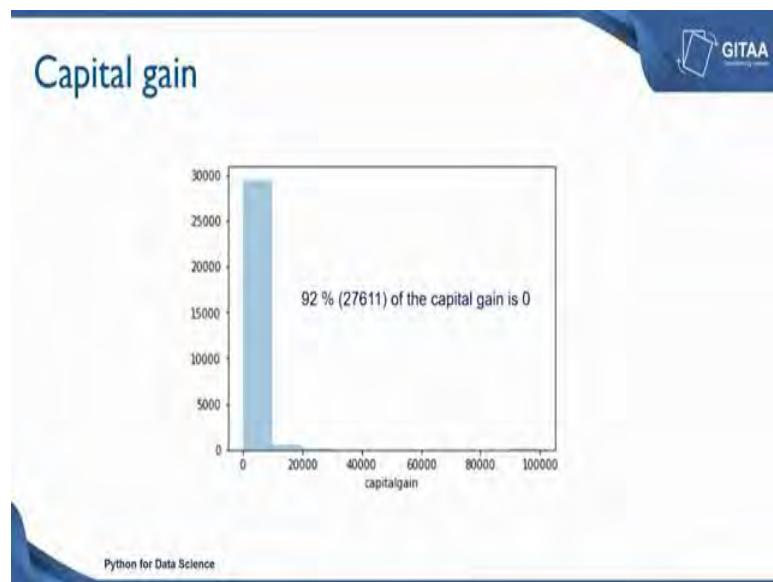
(Refer Slide Time: 30:21)



So, here is the cross table for occupation versus salary, so that we will get an idea whether the salary status is dependent on occupation or not. So, from the output, I have highlighted two levels here, one is executive managerial and the other one is. So, I have highlighted two levels; one is executive managerial and the other is prof specialty.

And those who make more than 50000 US dollars per year are more likely to work as a managers and professionals. Hence it can be an important variable in avoiding the misuse of subsidies. So, let us see how this variable is impacting the salary status when we build the model.

(Refer Slide Time: 31:06)

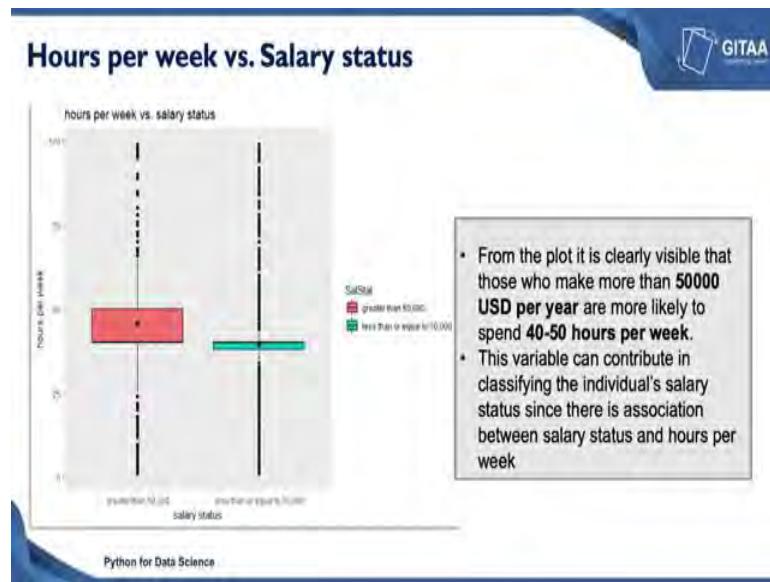


Capitalgain is one of the variable that we have from the data frame which is an important variable. Because if the capitalgain of the individual is high, then the salary status could be high. So, this could be one of the important variables to classify the individuals salary status and the plot shows the frequency distribution of the capitalgain variable.

Though the capitalgain is ranging from 0 to 1 lakh, though the capitalgain is ranging from 0 to 1 lakh, but there are more observations in between the bins 0 to 20000, so that means, 92 percentage of the capitalgain is 0 which corresponds to 27611 observation, and only 8 percentage of the people have gained something from selling their asset or a gain profit out of their investment. So, this could be one of the important variable to classify the individual salary status.

So, let us see whether capitalgain is an important variable when we build the model capitalloss values are also ranging from 0 to 4000, but 95 percentage of the capitalloss is 0. So, in our records either we have the records for capital gain or loss because either they will loosed or gain from the investment. And if you see 28721 individual's capitalloss is 0, so either they would have not invested or the capitalloss is 0 for them or the capitalloss is still 0 that could be the reason because they have not invested or they have not loss anything from their investment. So, let us see whether this will be an important variable when we try to; when we try to include this in a model.

(Refer Slide Time: 32:58)



The next one is the hours spent variable. So, the plot shows the relationship between the salary status and hour spent using the box plot. On the x-axis, I have the salary status; on the y-axis I have the hours per. And on the x-axis the salary status is represented as greater than 50 and less than or equal to 50. And the plot shows the relationship between that the relationship that exists between the hours spend and the salary status.

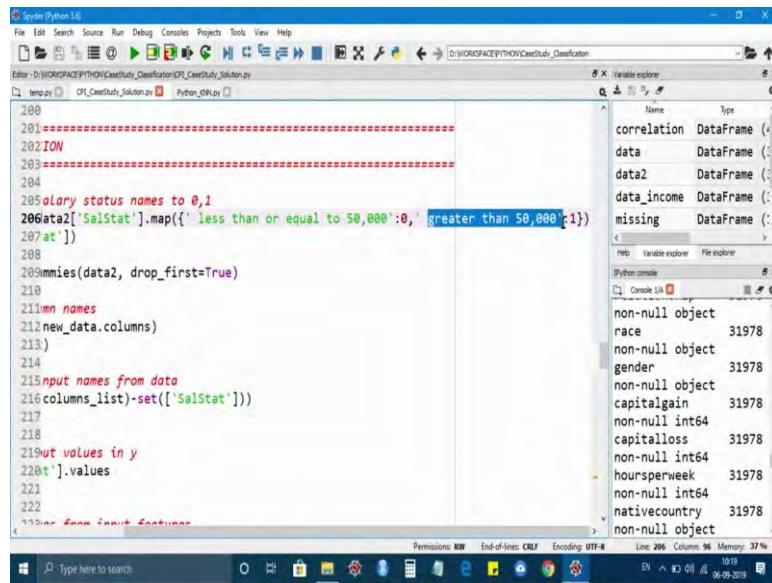
Here we are going to check how the salary status is vary with respect to the hour spend by the individual in a company. And if you see here from the plot it is clearly visible that those who make more than 50000 US dollars per year or more likely to spend between 40 to 50 hoursperweek on an average. And others variable can contribute in classifying the individual's salary status since there is an association between salary status and hour per week.

Because when you consider the less than or equal to 50000 category, the median is very low when compare to the greater than 50000 category. And the minimum hours spent and the maximum hours spent by the individual is also very low that could also be the reason that is why they are earning less than 50000 US dollars per year. But all these interpretation that we have made are from the data that we have, it might decorate with respect to different sense of data.

**Python for Data Science**  
**Prof. Raghunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 24**  
**Case Study - Classification**

(Refer Slide Time: 00:16)



The screenshot shows the Spyder Python IDE interface. The code editor window displays the following Python script:

```
208
209=====
210ION
211=====
212
213salary status names to 0,1
214data2['SalStat'].map({' less than or equal to 50,000':0, ' greater than 50,000':1})
215
216dummies(data2, drop_first=True)
217
218mm names
219new_data.columns
220
221
222npnput names from data
223columns_list-set(['SalStat'])
224
225ut values in y
226t].values
227
228
229or from input features
```

The variable explorer on the right side of the interface lists the following variables:

| Name            | Type      | Count |
|-----------------|-----------|-------|
| correlation     | DataFrame | 1     |
| data            | DataFrame | 1     |
| data2           | DataFrame | 1     |
| data_income     | DataFrame | 1     |
| missing         | DataFrame | 1     |
| non-null object | object    | 31978 |
| race            | object    | 31978 |
| non-null object | object    | 31978 |
| gender          | object    | 31978 |
| non-null object | object    | 31978 |
| capitalgain     | int64     | 31978 |
| capitalloss     | int64     | 31978 |
| non-null int64  | int64     | 31978 |
| hoursperweek    | int64     | 31978 |
| nativecountry   | object    | 31978 |
| non-null object | object    | 31978 |

So, by exploring on the data we have looked at the distribution of each of the variables and the relationship that exists between the variables. So now, we are going to build a Logistic Regression Model, logistic regression is a machine learning classification algorithm that is used to predict the probability of a categorical dependent variable. So, using logistic regression we will build a classify model based on the available data. The first step that we are doing here is re indexing the salary status names to 0 and 1 because the machine learning algorithms cannot work with categorical data directly.

So, categorical data must be converted to numbers. So, we can assign 0 to less than or equal to 50000 and one to greater than 50000 using the map function and we are giving the values using the dictionary. Where we have mapped all the less than or equal to 50000 string values to 0 and greater than 50000 to an integer called 1, so let us run that.

(Refer Slide Time: 01:23)

The screenshot shows the Spyder Python IDE interface. The code editor displays a script named 'temp.py' containing Python code for logistic regression. A warning message from pandas is displayed in the status bar, cautioning against using .loc[row\_indexer,col\_indexer] = value instead of .at or .iat for indexing.

```
200
201 # =====
202 # LOGISTIC REGRESSION
203 # =====
204
205 # Reindexing the salary status
206 data2['SalStat']=data2['Sal'
207 print(data2['SalStat'])
208
209 new_data=pd.get_dummies(dat
210
211 # Storing the column names
212 columns_list=list(new_data.
213 print(columns_list)
214
215 # Separating the input name
216 features=list(set(columns_1
217 print(features)
218
219 # Storing the output values
220 y=new_data['SalStat'].values
221 print(y)
222
223 # Storing the values from input features
```

In [41]:

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-versus-copy>

So when you open the data to data frame you will be able to see the salary status values as zeros and ones. So, the method that we used here is integer encoding. So, that we can invert the encoding later and get labels back from integer values such as in the case of making a prediction and using the pandas function to get\_dummies. We can convert the categorical variables into dummy variables which is called as one hot encoding. It refers to splitting the column which has the categorical data to many columns depending on the number of categories present in the column.

So, let us see how to do that. So, we are using the get\_dummies function and inside the function we have specified the data frame name and I have stored the output to new data frame called new\_data.

(Refer Slide Time: 02:16)

The screenshot shows the Spyder Python IDE interface. On the left, there are three tabs: 'temp.py', 'CPI\_CaseStudy\_Solution.py', and 'Python JSON.py'. The 'temp.py' tab is active and contains the following code:

```
200
201 # =====
202 # LOGISTIC REGRESSION
203 # =====
204
205 # Reindexing the salary state
206 data2['SalStat']=data2['Sal']
207 print(data2['Salstat'])
208
209 new_data=pd.get_dummies(data2)
210
211 # Storing the column names
212 columns_list=list(new_data)
213 print(columns_list)
214
215 # Separating the input name
216 features=list(set(columns_1))
217 print(features)
218
219 # Storing the output values
220 y=new_data['SalStat'].values
221 print(y)
222
223 # Storing the values from input features
```

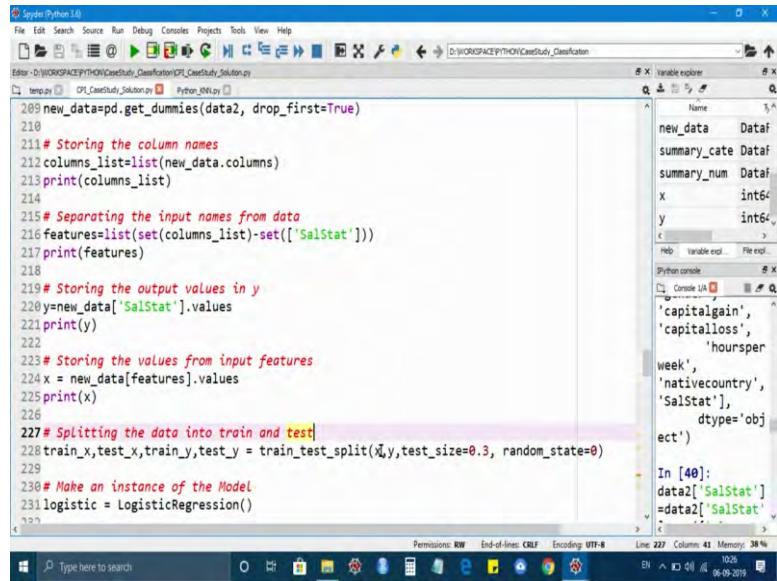
In the center, a 'new\_data : DataFrame' window displays a data frame with 14 rows and 7 columns. The columns are labeled: Index, type\_Local-gov, JobType\_Private, type\_Self-emp-no-job, type\_Self-emp-no-job\_type, State, and SalStat. The data consists of binary values (0 or 1) for most columns, except for 'State' which has categorical values like 'CA', 'TX', etc.

On the right, the 'Variable explorer' shows several variables: data\_income, missing, new\_data, summary\_cate, and summary\_num. The 'Console' tab shows the command: `In [41]: new_data=pd.get_dummies(data2, drop_first=True)`.

So, when we open the new\_data if you see here if you check the data frame each column contains 0 or 1. So, let us take the example of the JobType variable, JobType earlier had seven categories to it, so each of the categories have been splintered in to columns. So, that the column has only the value 0 and 1. For example, the first row sees it has one under the column called JobType\_private, then it means that the first row corresponds to the private JobType. Similarly you can interpret for the other values. So, now, we have mapped all the string values to integer values, so that we can work with any machine learning algorithms.

Next we are going to select the features where we need to divide the given columns into two types that is one having independent variables and one having the dependent variables .The dependent variable we are going to represent using letter called y and the independent variables being represented as x.

(Refer Slide Time: 03:22)



The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'temp.py' with the following content:

```
209 new_data=pd.get_dummies(data2, drop_first=True)
210
211 # Storing the column names
212 columns_list=list(new_data.columns)
213 print(columns_list)
214
215 # Separating the input names from data
216 features=list(set(columns_list)-set(['SalStat']))
217 print(features)
218
219 # Storing the output values in y
220 y=new_data['SalStat'].values
221 print(y)
222
223 # Storing the values from input features
224 x = new_data[features].values
225 print(x)
226
227 # Splitting the data into train and test
228 train_x,test_x,train_y,test_y = train_test_split(X,y,test_size=0.3, random_state=0)
229
230 # Make an instance of the Model
231 logistic = LogisticRegression()
```

The variable explorer window on the right shows the following variables:

- new\_data: Dataframe
- summary\_cate: Dataframe
- summary\_num: Dataframe
- x: int64
- y: int64
- features: list
- capitalgain: string
- capitalloss: string
- hoursperweek: float
- nativecountry: string
- SalStat: string

The status bar at the bottom indicates: Line 227 Column 41 Memory: 38 %.

For that we are going to store the column names as a list, by accessing the column names from the new\_data dataframe and we are storing it under a list called columns\_list. So, it will have only the column names from the new data dataframe. So, now, we need to separate the input variables from the data.

So, let us exclude the salary status variable form the columns\_list and store it as features. So, from the list columns list we are going to exclude only the variable salary status. So, that we will have only the independent variables and storing that output to variable called features. So, let us print features and see what it has. So, it basically has all the column names from the columns list, where we do not have only the salary status value.

So, if you look at the variable explorer originally the columns\_list had 95 values, when we excluded the salary status variable from it the features list *have only* 95 the features list have only 94 values. But here we just have the column names now, so let us store the output values in y using the .values, the .values can be used to extract the values from data frame or a vector. So, I am going to extract the values from salary status and store it into y.

Similarly, we can do that to extract the values from features. So, that you will have all the corresponding values from the independent variables and then we can store it under the variable called x. Now x and y contains the integer values, where the x dimension is 30162 cross 94 and the y dimension is 30162. Next we are going to split the data into

train and test we have to divide the data set into a training set and a test set. So, that we can build the model on train set and deserve some part of data to test the model on. This can be done by using the `train_test_split` command, the input parameters for the train test split command would be `x` and `y`, where `x` represents the input values and `y` represents the output values and then comes the test size.

I have given it as 0.3 that represents the proportion of the data set to include in the test split. The next is the random state I have given a value I have given an integer value as 0. So, here random state is the seed used by the random number generator, so that each and every time you run this line while sampling same set of samples will be chosen. If you have not set the random seed then different set of samples will be chosen for the analysis and I have saved the output into four different variables `train_x` `test_x` `train_y` and `test_y`.

So, let us see the dimension and what are the values that each of the variables have. So, first what we have done here is we have splitted our data into train and test, under train we have two sets of data which has only the input variables and the other will have the only the output variable.

Similarly, we have done it for the test set as well. So, the test will contain the input variable separately and the output variable separately. So, if you look at the `train_x` size 70 percentage of the data is contained in `train_x` which is 21113 observation with 94 variables.

(Refer Slide Time: 07:38)

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** Displays Python code for data processing:

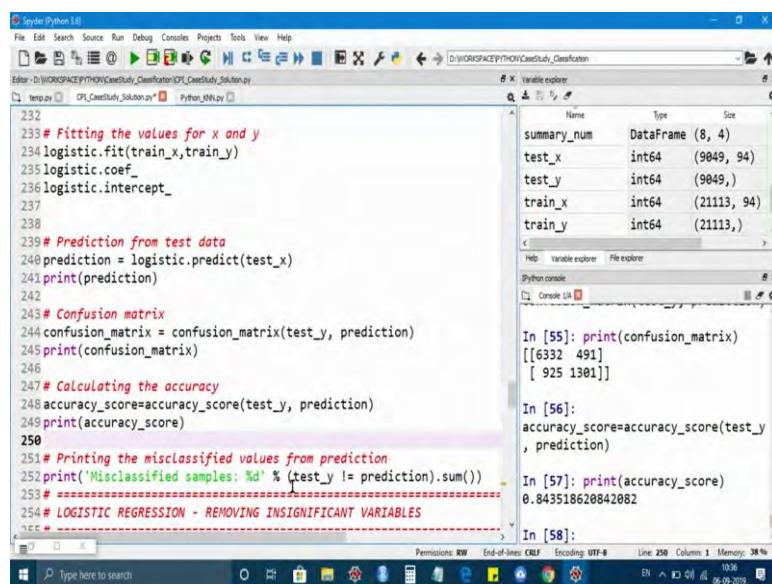
```
209 new_data=pd.get_dummies(dat
210
211 # Storing the column names
212 columns_list=list(new_data,
213 print(columns_list)
214
215 # Separating the input name
216 features=list(set(columns_1
217 print(features)
218
219 # Storing the output values
220 y=new_data['SalStat'].value
221 print(y)
222
223 # Storing the values from i
224 x = new_data[features].valu
225 print(x)
226
227 # Splitting the data into t
228 train_x,test_x,train_y,test
229
230 # Make an instance of the Model
231 logistic = LogisticRegression()
232
```
- Variable Explorer:** Shows the dimensions and types of variables:

| Name      | Type      | Size        | Value |
|-----------|-----------|-------------|-------|
| new_data  | DataFrame | (8, 4)      | Co... |
| columns_1 | int64     | (9049, 94)  | an... |
| columns_2 | int64     | (9049,)     | an... |
| train_x   | int64     | (21113, 94) | an... |
| test_x    | int64     | (21113,)    | an... |
| train_y   | int64     | (38162, 94) | an... |
| test_y    | int64     | (38162,)    | an... |
- File Explorer:** Shows the project structure.
- Output Window:** Shows the results of the code execution.

And the train\_y will have only the output column which will have only the values as zeros and ones. Similarly if you look at the test\_x size it has 30 percentage of the data to it that is 9049 observations with 94 variables and the test wise dimension is about 9400 9049 observation with only the salary status as output variable. Now you have split out the data into train set and test set.

So, next we are going to create a logistic regression classifier instance using the LogisticRegression function. So, here is the function that is logistic regression. So, let us make an instance of the logistic regression classifier.

(Refer Slide Time: 08:25)



The screenshot shows the Spyder Python IDE interface. The code editor displays a script named 'temp.py' with the following content:

```

232 # Fitting the values for x and y
233 logistic.fit(train_x,train_y)
234 logistic.coef_
235 logistic.intercept_
236
237
238
239 # Prediction from test data
240 prediction = logistic.predict(test_x)
241 print(prediction)
242
243 # Confusion matrix
244 confusion_matrix = confusion_matrix(test_y, prediction)
245 print(confusion_matrix)
246
247 # Calculating the accuracy
248 accuracy_score=accuracy_score(test_y, prediction)
249 print(accuracy_score)
250
251 # Printing the misclassified values from prediction
252 print('Misclassified samples: %d' % (test_y != prediction).sum())
253 #####
254 # LOGISTIC REGRESSION - REMOVING INSIGNIFICANT VARIABLES
255

```

The Variable explorer shows the following data structures:

| Name        | Type      | Size        |
|-------------|-----------|-------------|
| summary_num | DataFrame | (8, 4)      |
| test_x      | int64     | (9049, 94)  |
| test_y      | int64     | (9049,)     |
| train_x     | int64     | (21113, 94) |
| train_y     | int64     | (21113,)    |

The Python console shows the following outputs:

- In [55]: `print(confusion_matrix)`  
[[6332 491]  
[ 925 1301]]
- In [56]:  
`accuracy_score=accuracy_score(test_y, prediction)`
- In [57]: `print(accuracy_score)`  
0.843518620842082
- In [58]:

Then we can fit the model on the train set using the fit function and input for the fit function should be the input variables and the output variable. So, we have that under train\_x and train\_y respectively and I am using the LogisticRegression instance here to fit the model on train data.

So, now we have fitted the model on to the train data. So, you can extract some of the attributes from the logistic regression classify model by using the model name that is logistic and by using the .operator you will be able to access some of the attributes from it. So, let us get the coefficients of the logistic regression model. So, so in the console you will be able to get the coefficient values of all the variables. Similarly if you want to look at the intercept value you can also get that.

So, you can get that by accessing the intercept from the logistic regression model. So, the intercept is -3.8456. So, now we have built the model which will be used to classify the individual salary status as less than or equal to 50000 or greater than 50000. So, now we need to predict it on to a new data set, so that we can see how the model is performing. So, we have over data frame called test\_x which the model has not seen yet. So, I am going to predict the model on to the test\_x data frame using the predict function and I am saving my output to a variable called prediction. So, that my so that the predictions will be stored in to the variable called prediction.

So, let us print the prediction and see what it has ideally it should be having only the values are zeros and one, which basically gives you the salary status of the test data frame. So, now we have built a logistic regression model and we have also tested it on to a new data frame called test\_x and we also got the predictions.

So now, we have to evaluate the model using the confusion matrix, confusion matrix is a table that is used to evaluate the performance of a classification model. The confusion matrix output gives you the number of correct predictions and the number of incorrect prediction and it will sum up all the values class wise. So, let us see how to use the confusion matrix in order to evaluate the model.

So, confusion matrix is the function the input should be just the actual and the predicted values, the actual salary status is under the vector test\_y and the predictions are under the variable prediction. So, those can be the input for the confusion matrix and we are storing the output into object called confusion matrix. So, let us print the confusion matrix and see the output.

So, in the output you have four values, the diagonal values gives you the total number of correctly classified samples and the off diagonal values gives you the total number of wrongly classified samples. So, in the column what you have here is the predictions and in the row wise what you have is the what you have is the class for the actual and in the rows you have the actual classes.

For example, if the actual class is less than or equal to 50 and the model has predicted 6332 observations as less than or equal to 50000 but being less than or equal to 50000 is the actual class the model has predicted 491 observation as greater than 50000. Similarly given the salary status is greater than given the actual salary status is greater than 50000,

the model has predicted thousand 301 observation as greater than 50925 observations as less than or equal to 50000.

So, there are many misclassification secure the model has not classified all the observations correctly. So, using a measure called accuracy we will be able to get the accuracy score of the model that variable. So, it gives as an idea about how accurate our model is. So, you can get the accuracy for the model that we have built using the function called `accuracy_score`, the input can be the actual class and the predicted class and we are storing it into a variable called `accuracy_score` and when we print the accuracy, so the values turned out to be 0.8435. So, it means that 84 percentage of the time the model is able to classify the records correctly. So, now we have got an idea about how many misclassifications are there and what is the accuracy for the model that we have built.

We can also get the misclassified values from the predictions, like how many number of misclassifications are there you can do that by setting a simple condition you can get that by giving a simple condition. So, if you print the output it says 1416 observation has been misclassified. So, now you got an idea about how many misclassified number of samples were also there. So, now we have built a logistic regression model, but you can also improve the accuracy of the model by reducing the number of misclassified samples.

So, one of the method is by removing the insignificant variables, when we exclude more on the data we found out that there were sum in significant variables that might not contribute more in classifying the individual salary status. So, we are going to ignore those variables when we model it. So, what we are going to do here is we are going to build a new logistic regression model by removing all the insignificant variables.

(Refer Slide Time: 15:02)

The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'temp.py' with the following content:

```
275 # Storing the output values in y
276 y=new_data['SalStat'].values
277 print(y)
278
279 # Storing the values from input features
280 x = new_data[features].values
281 print(x)
282
283 # Splitting the data into train and test
284 train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.1)
285
286 # Make an instance of the Model
287 logistic = LogisticRegression()
288
289 # Fitting the values for x and y
290 logistic.fit(train_x,train_y)
291
292 # Prediction from test data
293 prediction = logistic.predict(test_x)
294
295 # Calculating the accuracy
296 accuracy_score=accuracy_score(test_y, prediction)
297 print(accuracy_score)
298
```

The Variable explorer window shows the following variables:

| Name           | Type      | Size        | Value                                                                                                                                                                      |
|----------------|-----------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| accuracy_score | float64   | 1           | 0.8388772240026522                                                                                                                                                         |
| cols           | list      | 4           | [1, 0, 2, 3]                                                                                                                                                               |
| columns_list   | list      | 44          | [1, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44] |
| correlation    | DataFrame | (4, 4)      | Correlation matrix                                                                                                                                                         |
| data           | DataFrame | (31978, 13) | Original dataset                                                                                                                                                           |
| data2          | DataFrame | (30162, 13) | Dataset after dropping insignificant columns                                                                                                                               |

The Python console window shows the following outputs:

```
In [104]:
In [104]: print(accuracy_score)
0.8388772240026522
In [105]:
```

So, let us first map all the less than or equal to 50 to 0 and greater than to 1 and then store it to the original data frame salary status and then store it to the variable called salary status of the data frame data2 and then these are the variables that were insignificant that is gender, nativecountry, race and JobType and I am creating the list of columns and I am storing it under the list called calls and I am going to drop all these listed columns from the data frame data2, so that I am not going to consider all these variables when I model it.

And the new data frame is created by dropping out all the insignificant variables to proceed with the further analysis and after removing the columns we need to get the dummies of all the categorical variables. So, we are also doing that and the other steps remind the same like I have stored all the column names in two columns\_list and then I have separated the input names from the data by excluding only the salary status variable. So, features will have only the set of column names of the input variables and we are going to extract the values in y and we are going to extract the values from salary status and store it into y.

Similarly we can also extract the values from the features and store it as x and then again we are splitting the data into train and test. So, that we can build the model on train and we can use the test data frame to test the model on and then we are creating an instance

call logistic from the function LogisticRegression, which is used to build the logistic regression classifier model and we are using the instance created that is logistic.

We are going to fit the logistic regression model on the train data and after building the model we can predict the model on to the new data frame that is test\_x. So, once we have got the prediction we are interested in getting or we are interested in improving the accuracy by removing all the insignificant variables. So, let us check whether the accuracy has improved by removing all the insignificant variables, we are going to get the accuracy using the accuracy score function.

So, when we print the accuracy score the accuracy is turned out to be 8.8388 which has higher than the accuracy score that we have got from the model were we have used all the variables. So, the accuracy has dropdown from 85 percent to 83 percent, because we have removed all the insignificant variables.

So, by removing the insignificant variables we are not getting a better model which gives us the better accuracy, when we remove any variables of course we are losing some information from the data. So, that could be the reason that is why we are getting a decrease in accuracy, but as compared to the model with all the variables it is not very low it is just a slight decrease in the accuracy. So, if you want to reduce the data collection part, if you want to retain only the significant variables which is used to classify the data. Then we can remove all the insignificant variables and keep this model as the final model.

(Refer Slide Time: 18:32)

The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'CPI\_CastStudy\_Solution.py' with the following content:

```
384 # *****
385
386 # importing the library of KNN
387 from sklearn.neighbors import KNeighborsClassifier
388
389 # import library for plotting
390 import matplotlib.pyplot as plt
391
392
393 # Storing the K nearest neighbors classifier
394 KNN_classifier = KNeighborsClassifier(n_neighbors = 5)
395
396 # Fitting the values for X and Y
397 KNN_classifier.fit(train_x, train_y)
398
399 # Predicting the test values with model
400 prediction = KNN_classifier.predict(test_x)
401
402 # Performance metric check
403 confusion_matrix = confusion_matrix(test_y, prediction)
404 print("\t","Predicted values")
405 print("Original values","\n",confusion_matrix)
406
407 # Calculating the accuracy
```

The variable explorer shows the following variables:

| Name             | Type      | Size        |
|------------------|-----------|-------------|
| accuracy_score   | float64   | 1           |
| columns_list     | list      | 95          |
| confusion_matrix | int64     | (2, 2)      |
| correlation      | DataFrame | (4, 4)      |
| data             | DataFrame | (31978, 13) |

The Python console window shows the following error:

```
ebea13d74b8>, line 1, in <module>
 confusion_matrix =
confusion_matrix(test_y,
prediction)
TypeError: 'numpy.ndarray' object
is not callable
```

The In [22]: section shows the command being run.

Now we have looked at the performance of the logistic regression model, so similarly we are going to build a KNN model, KNN classifier model to classify the records into any one of the categories of the salary status. So, to build a KNN model we are going to import k nearest neighbor classifier package from the sklearn neighbors. So, we have to import that, so after importing KNeighborClassifier we are importing the matplot library we are importing pyplot from matplot library as plt.

We will be using the matplot library to visualize some of the outputs of the k nearest neighbor classifier model so let us import that. So, after importing the necessary libraries we are creating an instance for k nearest neighbors, we are creating an instance for k nearest neighbor classifier using the function k neighbors classifier and inside the function I have specified n neighbors as 5 that is the k value.

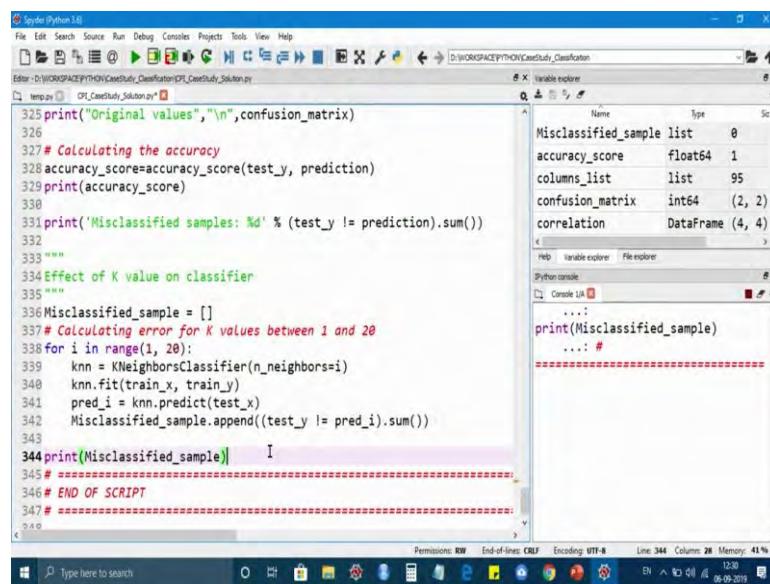
So, that it consider five neighbors when classifying the data into less than or equal to 50000 or greater than 50000. So, if it considers 5 neighbors, then it will take the majority classes from the 5 neighbors and then it will classify the new data based on the majority voting method and we are also saving the output to an object called KNN classifier. So, KNN classifier is the model, so using the instance that we have created using the k neighbor's classifier that is KNN classifier. We are going to fit the model on to a data frame called on to the train data frame using the fit function and input to the fit function should be the input values and the output values of from the train data frame.

So, now we have build the classify model using the KNN algorithm now let us see how the model is performing in a new data frame. So, we have a test data frame so that we can test a model on. So, I am going to predict the model on to a new data frame called test using the predict function.

So, the input should be the input values of the test data frame and I am saving the output into a variable call prediction. So, the prediction will have the values of the salary status as zeros and one, where 0 represents less than or equal to 50 and 1 represent greater than or equal to 50000. So, now, we have predicted the values and you have also seen the values from the prediction output. Now, it is time to check how good of a model is performing using the confusion matrix.

So, the input can be the actual values and the predicted values of the salary status, which is there under test\_y and prediction respectively and we have also save that to a variable called confusion\_matrix.

(Refer Slide Time: 21:25)



The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'temp.py' with the following content:

```

325 print("Original values", "\n",confusion_matrix)
326
327 # Calculating the accuracy
328 accuracy_score=accuracy_score(test_y, prediction)
329 print(accuracy_score)
330
331 print('Misclassified samples: %d' % (test_y != prediction).sum())
332
333 """
334 Effect of K value on classifier
335 """
336 Misclassified_sample = []
337 # Calculating error for K values between 1 and 20
338 for i in range(1, 20):
339 knn = KNeighborsClassifier(n_neighbors=i)
340 knn.fit(train_x, train_y)
341 pred_i = knn.predict(test_x)
342 Misclassified_sample.append((test_y != pred_i).sum())
343
344 print(Misclassified_sample) I
345 # ****END OF SCRIPT*****
346 # ****END OF SCRIPT*****
347 #

```

The variable explorer window on the right shows the following variables:

| Name                 | Type      |
|----------------------|-----------|
| Misclassified_sample | list      |
| accuracy_score       | float64   |
| columns_list         | list      |
| confusion_matrix     | int64     |
| correlation          | DataFrame |

The Python console window shows the output of the command 'print(Misclassified\_sample)'.

So, now, let us look at the output of the confusion matrix. So 6338 observations have been classified properly as less than or equal to 51285 observations are correctly classified as greater than or equal to 50000 with the actual class. When the actual salary status is greater than 50, but the model has classified it as less than or equal to 50. So, now we have using the confusion matrix we have got an idea about how many

misclassified samples are there and how many correctly classified and how many correct predictions are there.

We can also look at the accuracy score using the accuracy score function we can also look at the accuracy of the model that we have built using the accuracy score function and the input to the accuracy score function should be the actual and the predicted classes of the salary status. So, let us print the accuracy score.

So, the accuracy of the KNN model is 0.8424, 84 percentage of the time the KNN model is able to classify the records correctly. If you want to check how many number of misclassified samples are there you can also check that using the same condition that you have already used for the logistic regression, when we check the output the misclassified samples are 1456 for the KNN model that we have built.

So, now under the neighbors argument while we are building the KNN model we have just randomly fixed a k value as 5. So, here we are trying to calculate the error for k values between 1 and 20 by iterating through for loop. So, I have given the range as 1 to 20, so that the k neighbors classifier will takes the k value from 1 to 20 and builds the model based on the k value, since I have given i under the neighbors.

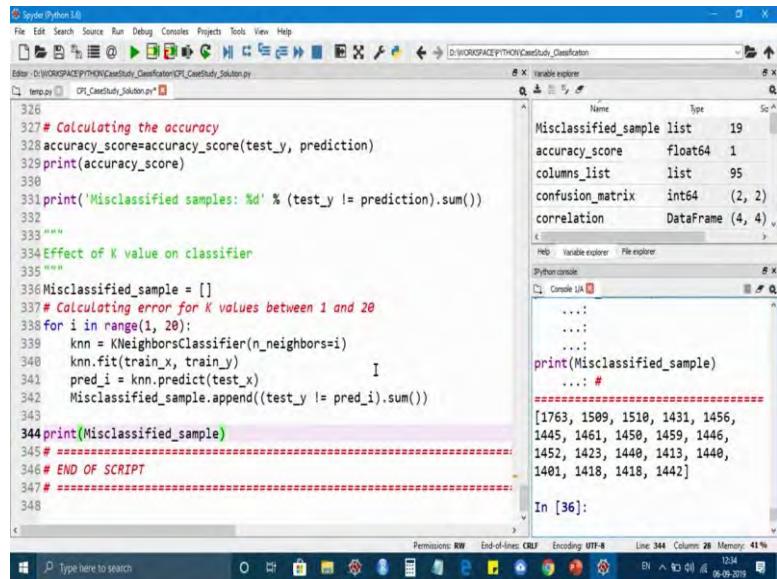
So, once you have created an instance called KNN we are going to fit the model on the train data frame using the fit function and the train\_x becomes the input values and the train\_y as the output value. So, once we have build the model you are going to predict it on to a new data frame called test test\_x using the predict function. So, all the creation of instance building the model and predicting it on test are happening inside the for loop.

So, that for each and every value of k you will get a different models and once we have predicted and the predictions are stored into a variable called pred\_i. So, that you will have the predictions under pred\_i and in the next step we are going to look at the number of misclassified samples when we fix the k values from 1 to 20. So, you will have the misclassified samples for each and every value of k that we have fixed here.

So, let us run the whole for loop and see what is the output. So, the print so the misclassified samples will give you the number of misclassified sample with respect to each values of k. So, based on the output you will be able to arrive at a k value for your k

nearest neighbor model, wherever you have the less number of misclassified samples you can fix that as the k value to build any k to build the KNN model.

(Refer Slide Time: 24:53)



The screenshot shows the Spyder Python IDE interface. The code editor displays a script named 'CPL\_CaseStudy\_Solution.py' with several print statements and a loop for calculating accuracy scores. The variable explorer shows a list of variables including 'Misclassified\_sample list', 'accuracy\_score', 'columns\_list', 'confusion\_matrix', and 'correlation'. The Python console window shows the output of the script, specifically the list of misclassified sample IDs: [1763, 1509, 1518, 1431, 1456, 1445, 1461, 1458, 1459, 1446, 1452, 1423, 1448, 1413, 1448, 1401, 1418, 1418, 1442]. The status bar at the bottom indicates the line number is 344, column 28, memory usage is 41%, and the date is 06-09-2019.

```
326
327 # Calculating the accuracy
328 accuracy_score=accuracy_score(test_y, prediction)
329 print(accuracy_score)
330
331 print('Misclassified samples: %d' % (test_y != prediction).sum())
332
333 """
334 Effect of K value on classifier
335 """
336 Misclassified_sample = []
337 # Calculating error for K values between 1 and 20
338 for i in range(1, 20):
339 knn = KNeighborsClassifier(n_neighbors=i)
340 knn.fit(train_x, train_y)
341 pred_i = knn.predict(test_x)
342 Misclassified_sample.append((test_y != pred_i).sum())
343
344 print(Misclassified_sample)
345 # =====#
346 # END OF SCRIPT
347 # =====#
348
```

So, from the output it is clear that for the k value 16 we are getting the number of misclassified samples as 1401. So, if we fix the k value of 16 then the model was performing good, because we are getting a less number of misclassification. Then we can fix this as a k value when we build a when we want to improve the model further. So, that so that we will not end up in misclassifying the salary status.

So, in this case study we have looked at two algorithms, one is logistic regression and the other one is KNN both of the algorithms gives you the same performance when we looked at the accuracy and in terms of number of misclassification.

But on the whole the logistic regression is performing better with the accuracy of 85 percent that when the KNN model gives you the accuracy of 84 percent but we can also improve the model further by dropping few more insignificant variables in the logistic regression as well as in the KNN classifier model because in the KNN classifier model that we have built here is only with is only including all the variables; we have not excluded this insignificant variable and check the accuracy. So, we can also do that to improve the performance of the classifier model that we have built.

**Python for Data Science**  
**Prof. Prathap Haridoss**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 25**  
**Introduction to Regression Case Study**

So, the previous case study that we introduced was a classification case study and as a second case study we are going to Introduce a Regression or a function approximation case study. It basically follows the same format as the classification case study. So, I am going to quickly go through this case study introduction so that you can see how this case study is solved in Python.

So, the case study is on predicting the price of pre-owned cars. So, right at the beginning we can see that here there is a slight difference between what we saw in the last case study and this. In the last case study we simply wanted to classify individuals into two possible categories people who are making less than 50,000 and people who are making more than 50,000 whereas, here what we are looking to do is we are actually trying to see if we can predict the price of pre-owned cars. So, there is just no category, it is a value for a pre-owned car and how do we predict the price of pre-owned cars.

(Refer Slide Time: 01:31)

**Problem statement**

*Storm Motors is an e-commerce company who act as mediators between parties interested in selling and buying pre-owned cars.*

*For the year 2015-2016, they have recorded data about the seller and car including-*

- Specification details*
- Condition of car*
- Seller details*
- Registration details*
- Web advertisement details*
- Make and model information*
- Price*

*Storm Motors wishes to develop an algorithm to predict the price of the cars based on various attributes associated with the car.*



Python for Data Science

So, the problem statement is a Storm Motors is an e-commerce company, then they act as mediators between parties interested in selling and buying pre-owned cars and they have

a lots of data based on sales that have happened through them or otherwise and what the interest is in is to make a sale quickly. So, if the price is appropriate and that is something that you can satisfy both the seller and the buyer, then you will have your cars moving fast.

So, what Storm Motors wants to do is they want to develop an algorithm, so that they can predict the price of the cars based on various attributes associated with the car. So, you might think of using a such a model from a Storm Motors viewpoint in multiple ways. If if a seller is asking for a price you could put the attributes of the car that the seller is selling into this model and then come up with a predicted price, and if it is much above what it is predicting then you can tell the seller that you know you are asking for too much and you are not likely to sell this car at this price.

And, similarly a buyer comes and bids for a car at a much lesser price than what has been put up, then you could show the results of the model and tell the buyer, look you are asking for a very cheap price so, the seller is unlikely to give you this car at this price. So, if you got this right then you could optimize this transaction and then have a both the parties being happy and then you have a better business.

(Refer Slide Time: 03:33)

| Variable description |           |                                                                           |                            |
|----------------------|-----------|---------------------------------------------------------------------------|----------------------------|
| Variables            | Data Type | Description                                                               | Categories                 |
| dateCrawled          | date      | date when the ad first crawled, all field values are taken from this date | --                         |
| name                 | string    | string consisting of car name, brand, model etc                           | combination of strings     |
| seller               | string    | nature of seller                                                          | private, commercial        |
| offerType            | string    | whether the car is on offer or has the buyer requested for an offer       | offer, request             |
| price                | integer   | price on the ad to sell the car (\$)                                      | --                         |
| abtest               | string    | two versions of ad                                                        | test, control              |
| vehicleType          | string    | types of cars                                                             | cabrio, coupe and 5 seater |
| yearOfRegistration   | integer   | year in which was first registered                                        |                            |



And, much like what we saw in the classification problem again we think about this data in a matrix format. So, Storm Motors has data for about 50,000 cars in their database that have been sold or that have been processed in one way another and there

are these 19 variables that are associated with this problem. Clearly, one of these variables is going to be the outcome variable or the price of the car and the other variables are variables that we hope have enough information in them so that we can basically predict the price of the car.

And, much like in the last example if we go through these variables so, there is a variable called dateCrawled and the data type is date. So, this is a variable where a once Storm Motors puts the ad out when did it catch the first eyeball. So, when was the first time this ad was crawled and all field values taken from this data, so, that date is the first date crawl. So, it gives you an idea of when people roughly started looking at a this car.

Now, the name is another variable and this is kind of a composite variable. This is been downloaded from someplace. So, this is a string variable, but this is a string variable that could consist of car name, brand, model etcetera. So, it is kind of a composite string and in this data set you will see that it is not always consistent in terms of the order in which all of the information comes in.

Seller whether it is a private or a commercial seller; offer type is whether the buyer has looked at a particular car and then gave an offer saying this is your price, but I am willing to buy this car for this price or it is a price that the seller has asked for. So, that is the other variable that we have. Price is the price on the ad to sell the car which is the outcome variable that we are interested in and the way these ads are put in there are certain characteristics of these ads and as part of this exercise Storm Motors also had some specific studies that they want to conduct.

So, the ads could be of the test type or the control type and the data type is a string vehicle type is a string whether it is a Cabrios, SUV, Coupe and 5 more different types of vehicle; year of registration year in which the car was first registered which is an integer variable.

Now, you can see that many of these clearly will have an impact on what price you can sell the car right. So, for example, vehicle type is an important aspect one would one would assume and there are more variables that we will see; seller for example, private or commercial could have an impact or might not. So, this is something that we might not know; private sellers might want to hold on and get the best price for the car,

sometimes commercial guys might want to move cars out. So, we really do not know how much an impact it will have, but these are all relevant variables.

So, whenever you look at a data sense problem you basically look at whether the variables are relevant to the problem and then how relevant and quantitatively, how much the impact and so on, only the data will tell you. So, that is post the solution you will be able to understand.

(Refer Slide Time: 07:37)

The screenshot shows a table titled "Variable description" with the following details:

- Total size :50000 x 19
- Data file : cars\_sample.csv

| Variables           | Data Type             | Description                                                                                              | Categories                          |
|---------------------|-----------------------|----------------------------------------------------------------------------------------------------------|-------------------------------------|
| gearbox             | string                | type of gearbox                                                                                          | manual or automatic                 |
| powerPS             | integer               | power of the car (HP)                                                                                    | --                                  |
| model               | string                | model type of the car                                                                                    | 3er,xc_reihe and 248 more           |
| kilometer           | integer               | number of kilometres the car has travelled                                                               | --                                  |
| monthOfRegistration | integer (qualitative) | month of registration                                                                                    | 1,2,3,...,12                        |
| fuelType            | string                | types of fuel                                                                                            | petrol, diesel and 5                |
| brand               | string                | make of car                                                                                              | bmw, audi, volkswagen and 30 others |
| notRepairedDamage   | string                | status of repair for damages<br>if yes damages have not been rectified; if no damages were taken care of |                                     |

At the bottom left of the slide, there are navigation icons: back, forward, search, and Python for Data Science.

Now, going on to more variables and there is a variable called gearbox which is string, it can be manual or automatic; powerPS – power of the car and this is an integer and there are multiple values this can take. Model of the car model type of the car for example, if it is a Hyundai is it an i10, i20 and so on; kilometre – number of the number of kilometers the car has travelled which is an integer variable. Month of registration, fuelType whether it is a patrol car, diesel car and 5 more. Brand – what type of car is it, is it a BMW, Mercedes and many others.

Now, we also have another variable which might be important from a price of car view point: notRepairedDamage is a variable name. So, this is a string if it is yes, then basically what it says is there has been a damage and it is not been repaired. So, are not rectified and no means there has been a damage, but the car has been repaired and rectified. So, you can again see how this variable might have an impact in terms of the final price.

(Refer Slide Time: 09:03)

The screenshot shows a Jupyter Notebook interface. At the top, it says "Variable description". Below that, it shows "Total size : 50000 x 19" and "Data file : cars\_sample.csv". A table titled "Variable description" is displayed with four columns: "Variables", "Data Type", "Description", and "Categories". The table contains three rows of data:

| Variables   | Data Type | Description                                     | Categories |
|-------------|-----------|-------------------------------------------------|------------|
| dateCreated | date      | date at which the ad at storm motor was created | --         |
| postalCode  | integer   | postal code of seller                           | --         |
| lastSeen    | date      | when the crawler saw this ad last online        | --         |

At the bottom of the slide, there is a small video player icon and the text "Python for Data Science".

So, dateCreated another variable of data type date data to with that Storm Motors was created; postalCode – postal code of the seller and lastSeen – when the crawler saw this ad last online. So, what has been the most recent activity in terms of interest in terms of this car.

(Refer Slide Time: 09:23)

The screenshot shows a Jupyter Notebook interface. At the top, it says "Variable description". Below that, it says "The variables can be grouped in to different buckets based on the information". A table titled "Details" is displayed with two columns: "Details" and "Variables". The table contains six rows of data:

| Details               | Variables                                                   |
|-----------------------|-------------------------------------------------------------|
| Specification details | gearbox, power, fuelType                                    |
| Condition of car      | notRepairedDamaged, kilometer                               |
| Seller details        | seller, postalCode                                          |
| Registration details  | yearOfRegistration, monthOfRegistration                     |
| Make and model        | brand, model, vehicleType                                   |
| Advertisement details | dateCrawled, name, abtest, dateCreated, lastSeen, offerType |

At the bottom of the slide, there is a small video player icon and the text "Python for Data Science".

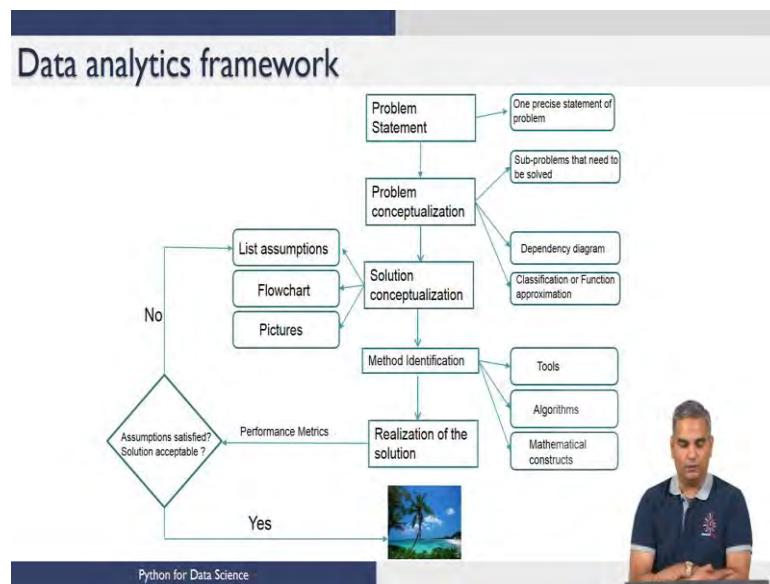
Now, you can see in kind of think about this and then say each of these variables could have an impact. For example, the last thing that I talked about if a car is not getting any eyeballs at all, then you know maybe it is it is it is not priced right. So, you have to really

think about how to price it, so that you can sell it ok. So, based on this information the variables can be grouped into different buckets which is something that you can think about.

And, say well, the vehicle specification details such as gearbox, power, fuel type is one group; condition of the car and not repair damage and kilometres another group. Both of them in some sense tell you how the car is likely to be in what condition is it likely to be and so on if it is run a large number of kilometres then you know it is a older car so, condition might not be great and of course, not repair damage directly gives you information about the condition of a car.

Seller details again which part of the geography the seller comes from and so on. So, those can be grouped under seller details. Registration details – year of registration, month of registration of course, the car itself make an model and what is happening based on the advertisements that we have and does that give you any clue in terms of what is likely to the price is another group that you can think of.

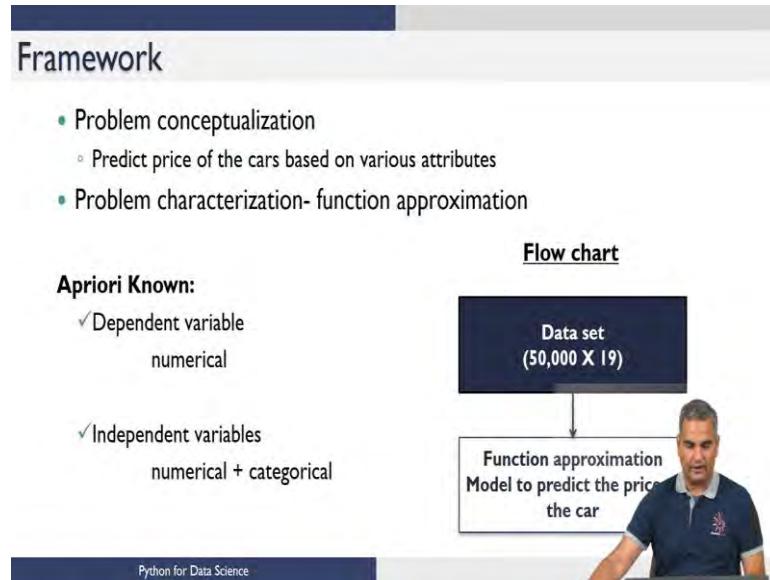
(Refer Slide Time: 11:05)



Again, I do not want to go through this slide again. We use the same process that we talked about in the classification example. Same notions of problem statement, very clear here. We want to predict a price of a used car and again since it is a very simple straight one problem you do not really need to look at sub-problems and so on. It is a function approximation or a regression problem and much like how we talked about the previous

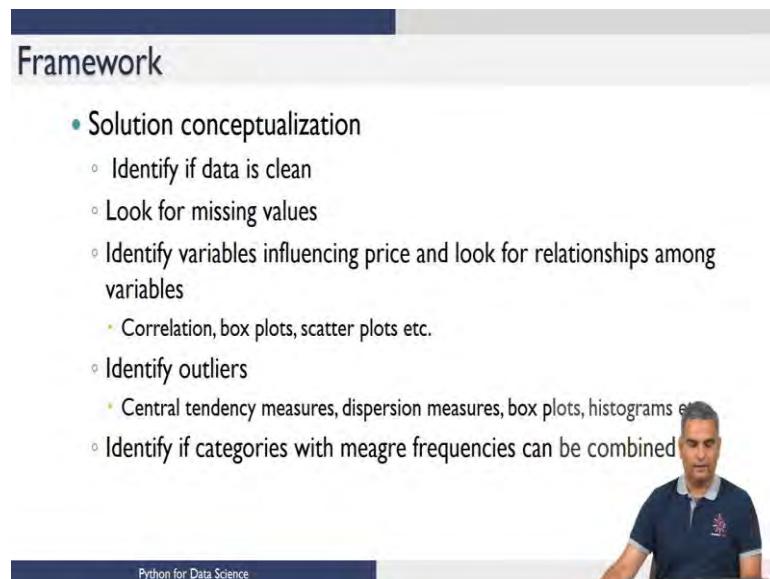
case, again here we are going to run a several different models and then find out whichever model does well and all of this we are going to again do this using Python. So, that is basically what we have.

(Refer Slide Time: 11:47)



So, we know that the dependent variable is a numerical variable which is the price of the car that we want to predict and again independent variables have a combination of numerical and categorical variables.

(Refer Slide Time: 12:03)



Go through a pretty much same kind of ideas that we talked about in the classification problem. First look if the data is clean, look for missing values, look for variables that might influence price. So, this is where you do some form of visualization, descriptive statistics and so on, and also in this case one of the important things is to identify outliers. There are data points which really do not make much sense at all based on whatever logic that we used.

So, for example, if there are a lot of data points which say price of car is 0, then you know there might be some issue with that data. So, you want to kind of remove those kinds of data points. Similarly, power of the car; if it is a ridiculously small number then you know cars do not come at those powers and so on. So, you can remove those. So, that is another important thing that you will see in this case study and how do you think about outliers the very formal mathematical ways of thinking about outliers and removing them and there are also more common sense notions of what outliers are and then you can remove these outliers.

And, also if you have certain categories with very little frequency of occurrence then you can think about whether you can combine two categories into a same category and so on, so that you have a much more compact data set that you can work with.

(Refer Slide Time: 13:43)

## Framework

- Solution conceptualization (contd.)
  - Filter data based on logical checks
    - Price, year of registration, power
  - Reduced number of data
- Method identification
  - Linear regression
  - Random forest



So, as I mentioned before, we can filter data based on logical checks, price, year of registration power. So, in some cases you will notice in this data set the year of

registration does not make sense at all. So, there are years which are past 2019 and there are years which are very far back from 2019. So, you can remove data like that and then get a reduced number of data.

And, as I mentioned before we are going to look at two different techniques one is linear regression which sees if there is one linear model which will fit the output variable as a function of the input variables. If that is the case then the model is simple, analysis is simple, variable importance everything is simple. So, we will first try that and also we will see whether a non-sample method such as a random forest approach will work in this case better than a simple linear regression approach and basically it is a trade-off in terms of understandability of the model, how much better one model does over the other and so on.

And, what you are going to use this model for; are you going to embed this model in some other optimization and so on, in which case you might want to worry about complexity explainability and all that. Otherwise if you are just going to use this in this example if a random forest model that is much better you might simply go ahead and use it for this example.

(Refer Slide Time: 15:11)

## Framework

- Realization of solution
  - Assumption checks using regression diagnostics
  - Evaluate performance metrics
  - If assumptions are satisfied and solutions are acceptable then model is good
  - If performance metrics are not reasonable then a single model is not able to capture the variation in price as a whole
  - In such cases, it would be better to subset data and build separate models



Python for Data Science

So, you will notice that you will do assumption checks using regression diagnostics. Linear regression models come up with certain assumptions which once you build a model you can verify whether the model allows us to understand whether

this data follows the assumptions that we are laid out at the beginning. You can evaluate performance metrics.

In this case the performance metric is very straight forward which is basically how much error is there in your prediction and if you are happy with the result you simply live with it; if you are not happy with the result then you might say well there might be some other processing of the data I can do and then maybe get better models. Maybe one idea would be to better subset data and say I will build separate models for this cars of.

One very simple idea is let me group all the cars with very low price and then maybe build a model which I might call as a model for low priced cars and then I could subset all the cars with a larger price and then build a model for higher priced cars and so on. So, these are things that come out of your analysis after you build the first level of data science solution and then understand whether that is good enough or not or you have to do more to make your results acceptable for use.

So, in the lecture that follows this, you will see how you solve this whole problem in Python and look at these various choices and then see how you give a holistic solution to this problem. Hope these case studies are useful in your journey towards understanding data science and in particular using Python to understand data science and I just want to say that this is just a beginning. These are simple case studies, nonetheless they are themselves useful case studies for you to understand a process by which you solve these problems and as you go forward hopefully you will encounter a more complex and rich problems that that you get to solve in this area of data science.

Thank you.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 26**  
**Case study on regression**

Welcome to the Case study on regression. In this case study we are going to take a problem that is on predicting price of pre owned cars.

(Refer Slide Time: 00:19)

**Problem statement**

*Storm Motors is an e-commerce company who act as mediators between parties interested in selling and buying pre-owned cars.*

*For the year 2015-2016, they have recorded data about the seller and car including-*

- Specification details*
- Condition of car*
- Seller details*
- Registration details*
- Web advertisement details*
- Make and model information*
- Price*

*Storm Motors wishes to develop an algorithm to predict the price of the cars based on various attributes associated with the car.*

So, let us go ahead and take a look at the problem statement. Now storm motors they are any commerce company and they act as mediators between parties who are interested in selling or buying a pre owned cars. Now these are second hand cars and storm motors act as mediators. Now in specific for the year 2015 to 2016, storm motors they have recorded the data about the seller and car details.

Now, these set of details includes specification details, condition of the car, seller details, registration details, web advertisement details, make and model information and price. Now these are several buckets and there are specific parameters or variables that storm motors as collected and each of these variables will definitely fall into one of these buckets. Now storm motors what they want to do is that, they want to develop an algorithm that will help them predict the price of the pre-own cars and this can be based on the various attributes that are associated with the car.

(Refer Slide Time: 01:23)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
5 #author: GITAA
6 #
7 #
8 # =====
9 # PREDICTING PRICE OF PRE-OWNED CARS
10# =====
11
12import pandas as pd
13import numpy as np
14import seaborn as sns
15
16# =====
17# Setting dimensions for plot
18# =====
19
20sns.set(rc={'figure.figsize':(11.7,8.27)})
21
22
23# =====
24# Reading CSV file
25# =====
26cars_data=pd.read_csv('cars_sample.csv')
27
28# =====
29# Creating copy
```

The variable explorer shows a DataFrame named 'cars\_data' with 50001 rows and 19 columns. The columns listed are: dateCreated, name, seller, offerType, price, witness, veh...

The Python console shows the following history:

```
In [3]: import pandas as pd
...: import numpy as np
...: import seaborn as sns
In [4]: sns.set(rc={'figure.figsize':(11.7,8.27)})
In [5]: cars_data=pd.read_csv('cars_sample.csv')
In [6]:
```

So, let us see how to solve this case study in python. So, let us start by importing a few required packages, first we have going to import pandas for reading all the files from the CSV format, then I am going to do a few numerical operation and hence I am going to import numpy as well. And I am going to visualize the data and derive some insides and going to import seaborn for it. So, I am importing pandas as pd numpy as np and see born as sns. So, let us just import these packages.

So, these packages have been imported now. Now I am setting a dimensions for all the plots that I am going to be generating, I am using the function sns touch set is a function from the seaborn package and within the parenthesis, I am giving the figure size and this sets the dimensions for rather. And this will set the dimensions for all our plots. So, let us begin by importing the CSV file. So, you have been given a data called cars\_sample.csv. Now I have already set my working directory and the data is also residing in my working directory.

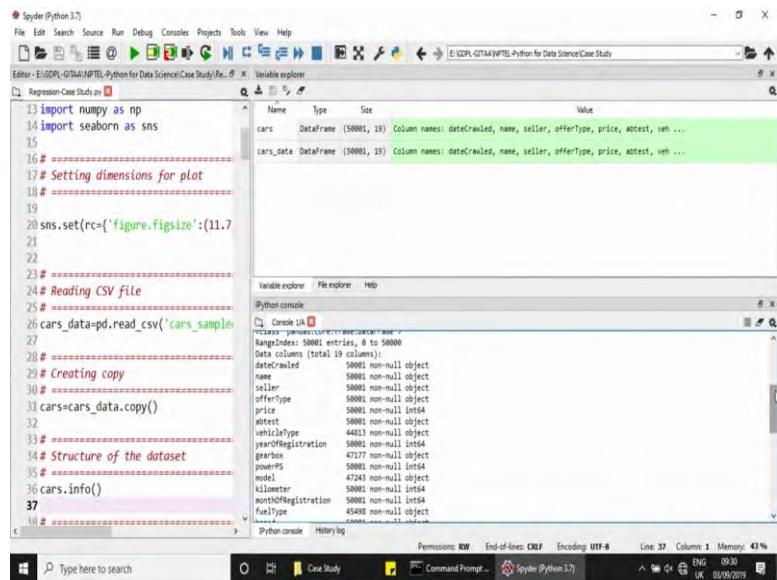
So, I am just going to right away read it using the pd.read\_csv function. So, the data has been read and we have 50001 records and 19 columns let us just open and see what does the data have to say.

(Refer Slide Time: 03:01)

| Index | dateCrawled | name                 | seller  | offerType | price   | test      | vehicleType   | yearOfRegistration | gearbox   | powerPS | model    | kilometer | consumption |
|-------|-------------|----------------------|---------|-----------|---------|-----------|---------------|--------------------|-----------|---------|----------|-----------|-------------|
| 0     | 30/03/2016  | Zu_verkaufen_private | offer   | 4450      | test    | limousine | 2003          | manual             | 150       | 183     | xc_reihe | 150000    | 3           |
| 1     | 23/03/2016  | VW_Passat_35i        | offer   | 13299     | control | svv       | 2005          | manual             | 150       | 183     | xc_reihe | 150000    | 6           |
| 2     | 09/04/2016  | Volvo_XC90_L_        | private | offer     | 3200    | test      | bus           | 2003               | manual    | 181     | touran   | 150000    | 11          |
| 3     | 09/03/2016  | Volkswagen_L_        | private | offer     | 4500    | control   | small car     | 2006               | manual    | 86      | idiz     | 60000     | 12          |
| 4     | 17/03/2016  | Seat_Ibiza_L_        | private | offer     | 18750   | test      | svv           | 2008               | automatic | 185     | xc_reihe | 150000    | 11          |
| 5     | 02/04/2016  | VW_Passat_35i        | private | offer     | 988     | test      | limousine     | 1995               | manual    | 99      | passat   | 150000    | 2           |
| 6     | 03/04/2016  | Opel_Astra           | private | offer     | 400     | test      | station wagon | 1996               | manual    | 0       | astra    | 150000    | 5           |
| 7     | 24/03/2016  | Mercedes_Ben_        | private | offer     | 1399    | test      | coupe         | 1997               | manual    | 136     | clk      | 150000    | 11          |
| 8     | 15/03/2016  | Vectra_C_Car_        | private | offer     | 4600    | test      | station wagon | 2005               | manual    | 122     | vectra   | 150000    | 12          |
| 9     | 05/04/2016  | Skoda_OctaviaL       | private | offer     | 8340    | test      | limousine     | 2005               | automatic | 140     | octavia  | 125000    | 4           |
| 10    | 21/03/2016  | Mercedes_Ben_        | private | offer     | 1870    | control   | limousine     | 2001               | manual    | 82      | a_klasse | 150000    | 1           |
| 11    | 25/03/2016  | Opel_Astra_          | private | offer     | 2500    | test      | coupe         | 2001               | manual    | 185     | astra    | 125000    | 4           |
| 12    | 12/03/2016  | Toyota_Yaris_        | private | offer     | 990     | test      | small car     | 2001               | manual    | 68      | yaris    | 150000    | 9           |
| 13    | 07/03/2016  | BMW_e46_318t         | private | offer     | 3000    | test      | nan           | 2016               | manual    | 116     | 3er      | 150000    | 0           |
| 14    | 08/03/2016  | Saab_9000i           | private | offer     | 2200    | control   | limousine     | 2003               | manual    | 83      | meriva   | 125000    | 5           |
| 15    | 02/04/2016  | Volkswagen_Golf      | private | offer     | 1600    | test      | station wagon | 1999               | manual    | 140     | nan      | 150000    | 3           |
| 16    | 12/03/2016  | Nissan_Terrano       | private | offer     | 1300    | test      | svv           | 1993               | manual    | 181     | others   | 150000    | 10          |
| 17    | 22/03/2016  | Iveco_KASTEN         | private | offer     | 4100    | control   | others        | 2002               | manual    | 125     | nan      | 150000    | 8           |
| 18    | 15/03/2016  | Vauxhall_Golf        | private | offer     | 600     | control   | small car     | 1998               | manual    | 75      | golf     | 150000    | 0           |
| 19    | 01/04/2016  | Golf_3_mit_t_        | private | offer     | 698     | test      | nan           | 2017               | nan       | 0       | golf     | 150000    | 0           |
| 20    | 13/03/2016  | Golf_3_mit_t_        | private | offer     | 698     | test      | nan           | 2017               | nan       | 0       | golf     | 150000    | 0           |
| 21    | 25/03/2016  | ...                  | ...     | ...       | ...     | ...       | ...           | ...                | ...       | ...     | ...      | ...       | ...         |

So, you have index from 0 to 50,001 you have the date crawled you have the name of the car; the seller type, the offerType the price you have a b test then vehicle type and year of registration and there are couple of other variables here, all these have been explain to you before the description of the variables. So, now, that we have read let us just create a copy of it. So, you have already seen how to create a copy of your data. So, you can do a deep copy and any change that you made to cars will not be reflected back in cars\_data.

(Refer Slide Time: 03:39)



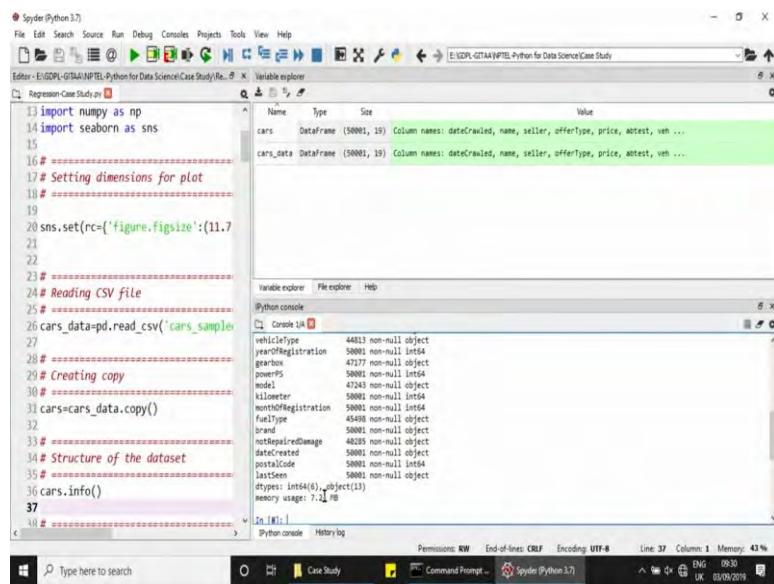
```

Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor: E:\DOP\GIT\DATA\NTEL\Python for Data Science\Case Study\Re...
Variable explorer
Regression-Case Study.ipynb
Name Type Size Value
cars DataFrame (50001, 19) Column names: dateCrawled, name, seller, offerType, price, test, veh ...
cars_data DataFrame (50001, 19) Column names: dateCrawled, name, seller, offerType, price, test, veh ...
Variable explorer File explorer Help
Python console
Console IPython
In [1]: %load_ext autoreload
Out[1]: <IPython.core.magic.Magic>
In [2]: %autoreload 2
Out[2]: None
In [3]: import numpy as np
Out[3]: <module 'numpy' (built-in)>
In [4]: import seaborn as sns
Out[4]: <module 'seaborn' (built-in)>
In [5]: # =====
In [6]: # Setting dimensions for plot
Out[6]: None
In [7]: # =====
In [8]: sns.set(rc={'figure.figsize':(11,7
Out[8]: None
In [9]: # =====
In [10]: # Reading CSV file
Out[10]: None
In [11]: # =====
In [12]: cars_data=pd.read_csv('cars_sample'
Out[12]: <pandas.core.frame.DataFrame object at 0x0000000004E8A800>
In [13]: # =====
In [14]: # Creating copy
Out[14]: None
In [15]: # =====
In [16]: cars=cars_data.copy()
Out[16]: <pandas.core.frame.DataFrame object at 0x0000000004E8A800>
In [17]: # =====
In [18]: # Structure of the dataset
Out[18]: None
In [19]: # =====
In [20]: cars.info()
Out[20]: <pandas.core.frame.DataFrame object at 0x0000000004E8A800>
In [21]: # =====
In [22]:
Out[22]: None
In [23]: # =====
In [24]: # Reading CSV file
Out[24]: None
In [25]: # =====
In [26]: cars_data=pd.read_csv('cars_sample'
Out[26]: <pandas.core.frame.DataFrame object at 0x0000000004E8A800>
In [27]: # =====
In [28]: # =====
In [29]: # =====
In [30]: # =====
In [31]: cars=cars_data.copy()
Out[31]: <pandas.core.frame.DataFrame object at 0x0000000004E8A800>
In [32]: # =====
In [33]: # =====
In [34]: # =====
In [35]: # =====
In [36]: cars.info()
Out[36]: <pandas.core.frame.DataFrame object at 0x0000000004E8A800>
In [37]: # =====
In [38]:
Out[38]: None
In [39]: # =====
In [40]:
Out[40]: None
In [41]: # =====
In [42]:
Out[42]: None
In [43]: # =====
In [44]:
Out[44]: None
In [45]: # =====
In [46]:
Out[46]: None
In [47]: # =====
In [48]:
Out[48]: None
In [49]: # =====
In [50]:
Out[50]: None
In [51]: # =====
In [52]:
Out[52]: None
In [53]: # =====
In [54]:
Out[54]: None
In [55]: # =====
In [56]:
Out[56]: None
In [57]: # =====
In [58]:
Out[58]: None
In [59]: # =====
In [60]:
Out[60]: None
In [61]: # =====
In [62]:
Out[62]: None
In [63]: # =====
In [64]:
Out[64]: None
In [65]: # =====
In [66]:
Out[66]: None
In [67]: # =====
In [68]:
Out[68]: None
In [69]: # =====
In [70]:
Out[70]: None
In [71]: # =====
In [72]:
Out[72]: None
In [73]: # =====
In [74]:
Out[74]: None
In [75]: # =====
In [76]:
Out[76]: None
In [77]: # =====
In [78]:
Out[78]: None
In [79]: # =====
In [80]:
Out[80]: None
In [81]: # =====
In [82]:
Out[82]: None
In [83]: # =====
In [84]:
Out[84]: None
In [85]: # =====
In [86]:
Out[86]: None
In [87]: # =====
In [88]:
Out[88]: None
In [89]: # =====
In [90]:
Out[90]: None
In [91]: # =====
In [92]:
Out[92]: None
In [93]: # =====
In [94]:
Out[94]: None
In [95]: # =====
In [96]:
Out[96]: None
In [97]: # =====
In [98]:
Out[98]: None
In [99]: # =====
In [100]:
Out[100]: None
In [101]: # =====
In [102]:
Out[102]: None
In [103]: # =====
In [104]:
Out[104]: None
In [105]: # =====
In [106]:
Out[106]: None
In [107]: # =====
In [108]:
Out[108]: None
In [109]: # =====
In [110]:
Out[110]: None
In [111]: # =====
In [112]:
Out[112]: None
In [113]: # =====
In [114]:
Out[114]: None
In [115]: # =====
In [116]:
Out[116]: None
In [117]: # =====
In [118]:
Out[118]: None
In [119]: # =====
In [120]:
Out[120]: None
In [121]: # =====
In [122]:
Out[122]: None
In [123]: # =====
In [124]:
Out[124]: None
In [125]: # =====
In [126]:
Out[126]: None
In [127]: # =====
In [128]:
Out[128]: None
In [129]: # =====
In [130]:
Out[130]: None
In [131]: # =====
In [132]:
Out[132]: None
In [133]: # =====
In [134]:
Out[134]: None
In [135]: # =====
In [136]:
Out[136]: None
In [137]: # =====
In [138]:
Out[138]: None
In [139]: # =====
In [140]:
Out[140]: None
In [141]: # =====
In [142]:
Out[142]: None
In [143]: # =====
In [144]:
Out[144]: None
In [145]: # =====
In [146]:
Out[146]: None
In [147]: # =====
In [148]:
Out[148]: None
In [149]: # =====
In [150]:
Out[150]: None
In [151]: # =====
In [152]:
Out[152]: None
In [153]: # =====
In [154]:
Out[154]: None
In [155]: # =====
In [156]:
Out[156]: None
In [157]: # =====
In [158]:
Out[158]: None
In [159]: # =====
In [160]:
Out[160]: None
In [161]: # =====
In [162]:
Out[162]: None
In [163]: # =====
In [164]:
Out[164]: None
In [165]: # =====
In [166]:
Out[166]: None
In [167]: # =====
In [168]:
Out[168]: None
In [169]: # =====
In [170]:
Out[170]: None
In [171]: # =====
In [172]:
Out[172]: None
In [173]: # =====
In [174]:
Out[174]: None
In [175]: # =====
In [176]:
Out[176]: None
In [177]: # =====
In [178]:
Out[178]: None
In [179]: # =====
In [180]:
Out[180]: None
In [181]: # =====
In [182]:
Out[182]: None
In [183]: # =====
In [184]:
Out[184]: None
In [185]: # =====
In [186]:
Out[186]: None
In [187]: # =====
In [188]:
Out[188]: None
In [189]: # =====
In [190]:
Out[190]: None
In [191]: # =====
In [192]:
Out[192]: None
In [193]: # =====
In [194]:
Out[194]: None
In [195]: # =====
In [196]:
Out[196]: None
In [197]: # =====
In [198]:
Out[198]: None
In [199]: # =====
In [200]:
Out[200]: None
In [201]: # =====
In [202]:
Out[202]: None
In [203]: # =====
In [204]:
Out[204]: None
In [205]: # =====
In [206]:
Out[206]: None
In [207]: # =====
In [208]:
Out[208]: None
In [209]: # =====
In [210]:
Out[210]: None
In [211]: # =====
In [212]:
Out[212]: None
In [213]: # =====
In [214]:
Out[214]: None
In [215]: # =====
In [216]:
Out[216]: None
In [217]: # =====
In [218]:
Out[218]: None
In [219]: # =====
In [220]:
Out[220]: None
In [221]: # =====
In [222]:
Out[222]: None
In [223]: # =====
In [224]:
Out[224]: None
In [225]: # =====
In [226]:
Out[226]: None
In [227]: # =====
In [228]:
Out[228]: None
In [229]: # =====
In [230]:
Out[230]: None
In [231]: # =====
In [232]:
Out[232]: None
In [233]: # =====
In [234]:
Out[234]: None
In [235]: # =====
In [236]:
Out[236]: None
In [237]: # =====
In [238]:
Out[238]: None
In [239]: # =====
In [240]:
Out[240]: None
In [241]: # =====
In [242]:
Out[242]: None
In [243]: # =====
In [244]:
Out[244]: None
In [245]: # =====
In [246]:
Out[246]: None
In [247]: # =====
In [248]:
Out[248]: None
In [249]: # =====
In [250]:
Out[250]: None
In [251]: # =====
In [252]:
Out[252]: None
In [253]: # =====
In [254]:
Out[254]: None
In [255]: # =====
In [256]:
Out[256]: None
In [257]: # =====
In [258]:
Out[258]: None
In [259]: # =====
In [260]:
Out[260]: None
In [261]: # =====
In [262]:
Out[262]: None
In [263]: # =====
In [264]:
Out[264]: None
In [265]: # =====
In [266]:
Out[266]: None
In [267]: # =====
In [268]:
Out[268]: None
In [269]: # =====
In [270]:
Out[270]: None
In [271]: # =====
In [272]:
Out[272]: None
In [273]: # =====
In [274]:
Out[274]: None
In [275]: # =====
In [276]:
Out[276]: None
In [277]: # =====
In [278]:
Out[278]: None
In [279]: # =====
In [280]:
Out[280]: None
In [281]: # =====
In [282]:
Out[282]: None
In [283]: # =====
In [284]:
Out[284]: None
In [285]: # =====
In [286]:
Out[286]: None
In [287]: # =====
In [288]:
Out[288]: None
In [289]: # =====
In [290]:
Out[290]: None
In [291]: # =====
In [292]:
Out[292]: None
In [293]: # =====
In [294]:
Out[294]: None
In [295]: # =====
In [296]:
Out[296]: None
In [297]: # =====
In [298]:
Out[298]: None
In [299]: # =====
In [300]:
Out[300]: None
In [301]: # =====
In [302]:
Out[302]: None
In [303]: # =====
In [304]:
Out[304]: None
In [305]: # =====
In [306]:
Out[306]: None
In [307]: # =====
In [308]:
Out[308]: None
In [309]: # =====
In [310]:
Out[310]: None
In [311]: # =====
In [312]:
Out[312]: None
In [313]: # =====
In [314]:
Out[314]: None
In [315]: # =====
In [316]:
Out[316]: None
In [317]: # =====
In [318]:
Out[318]: None
In [319]: # =====
In [320]:
Out[320]: None
In [321]: # =====
In [322]:
Out[322]: None
In [323]: # =====
In [324]:
Out[324]: None
In [325]: # =====
In [326]:
Out[326]: None
In [327]: # =====
In [328]:
Out[328]: None
In [329]: # =====
In [330]:
Out[330]: None
In [331]: # =====
In [332]:
Out[332]: None
In [333]: # =====
In [334]:
Out[334]: None
In [335]: # =====
In [336]:
Out[336]: None
In [337]: # =====
In [338]:
Out[338]: None
In [339]: # =====
In [340]:
Out[340]: None
In [341]: # =====
In [342]:
Out[342]: None
In [343]: # =====
In [344]:
Out[344]: None
In [345]: # =====
In [346]:
Out[346]: None
In [347]: # =====
In [348]:
Out[348]: None
In [349]: # =====
In [350]:
Out[350]: None
In [351]: # =====
In [352]:
Out[352]: None
In [353]: # =====
In [354]:
Out[354]: None
In [355]: # =====
In [356]:
Out[356]: None
In [357]: # =====
In [358]:
Out[358]: None
In [359]: # =====
In [360]:
Out[360]: None
In [361]: # =====
In [362]:
Out[362]: None
In [363]: # =====
In [364]:
Out[364]: None
In [365]: # =====
In [366]:
Out[366]: None
In [367]: # =====
In [368]:
Out[368]: None
In [369]: # =====
In [370]:
Out[370]: None
In [371]: # =====
In [372]:
Out[372]: None
In [373]: # =====
In [374]:
Out[374]: None
In [375]: # =====
In [376]:
Out[376]: None
In [377]: # =====
In [378]:
Out[378]: None
In [379]: # =====
In [380]:
Out[380]: None
In [381]: # =====
In [382]:
Out[382]: None
In [383]: # =====
In [384]:
Out[384]: None
In [385]: # =====
In [386]:
Out[386]: None
In [387]: # =====
In [388]:
Out[388]: None
In [389]: # =====
In [390]:
Out[390]: None
In [391]: # =====
In [392]:
Out[392]: None
In [393]: # =====
In [394]:
Out[394]: None
In [395]: # =====
In [396]:
Out[396]: None
In [397]: # =====
In [398]:
Out[398]: None
In [399]: # =====
In [400]:
Out[400]: None
In [401]: # =====
In [402]:
Out[402]: None
In [403]: # =====
In [404]:
Out[404]: None
In [405]: # =====
In [406]:
Out[406]: None
In [407]: # =====
In [408]:
Out[408]: None
In [409]: # =====
In [410]:
Out[410]: None
In [411]: # =====
In [412]:
Out[412]: None
In [413]: # =====
In [414]:
Out[414]: None
In [415]: # =====
In [416]:
Out[416]: None
In [417]: # =====
In [418]:
Out[418]: None
In [419]: # =====
In [420]:
Out[420]: None
In [421]: # =====
In [422]:
Out[422]: None
In [423]: # =====
In [424]:
Out[424]: None
In [425]: # =====
In [426]:
Out[426]: None
In [427]: # =====
In [428]:
Out[428]: None
In [429]: # =====
In [430]:
Out[430]: None
In [431]: # =====
In [432]:
Out[432]: None
In [433]: # =====
In [434]:
Out[434]: None
In [435]: # =====
In [436]:
Out[436]: None
In [437]: # =====
In [438]:
Out[438]: None
In [439]: # =====
In [440]:
Out[440]: None
In [441]: # =====
In [442]:
Out[442]: None
In [443]: # =====
In [444]:
Out[444]: None
In [445]: # =====
In [446]:
Out[446]: None
In [447]: # =====
In [448]:
Out[448]: None
In [449]: # =====
In [450]:
Out[450]: None
In [451]: # =====
In [452]:
Out[452]: None
In [453]: # =====
In [454]:
Out[454]: None
In [455]: # =====
In [456]:
Out[456]: None
In [457]: # =====
In [458]:
Out[458]: None
In [459]: # =====
In [460]:
Out[460]: None
In [461]: # =====
In [462]:
Out[462]: None
In [463]: # =====
In [464]:
Out[464]: None
In [465]: # =====
In [466]:
Out[466]: None
In [467]: # =====
In [468]:
Out[468]: None
In [469]: # =====
In [470]:
Out[470]: None
In [471]: # =====
In [472]:
Out[472]: None
In [473]: # =====
In [474]:
Out[474]: None
In [475]: # =====
In [476]:
Out[476]: None
In [477]: # =====
In [478]:
Out[478]: None
In [479]: # =====
In [480]:
Out[480]: None
In [481]: # =====
In [482]:
Out[482]: None
In [483]: # =====
In [484]:
Out[484]: None
In [485]: # =====
In [486]:
Out[486]: None
In [487]: # =====
In [488]:
Out[488]: None
In [489]: # =====
In [490]:
Out[490]: None
In [491]: # =====
In [492]:
Out[492]: None
In [493]: # =====
In [494]:
Out[494]: None
In [495]: # =====
In [496]:
Out[496]: None
In [497]: # =====
In [498]:
Out[498]: None
In [499]: # =====
In [500]:
Out[500]: None
In [501]: # =====
In [502]:
Out[502]: None
In [503]: # =====
In [504]:
Out[504]: None
In [505]: # =====
In [506]:
Out[506]: None
In [507]: # =====
In [508]:
Out[508]: None
In [509]: # =====
In [510]:
Out[510]: None
In [511]: # =====
In [512]:
Out[512]: None
In [513]: # =====
In [514]:
Out[514]: None
In [515]: # =====
In [516]:
Out[516]: None
In [517]: # =====
In [518]:
Out[518]: None
In [519]: # =====
In [520]:
Out[520]: None
In [521]: # =====
In [522]:
Out[522]: None
In [523]: # =====
In [524]:
Out[524]: None
In [525]: # =====
In [526]:
Out[526]: None
In [527]: # =====
In [528]:
Out[528]: None
In [529]: # =====
In [530]:
Out[530]: None
In [531]: # =====
In [532]:
Out[532]: None
In [533]: # =====
In [534]:
Out[534]: None
In [535]: # =====
In [536]:
Out[536]: None
In [537]: # =====
In [538]:
Out[538]: None
In [539]: # =====
In [540]:
Out[540]: None
In [541]: # =====
In [542]:
Out[542]: None
In [543]: # =====
In [544]:
Out[544]: None
In [545]: # =====
In [546]:
Out[546]: None
In [547]: # =====
In [548]:
Out[548]: None
In [549]: # =====
In [550]:
Out[550]: None
In [551]: # =====
In [552]:
Out[552]: None
In [553]: # =====
In [554]:
Out[554]: None
In [555]: # =====
In [556]:
Out[556]: None
In [557]: # =====
In [558]:
Out[558]: None
In [559]: # =====
In [560]:
Out[560]: None
In [561]: # =====
In [562]:
Out[562]: None
In [563]: # =====
In [564]:
Out[564]: None
In [565]: # =====
In [566]:
Out[566]: None
In [567]: # =====
In [568]:
Out[568]: None
In [569]: # =====
In [570]:
Out[570]: None
In [571]: # =====
In [572]:
Out[572]: None
In [573]: # =====
In [574]:
Out[574]: None
In [575]: # =====
In [576]:
Out[576]: None
In [577]: # =====
In [578]:
Out[578]: None
In [579]: # =====
In [580]:
Out[580]: None
In [581]: # =====
In [582]:
Out[582]: None
In [583]: # =====
In [584]:
Out[584]: None
In [585]: # =====
In [586]:
Out[586]: None
In [587]: # =====
In [588]:
Out[588]: None
In [589]: # =====
In [590]:
Out[590]: None
In [591]: # =====
In [592]:
Out[592]: None
In [593]: # =====
In [594]:
Out[594]: None
In [595]: # =====
In [596]:
Out[596]: None
In [597]: # =====
In [598]:
Out[598]: None
In [599]: # =====
In [600]:
Out[600]: None
In [601]: # =====
In [602]:
Out[602]: None
In [603]: # =====
In [604]:
Out[604]: None
In [605]: # =====
In [606]:
Out[606]: None
In [607]: # =====
In [608]:
Out[608]: None
In [609]: # =====
In [610]:
Out[610]: None
In [611]: # =====
In [612]:
Out[612]: None
In [613]: # =====
In [614]:
Out[614]: None
In [615]: # =====
In [616]:
Out[616]: None
In [617]: # =====
In [618]:
Out[618]: None
In [619]: # =====
In [620]:
Out[620]: None
In [621]: # =====
In [622]:
Out[622]: None
In [623]: # =====
In [624]:
Out[624]: None
In [625]: # =====
In [626]:
Out[626]: None
In [627]: # =====
In [628]:
Out[628]: None
In [629]: # =====
In [630]:
Out[630]: None
In [631]: # =====
In [632]:
Out[632]: None
In [633]: # =====
In [634]:
Out[634]: None
In [635]: # =====
In [636]:
Out[636]: None
In [637]:
```

So, I made a deep copy and I saved it as cars and we are going to be using cars to work here after. So, let us start by looking at the structure of the data, I am using the cars.info. So, this gives you the structure of the data. Now this tells me that is the data frame it has entries ranging from 0 to 50,000 and the total number of columns are 19 here what you also have is a number of non null objects and what is the data type of each of the column.

So, date crawled has 50,001 non null object it is an object data type similarly you have name for seller type, offerType, price and abtest all these are full and if you take vehicle type that is 44,813 non null entries. So, you have some missing values there and similarly even in gearbox model and fuel type this will give you just of all the column names and under each column the number of non null entries and what is the data type of each of the columns.

(Refer Slide Time: 04:49)



The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays the following Python script:

```

13 import numpy as np
14 import seaborn as sns
15
16 # ****
17 # Setting dimensions for plot
18 # ****
19
20 sns.set(rc={'figure.figsize':(11.7
21
22
23 # ****
24 # Reading CSV file
25 # ****
26 cars_data=pd.read_csv('cars_sample')
27
28 # ****
29 # Creating copy
30 # ****
31 cars=cars_data.copy()
32
33 # ****
34 # Structure of the dataset
35 # ****
36 cars.info()
37
38 # ****

```

The Variable explorer window shows the following data frame information:

| Name      | Type      | Size        | Value                                                                      |
|-----------|-----------|-------------|----------------------------------------------------------------------------|
| cars      | Dataframe | (50001, 19) | Column names: dateCrawled, name, seller, offerType, price, abtest, veh ... |
| cars_data | Dataframe | (50001, 19) | Column names: dateCrawled, name, seller, offerType, price, abtest, veh ... |

The Python console window shows the output of the cars.info() command:

```

vehicleType 44811 non-null object
yearOfRegistration 50001 non-null int64
gearbox 47277 non-null object
powerPS 50001 non-null int64
model 47243 non-null object
kilometer 50001 non-null int64
notRepairedDamage 40285 non-null object
dateCreated 50001 non-null object
postalCode 50001 non-null int64
lastSeen 50001 non-null object
dtyps: int64(6),object(13)
memory usage: 7.12 MB

```

So, far as we have 6 columns which are of int64 data type and there are 13 column which are of object data type. So, now, let us summarize the data I am going to be using the function cars.describe for this.

(Refer Slide Time: 05:03)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
29 # Creating copy
30 # =====
31 cars=cars_data.copy()
32
33 # =====
34 # Structure of the dataset
35 # =====
36 cars.info()
37
38 # =====
39 # Summarizing data
40 # =====
41
42 cars.describe()
43 pd.set_option('display.float_format', lambda x: '%.3f' % x)
44 cars.describe()
45
46
47 # To display maximum set of columns
48 pd.set_option('display.max_columns', 500)
49 cars.describe()
50
51
52 # =====
53 # Dropping unwanted columns
54 # =====
```

The Variable explorer window shows two DataFrames: 'cars' and 'cars\_data'. Both have 50001 rows and 19 columns. The columns are: dateCrawled, name, seller, offerType, price, abtest, veh...

The IPython console window shows the output of the 'cars.describe()' command. It includes summary statistics for each column: count, mean, std, min, 25%, 50%, 75%, and max. The output is in scientific notation.

So, when I used cars.describe you can see that there is a. So, when I use a we are doing a.describe function, you can see the summary is given only for a few variables and it is not given for every variable there are some dots that you see in between; now to get rid of this you can use the.

So, when you do a summary you can see the summary is given when you do a.describe, you can see the description gives you the output in a scientific notation what you also see is that it displays the summary of only a few variables and the remaining variables are displayed as '...'. So, let us see how to get rid of the scientific notation first, I am going to use the function.set\_option.

Within parenthesis what you have to give is displayed.float format because these are all float values and it is for these that you want to change the output. Now there is a lambda function that we are declaring inside. So, what I am saying here is converted to a three decimal place float value. So, let us just run this function again and then let us rerun cars.describe.

(Refer Slide Time: 06:23)

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** E:\GDPL-GITAAN\IPTEL\Python for Data Science\Case Study\Regression-Case Study.py
- Code:**

```
29 # Creating copy
30 # =====
31 cars=cars_data.copy()
32
33 # =====
34 # Structure of the dataset
35 # =====
36 cars.info()
37
38 # =====
39 # Summarizing data
40 # =====
41
42 cars.describe()
43 pd.set_option('display.float_format', lambda x: '%.3f' % x)
44 cars.describe()
45
46
47 # To display maximum set of columns
48 pd.set_option('display.max_columns', 500)
49 cars.describe()
50
51
52 # =====
53 # Dropping unwanted columns
```
- Variable explorer:** Shows two DataFrames: `cars` (50001 rows, 19 columns) and `cars_data` (50001 rows, 19 columns). Both have columns: dateCrawled, name, seller, offerType, price, abtest, veh...
- Python console:**
  - In [48]: `cars.describe()`
  - Out[48]:

|       | price        | yearOfRegistration | powerPS   | kilometer | ...       |
|-------|--------------|--------------------|-----------|-----------|-----------|
| count | 50001.000    | 50001.000          | 50001.000 | 50001.000 | ...       |
| mean  | 4599.865     | 2005.544           | ...       | 5.744     | 58775.217 |
| std   | 85818.470    | 122.992            | ...       | 3.711     | 25743.762 |
| min   | 0.000        | 1000.000           | ...       | 0.000     | 1867.000  |
| 25%   | 1150.000     | 1990.000           | ...       | 3.000     | 36559.000 |
| 50%   | 2950.000     | 2087.000           | ...       | 6.000     | 47304.000 |
| 75%   | 7100.000     | 2089.000           | ...       | 9.000     | 73404.000 |
| max   | 12345678.000 | 9999.000           | ...       | 12.000    | 99999.000 |
  - [8 rows x 6 columns]
  - In [49]:
- System Status:** Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 48, Column: 38, Memory: 44 %

So, here you can see all the values have been rounded off to three decimal places, but we still and not able to get the description for all the columns. So, for that you need to use again the same function `pd.set_option`. So, within parenthesis you can give the parameter `display.max_columns` and it will display the maximum number of columns and the next parameter is basically how many of a columns you wanted to be display.

So, I have given a big number here 500, but usually you will not be seen 500 columns in a data, but this is just to set the maximum number of columns that can be displayed.

(Refer Slide Time: 07:03)

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** E:\GDPL-GITAAN\IPTEL\Python for Data Science\Case Study\Regression-Case Study.py
- Code:**

```
29 # Creating copy
30 # =====
31 cars=cars_data.copy()
32
33 # =====
34 # Structure of the dataset
35 # =====
36 cars.info()
37
38 # =====
39 # Summarizing data
40 # =====
41
42 cars.describe()
43 pd.set_option('display.float_format', lambda x: '%.3f' % x)
44 cars.describe()
45
46
47 # To display maximum set of columns
48 pd.set_option('display.max_columns', 500)
49 cars.describe()
50
51
52 # =====
53 # Dropping unwanted columns
54 # =====
```
- Variable explorer:** Shows two DataFrames: `cars` (50001 rows, 19 columns) and `cars_data` (50001 rows, 19 columns). Both have columns: dateCrawled, name, seller, offerType, price, abtest, veh...
- Python console:**
  - In [49]: `cars.describe()`
  - Out[49]:

|       | price        | yearOfRegistration | powerPS   | kilometer  | ...       |
|-------|--------------|--------------------|-----------|------------|-----------|
| count | 50001.000    | 50001.000          | 50001.000 | 50001.000  | ...       |
| mean  | 4599.865     | 2005.544           | ...       | 5.744      | 58775.217 |
| std   | 85818.470    | 122.992            | 230.560   | 46295.234  |           |
| min   | 0.000        | 1000.000           | 0.000     | 5000.000   |           |
| 25%   | 1150.000     | 1990.000           | 69.000    | 125000.000 |           |
| 50%   | 2950.000     | 2087.000           | 105.000   | 150000.000 |           |
| 75%   | 7100.000     | 2089.000           | 158.000   | 150000.000 |           |
| max   | 12345678.000 | 9999.000           | 19312.000 | 150000.000 |           |
  - monthOfRegistration postalCode
  - count 50001.000 50001.000
  - mean 5.744 58775.217
  - std 3.711 25743.762
  - In [50]:
- System Status:** Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 50, Column: 1, Memory: 43 %

So, let us do that and let us read on cars.describe. So, now, you can see that all your columns are displayed here and you have a summary for all of it. Now price tells you that there are 50,000. So, under price if you check the count there are 50,001 records.

So, all the records having field and the mean is about 6559 and there is a standard there is a huge standard deviation, it is about 85,818 the minimum value is 0 the first quartile value is 1150, the second quartile which is also the median is 2950, the third quartile is 7190, but the maximum is very very huge and if you look at the difference between the mean and the median which is the second quartile, you can see there is a huge difference between the mean and the median there is itself shows you that price is very very secured.

And if you take care of registration you have 50,001 non null values the mean of year of registration is 2005, but this does not make sense because year of registration is an integer and it cannot be rounded off to a decimal.

But if you take the minimum of it you can see there are years ranging from 1000. Similarly, if you look at powerPS it again has 50,001 non null values, the mean being around 117 and if you look at the median, the mean being around 117, but if you look at the minimum value the minimum is 0 which does not make sense again and the first quartile is at around 70, the second quarter which is the median is around 105 and maximum is very very high, the maximum is around 19312 horse power. Now then you have kilometer, kilometer has again 50,001 non null entries, it is mean is about 1,25,613 kilometers, the minimum is again 5,000, the maximum being 1,50,000.

(Refer Slide Time: 08:57)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains a script named 'Regression-Case Study.py' with the following content:

```
29 # Creating copy
30 cars=cars_data.copy()
31
32
33 # =====
34 # Structure of the dataset
35 # =====
36 cars.info()
37
38 # =====
39 # Summarizing data
40 # =====
41
42 cars.describe()
43 pd.set_option('display.float_format', lambda x: '%.1f' % x)
44 cars.describe()
45
46
47 # To display maximum set of columns
48 pd.set_option('display.max_columns', 500)
49 cars.describe()
50
51
52 # =====
53 # Dropping unwanted columns
```

The variable explorer shows two DataFrames: 'cars' and 'cars\_data'. The 'cars' DataFrame has 50001 rows and 19 columns, with columns including dateCrawled, name, seller, offerType, price, attest, veh... The 'cars\_data' DataFrame has 50001 rows and 19 columns, with columns including dateCrawled, name, seller, offerType, price, attest, veh... The Python console displays summary statistics for the 'monthOfRegistration' column:

| count | 50001.000 |
|-------|-----------|
| mean  | 5.744     |
| std   | 50775.217 |
| min   | 0.000     |
| 25%   | 3.000     |
| 50%   | 5.000     |
| 75%   | 9.000     |
| max   | 12.000    |

The history log shows the command 'In [13]:'.

Next you have month of registration. Month of registration does not have any missing values the minimum again is 0. But 0 does not make sense and postal code is something that we will be getting rid off because we are not going to be using for this case study, but you can you are free to use this variable if you want to do a more spatial based analysis right. So, again even for postal code, they are all integers and you cannot have decimal values.

(Refer Slide Time: 09:23)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains a script named 'Regression-Case Study.py' with the following content:

```
47 # To display maximum set of columns
48 pd.set_option('display.max_columns', 500)
49 cars.describe()
50
51
52 # =====
53 # Dropping unwanted columns
54 # =====
55
56 col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
57 cars=cars.drop(columns=col, axis=1)
58
59 # =====
60 # Removing duplicate records
61 # =====
62
63 cars.drop_duplicates(keep='first', inplace=True)
64 #470 duplicate records
65
66 # =====
67 # Data cleaning
68 # =====
69
70 # No. of missing values in each column
71 cars.isnull().sum()
72
```

The variable explorer shows three variables: 'cars' (Dataframe, 49531 rows, 14 columns), 'cars\_data' (Dataframe, 50001 rows, 19 columns), and 'col' (list, 5 elements: ['name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen']). The Python console displays the sum of missing values for each column:

|                     | cars.isnull().sum() |
|---------------------|---------------------|
| dateCrawled         | 0                   |
| offerType           | 0                   |
| price               | 0                   |
| attest              | 0                   |
| yearOfRegistration  | 5352                |
| garage              | 0                   |
| powerPS             | 0                   |
| seats               | 2730                |
| kilometer           | 0                   |
| monthOfRegistration | 0                   |
| fuelType            | 4467                |
| brand               | 0                   |
| notRepairedDamage   | 9640                |

The history log shows the command 'In [17]:'.

So, let us just start by dropping unwanted columns up front. Now you have variables like name date crawled date created postal code and last seen, now we are not going to use these variables for our analysis and I am going to just drop them. So, I am creating a list of all the variable names here and once I run it all of them are stored under col and I am just going to use the.drop function and to drop them.

So, I do a cars.drop and I say columns is equal to col. So, these are the columns I want to drop, since these are columns I am giving the axis as 1. So, from 19 you can see the number of columns have come to 14, I have dropped 5 columns here. Now, we are going to see if there are any duplicate records in our data right and if there are any duplicate records, we are just going to keep the first occurrence of that such records. So, we started out with 50,001 records let us see when you do a.drop\_duplicates how many are we will use. So, one 50,001 it is come down to 49531. Now that we have removed it let us get into data cleaning now.

So, my first task is to compute the number of missing values under each column, now from the.info function we saw not all column were filled right. So, now, we are going to find the number of missing values and each column and later on we will come up with the method of how to logically fill such missing values. So, the first step is to basically calculate the number of missing values under each column. So, I am using the isnull function. So, isnull will basically return a data which is true or false.

It will mark the cells which are missing with true, but I am going to sum of the occurrences of the trues. So, I am just going to do it.sum for it. So, if you run the function you can see on my right that seller of a type price, ab test do not have any missing values now vehicle type has 5152 missing values, year of registration is again full, gearbox; however, has 2765 records missing.

And under model you can see you have about 2730 records missing. And under fuel you can see you have about 4467 records missing and about 9640 records are missing and not repaired or damaged. So, this is a column wise count of the number of cells that are missing.

(Refer Slide Time: 12:01)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays Python code for data cleaning and analysis. The code includes dropping columns, removing duplicates, and calculating year-wise counts. The variable explorer window shows the structure of the 'cars' DataFrame, including columns like 'seller', 'offerType', 'name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen', 'price', 'atmt', 'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model', 'kilometer', 'monthOfRegistration', 'fuelype', 'brand', and 'notRepairedDamage'. The Python console window shows the execution of the code, including the calculation of 'yearwise\_count' and its sorting.

```
56 col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
57 cars=cars.drop(columns=col, axis=1)
58
59 ##### Removing duplicate records
60 #####
61 # Removing duplicate records
62
63 cars.drop_duplicates(keep='first',inplace=True)
64 ##70 duplicate records
65
66 #####
67 # Data cleaning
68 #####
69
70 # No. of missing values in each column
71 cars.isnull().sum()
72
73
74 # Variable yearOfRegistration
75 yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
76 sum(cars['yearOfRegistration'] > 2018)
77 sum(cars['yearOfRegistration'] < 1950)
78 sns.regplot(x='yearOfRegistration', y='price', scatter=True,
79 fit_reg=False, data=cars)
80 # Working range- 1950 and 1975
81
```

So, now, let us just take year of registration first. So, in 975 what I am trying to do is I am trying to find year wise count based on this variable. Now, you can use the.value\_counts here and I am sorting the index to make sure that, here year of registration is sorted based on the years and not on the counts. So, by default if you do a.value\_counts, you would have seen that.

So, it gives you the category with the highest frequency on top, but I do not want that I basically wanted sorted placed on the years. So, let us just run this, this is too big output. So, I am saving it onto year wise under count variable.

(Refer Slide Time: 12:39)

| Index | yearOfRegistration |
|-------|--------------------|
| 1800  | 8                  |
| 1255  | 1                  |
| 1500  | 2                  |
| 1910  | 14                 |
| 1928  | 1                  |
| 1929  | 1                  |
| 1933  | 1                  |
| 1934  | 1                  |
| 1936  | 4                  |
| 1938  | 1                  |
| 1940  | 1                  |
| 1941  | 1                  |
| 1943  | 1                  |
| 1945  | 2                  |
| 1947  | 2                  |
| 1950  | 4                  |
| 1951  | 4                  |
| 1952  | 3                  |
| 1953  | 2                  |
| 1954  | 1                  |
| 1955  | 1                  |

Now, if you open and see. So, let us just see what this data frame has. It has index and it has year of registration index is the set of all years and year of registration on the that column you basically have the frequencies of each of these years. So, you have years that are ranging from 1000 and then you have year from 1910 and what you can also see if you scroll down is that, you have years which are in the future that is which are after 2019.

So, most of these do not make sense so, that is considerable amount of cleaning that we will have to do for this column. So, let us see how to come up with the strategy to clean them.

(Refer Slide Time: 13:19)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
56 col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
57 cars=cars.drop(columns=col, axis=1)
58
59 # =====
60 # Removing duplicate records
61 # =====
62
63 cars.drop_duplicates(keep='first',inplace=True)
64 ##70 duplicate records
65
66 # =====
67 # Data cleaning
68 # =====
69
70# No. of missing values in each column
71 cars.isnull().sum()
72
73
74# Variable yearOfRegistration
75 yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
76 sum(cars['yearOfRegistration'] > 2018)
77 sum(cars['yearOfRegistration'] < 1950)
78 sns.regplot(x='yearOfRegistration', y='price', scatter=True,
79 fit_reg=False, data=cars)
80 # Working range- 1950 and 2018
81
```

The variable explorer shows the following data structures:

| Name           | Type      | Size        | Value                                                              |
|----------------|-----------|-------------|--------------------------------------------------------------------|
| cars           | DataFrame | (49551, 14) | Column names: seller, offerType.                                   |
| cars_data      | DataFrame | (50001, 19) | Column names: dateCrawled, name,                                   |
| col            | list      | 5           | ('name', 'dateCrawled', 'dateCreated', 'yearwise_count', 'Series') |
| yearwise_count | Series    | (97,)       | Series object of pandas.core.series module                         |

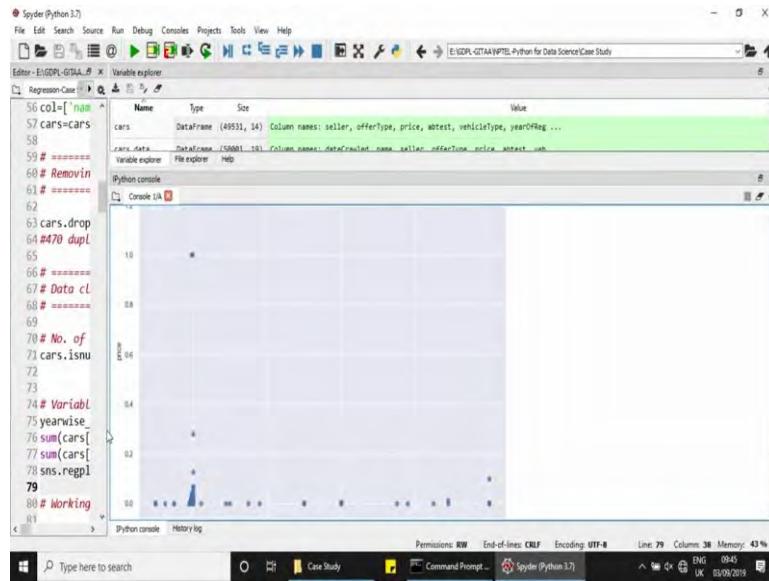
The Python console window shows the execution of the code, including the creation of a scatter plot:

```
In [17]: yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
In [18]: sum(cars['yearOfRegistration'] > 2018)
Out[18]: 26
In [19]: sum(cars['yearOfRegistration'] < 1950)
Out[19]: 38
In [20]: sns.regplot(x='yearOfRegistration', y='price', scatter=True,
... fit_reg=False, data=cars)
Out[20]: AxesSubplot at 0x1b0f5c4fac>
```

So, if you look at the sum of records which are greater than 2018 then that is only 26 right. This is 2 lesser number and we cannot predict in the future. We are in 2019, but I am not considering that because it is occurred only twice and it would not make sense to consider it. So, if you look at the number of cars where registration is greater than 2018.

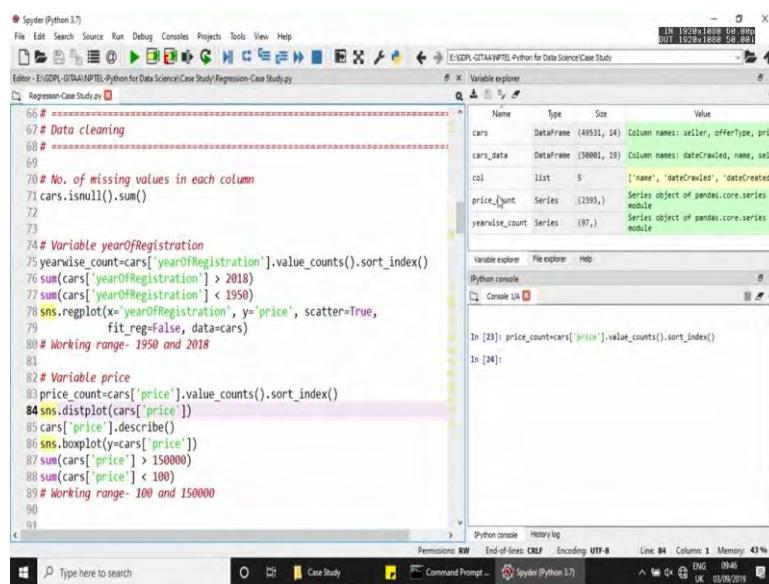
Now, you have about 26 of them 26 records have a year that is greater than 2018. Now similarly if you do for a lower and check what you can also see is, I am taking the limit 1950. So, if you look at the cars that are made before or that are registered before in 1950, then you have about 38 records of that kind right. Now these are too less correct and they are going to smear or effect or you know sway the effect of the model. So, we are going to get rid of them. So, the working range that I am going to set is between 1950 and 2018. So, just to reconfirm this if you do a scatter plot I am using the sns.regplot.

(Refer Slide Time: 14:31)



If you look at this scatter plot, I have the year of registration on x axis and I have the price on the y axis, let me just drag it to the size just to show you a better plot. So, you can see that the plot is not explaining anything you have all only you have only dots here and there and that is because there are values which are very very high and they are just smearing out the effect on the other points. So, we have to clean this column to really understand what is the effect of year of registration on price.

(Refer Slide Time: 15:09)



So, that is as far as year of registration goes, the next is the variable price. Now if you again do a.value counts and then you sort based on the ascending order of the price and I am just saving it on to the variable price\_count.

(Refer Slide Time: 15:25)

| Index | price |
|-------|-------|
| 15888 | 1     |
| 15889 | 2     |
| 15890 | 3     |
| 15891 | 61    |
| 15892 | 13    |
| 15893 | 2     |
| 15894 | 26    |
| 15895 | 1     |
| 15896 | 2     |
| 15897 | 40    |
| 15898 | 38    |
| 15899 | 1     |
| 16100 | 3     |
| 16101 | 2     |
| 16102 | 19    |
| 16103 | 4     |
| 16104 | 1     |
| 16299 | 5     |
| 16390 | 12    |
| 16391 | 4     |
| 16392 | 4     |

So, the left is basically the index and that is the value of the price of the car and the right is basically the frequencies of each occurrence. Now that is a too huge range. So, we have to do tweak it down and even price equal to 0 we will not make any sense, I mean it can be because if you want to just sell it for a few dollars or a few for a very low amount you can.

But then we are looking to generalize this model for a workable range of data and these values are these values can be very very extreme right. So, there is nothing wrong in selling a car for a lower price, but then we would like to generalize a model that would come up with the better prediction, but however, these if you have these values when they are going to smear the effect out. So, we are going to do check on this as well and we are going to arrive at a working range that is easy for us to understand right.

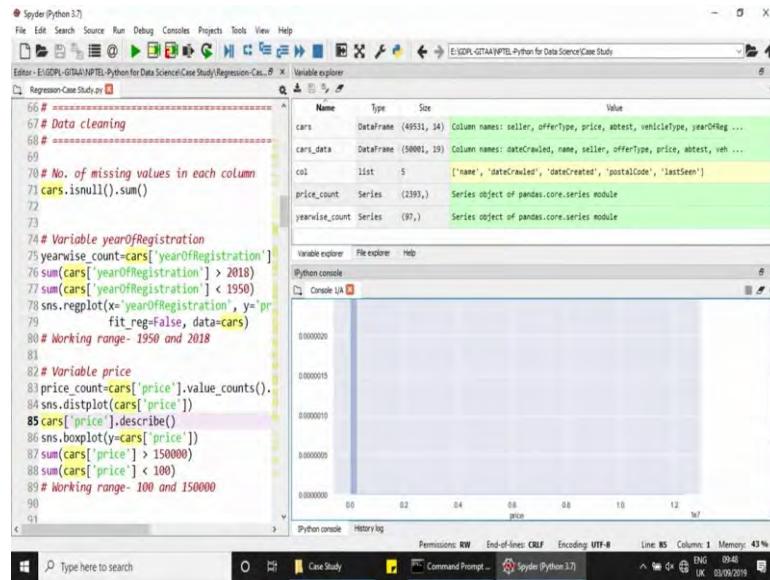
(Refer Slide Time: 16:29)

The screenshot shows the Spyder IDE interface with a Python script titled 'Regression-Car Study.py'. The code performs data cleaning, counts missing values, and analyzes variables like yearOfRegistration and price. The Variable explorer on the right shows the data types and sizes of various variables.

```
=====#
Data cleaning
=====#
No. of missing values in each column
cars.isnull().sum()
Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
 fit_reg=False, data=cars)
Working range- 1950 and 2018
Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
Working range- 100 and 150000
q1
```

So, now if you do histogram which is using the distplot right.

(Refer Slide Time: 16:31)



You can see that because there are a lot of entries under the zero prize category. So, this is something that we have to work on.

(Refer Slide Time: 16:47)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains Python code for data cleaning and analysis. The variable explorer shows several data frames and series. The console output displays the results of a describe() call on the 'price' column, showing statistics like mean (~6567), median (~2950), and range (~0 to ~12567). The status bar at the bottom indicates the current line (Line 86) and column (Column 1).

```
=====
Data cleaning
#
No. of missing values in each column
cars.isnull().sum()
#
#
Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
 fit_reg=False, data=cars)
#
Working range- 1950 and 2018
#
Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
#
Working range- 100 and 150000
q1
```

```
In [28]: cars['price'].describe()
Out[28]:
count 49531.000
mean 6567.228
std 86222.378
min 0.000
25K 1159.000
50K 2950.000
75K 2100.000
max 12567.000
Name: price, dtype: float64
```

So, let us just quickly reiterated by even looking at the describe. So, describe also tells you that the mean is about 6567, the median is way off the medium is around 2950. Now there is a huge difference than this itself accounts for the skewness in the data and the minimum here you can see is again 0 and the maximum value is also given. So, the range is really really wide and we have to narrow it down to come up with the generalized model.

(Refer Slide Time: 17:15)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains Python code for data cleaning and analysis. The variable explorer shows several data frames and series. The console output displays the results of a describe() call on the 'powerPS' column, showing statistics like count (~34) and median (~1748). The status bar at the bottom indicates the current line (Line 93) and column (Column 33).

```
=====
Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
 fit_reg=False, data=cars)
#
Working range- 1950 and 2018
#
Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
#
Working range- 100 and 150000
q1
#
Variable powerPS
power_count=cars['powerPS'].value_counts().sort_index()
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(y='powerPS', x='price', scatter=True,
```

```
In [27]: sum(cars['price'] > 150000)
Out[27]: 34
In [28]: sum(cars['price'] < 100)
Out[28]: 1748
In [29]:
```

So, let us just quickly do a box plot. Now a box plot also tells you that there are some outliers here right if I do a box plot for the price, you can see you cannot even see the box. In fact, what you just see is a line. Now this itself tells you that there are considerable outliers in your data which are very very extreme in nature and hence you are not able to really see the behavior of the variable. So, let us just do a quick check of the range.

So, I am setting the range between 100 to lakh and 50,000 dollars. So, the first is to see how many cars are price above 1,50,000 dollars and that is about 34 and if you do a price check on the lower end you will see that 1748 cars have a price lower than 100, but this is because 100 dollars to lakh and 50s decent range to work with. So, the next variable that we are going to look into is power ps. Now again if you do a values count and then you sort it and now I have saved it under power count.

(Refer Slide Time: 18:27)

| Index | powerPS |
|-------|---------|
| 259   | 1       |
| 260   | 91      |
| 261   | 1       |
| 262   | 2       |
| 264   | 4       |
| 265   | 87      |
| 266   | 1       |
| 268   | 1       |
| 270   | 8       |
| 271   | 13      |
| 272   | 122     |
| 273   | 1       |
| 274   | 1       |
| 275   | 20      |
| 276   | 1       |
| 277   | 4       |
| 278   | 2       |
| 279   | 34      |
| 280   | 49      |
| 281   | 1       |
| 283   | 1       |

So, here again you will see that the power values are on the extreme left and the right hand side is the frequencies of each. So, 5533 records have a power 0; again we have the same problem because the range is too diverse and we cannot really inform much. So, we have to again narrow it down. Somewhere in between you have a lot of occurrences of once some of them are two extreme for instance you have 19,312 horsepower that is very very extreme.

And which is occurred only once and so, we have to narrow down the range of power PS as well. So, let us just do a distplot just to see if there is any skewness yes there is you can see that from here itself because of entries that are of 0. So, this is something that we have to take into account we have to clean it up.

(Refer Slide Time: 19:23)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains Python code for data analysis, specifically working with a 'cars' DataFrame. The code includes filtering for years before 1950 and prices below 100 or 150,000, creating histograms for 'yearOfRegistration' and 'powerPS', and performing descriptive statistics on 'powerPS'. The Variable Explorer panel shows various data structures like DataFrames and Series. The IPython console displays the results of the 'describe()' method for 'powerPS', showing a mean of 116.54, standard deviation of 231.53, and a maximum value of 19312.00.

```

Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\V2PL-GITAAN\TEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study []
Variable yearOfRegistration
75 yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
76 sum(cars['yearOfRegistration'] > 1950)
77 sum(cars['yearOfRegistration'] < 1950)
78 sns.regplot(x='yearOfRegistration', y='price', scatter=True,
79 fit_reg=False, data=cars)
80 # Working range- 1950 and 2018
81
82 # Variable price
83 price_count=cars['price'].value_counts().sort_index()
84 sns.distplot(cars['price'])
85 cars['price'].describe()
86 sns.boxplot(y=cars['price'])
87 sum(cars['price'] > 150000)
88 sum(cars['price'] < 100)
89 # Working range- 100 and 150000
90
91
92 # Variable powerPS
93 power_count=cars['powerPS'].value_counts().sort_index()
94 sns.distplot(cars['powerPS'])
95 cars['powerPS'].describe()
96 sns.boxplot(y=cars['powerPS'])
97 sns.regplot(x='powerPS', y='price', scatter=True,
98 fit_reg=False, data=cars)
99 sum(cars['powerPS'] > 500)

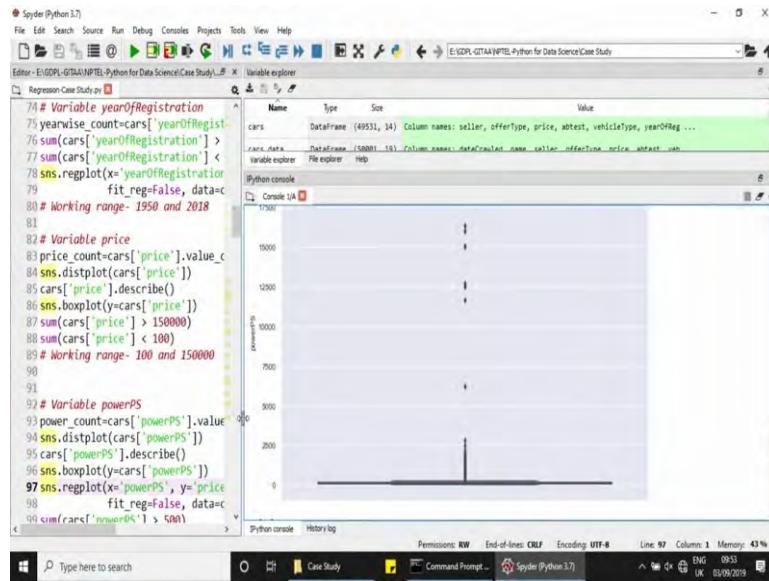
```

And if you do `a.describe()`, this will also show you this skewness we have already done this before just reiterating it here. So, the mean is about 116 and if you look at the minimum it is 0 and if you look at the first quartile it is around 76 it is around 69 and if you look at the median, that is around 105. Though the mean and median are not far away the standard deviation is quite huge.

Now for a mean value of 116 if you are going to deviate by around 200 units then that is a lot right. So, though the mean and the median are not very far, but they are still far there is still some skewness. But if you look at the range of powerPS it is too diverse and you also have values that are around 0.

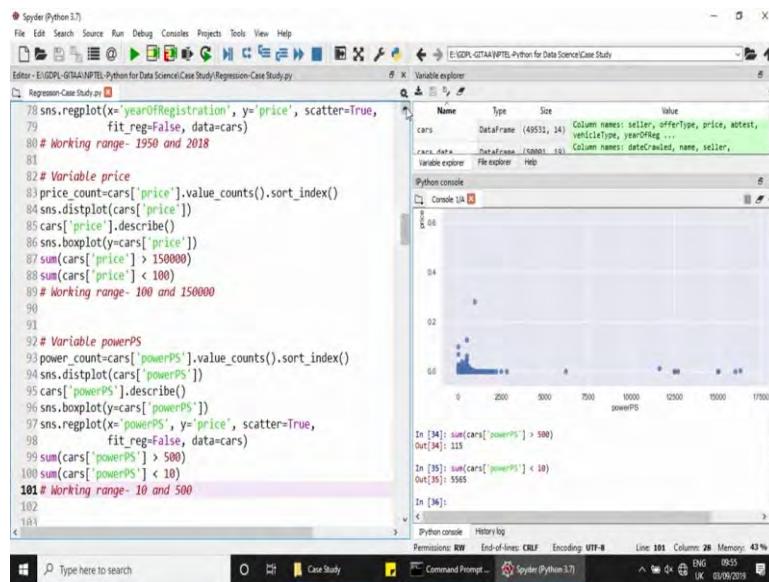
So, you just even if your mean and median are very very close you also have to take into account the standard deviation in the data and in this case it is very very huge right. So, we are going to take up this we are going to tweak the range down, but before we do that let us just quickly see if this is also getting reflected in the box plot.

(Refer Slide Time: 20:27)



This plot is a lot better compared to the price plot that we are earlier did there we were just able to see a line at least here you are able to see a small box. So, yes of course, you have some extreme values here that are actually compressing the box. So, we will have to deal with this, we have to come with a workable range of data. So, let us just plot and see if powerPS has an effect on price before we clean it up.

(Refer Slide Time: 21:03)



Again it is all bundled up together on the lower end and I think that is because of the values that are 0, we have to clean it up before we further do anything with this variable

because otherwise we will not be able to see the effect of this variable on price. So, let us just fixe a range. So, I am fixing a range between 10 and 500, now I have arrived at all these ranges by trial and error also the idea is that you do not want to let go of too many records correct.

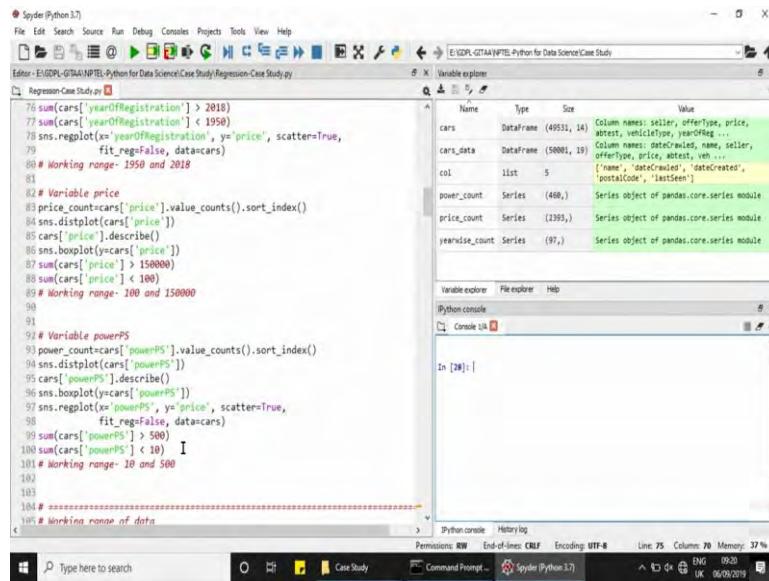
So, I am going to check the number of cars that have a powerPS of greater than 500 and that is about 115 and if you take the lower range and if you take the lower value the number of cars that are less than 10 it is about 5565. These are from trial and error and that is considerable reading that has been done to even check you know what is the minimum power that you need to start any vehicle.

So, the working range year of registration is between 19 to 50, the working range for year of registration is between 19 50 to 2018 and for price we are stick into 100 to 1, 50,000 dollars and the working range for powerPS variable is from 10 to 500. So, now, that we have checked the working range for three variables we are now going to clean the data by giving these variables and any further modifications we are going to do in the clean data.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 27**  
**Case study on regression Part II**

(Refer Slide Time: 00:15)



The screenshot shows the Spyder IDE interface with the following details:

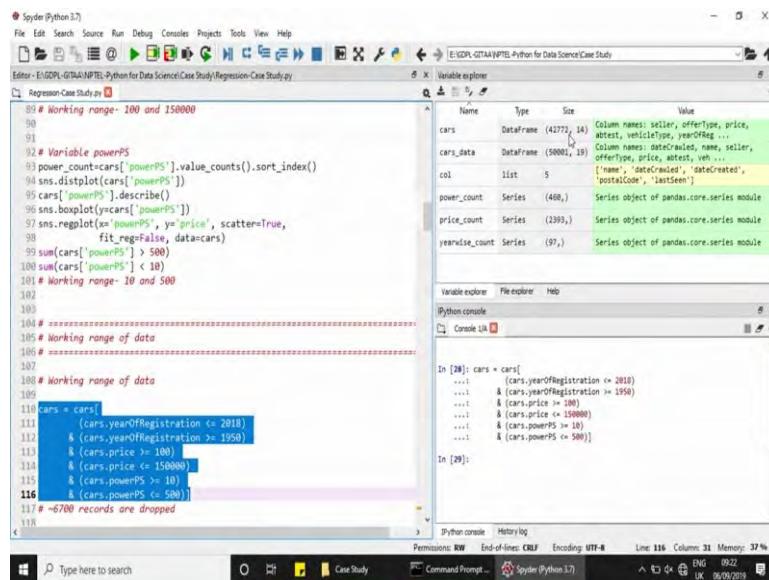
- Editor:** E:\V201-GITA\IITTEL\Python for Data Science\Case Study\Regression-Case Study.py
- Code:**

```

71 sum[cars['yearOfRegistration'] > 2018]
72 sum[cars['yearOfRegistration'] < 1950]
73 sns.regplot(x='yearOfRegistration', y='price', scatter=True,
74 fit_regmse=False, data=cars)
75 # Working range- 1950 and 2018
76
77 # Variable price
78 price_count=cars['price'].value_counts().sort_index()
79 sns.distplot(cars['price'])
80 cars['price'].describe()
81 sns.boxplot(y=cars['price'])
82 sum[cars['price'] > 150000]
83 sum[cars['price'] < 100]
84 # Working range- 100 and 150000
85
86 # Variable powerPS
87 power_count=cars['powerPS'].value_counts().sort_index()
88 sns.distplot(cars['powerPS'])
89 cars['powerPS'].describe()
90 sns.boxplot(y=cars['powerPS'])
91 sns.regplot(x='powerPS', y='price', scatter=True,
92 fit_regmse=False, data=cars)
93 sum[cars['powerPS'] > 500]
94 sum[cars['powerPS'] < 10]
95 # Working range- 10 and 500
96
97 # *****
98 # Working range of data
99
```
- Variable explorer:** Shows variables like cars, cars\_data, col, power\_count, price\_count, yearside\_count.
- Console:** In [28]:
- Bottom status bar:** Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 75, Column: 70, Memory: 37 %

So, now let us incorporate these cleaning ranges.

(Refer Slide Time: 00:18)



The screenshot shows the Spyder IDE interface with the following details:

- Editor:** E:\V201-GITA\IITTEL\Python for Data Science\Case Study\Regression-Case Study.py
- Code:**

```

81 # Working range- 100 and 150000
82
83 # Variable powerPS
84 power_count=cars['powerPS'].value_counts().sort_index()
85 sns.distplot(cars['powerPS'])
86 cars['powerPS'].describe()
87 sns.boxplot(y=cars['powerPS'])
88 sns.regplot(x='powerPS', y='price', scatter=True,
89 fit_regmse=False, data=cars)
90 sum[cars['powerPS'] > 500]
91 sum[cars['powerPS'] < 10]
92 # Working range- 10 and 500
93
94 # *****
95 # Working range of data
96
97 cars = cars[
98 (cars.yearOfRegistration <= 2018) &
99 (cars.yearOfRegistration >= 1950) &
100 (cars.price <= 150000) &
101 (cars.price >= 10) &
102 (cars.powerPS >= 10) &
103 (cars.powerPS <= 500)]
104 # ~6700 records are dropped
105
106
```
- Variable explorer:** Shows variables like cars, cars\_data, col, power\_count, price\_count, yearside\_count.
- Console:** In [29]: cars[...]
- Bottom status bar:** Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 116, Column: 31, Memory: 37 %

So, what I am trying to do here is from the cars data I am accessing yearOfRegistration and putting a bound on it. The upper bound for yearOfRegistration is 2018 and the lower

bound is 1950, both ranges inclusive. So, from cars I have accessed yearOfRegistration. Now, earlier I have used square braces this is the other way of accessing it this is also fine or if you are comfortable with the other way of accessing then that is also fine.

So, next is price, the lower bound for price is 100 and the upper bound for price is 1,50,000 and similarly for powerPS the lower bound is 10 and the upper bound is 500. So, if you see here, we started with around 49531 after removing duplicates; and now let us see how many records are we losing after this. So, we are roughly losing about 6700 records here. So, all these changes have been affected into cars. Also with yearOfRegistration it is hard to find out what is the age of the car.

(Refer Slide Time: 01:20)

The screenshot shows the Spyder Python IDE interface. On the left, there is a code editor window titled 'Regression-Case Study' containing Python code. The code includes filtering for 'powerPS' values between 100 and 150000, creating a histogram for 'powerPS' vs 'price', and filtering for 'powerPS' values between 10 and 500. It also shows a large number of rows being dropped (6700). On the right, there is a 'cars - Dataframe' viewer window showing a subset of the 'cars' DataFrame. The columns visible are 'vehicleType', 'yearOfRegistration', 'gearbox', and 'powerPS'. The data shows various car models from 1951 to 1956, with different gearbox types (manual, automatic) and powerPS values. A tooltip over the 'yearOfRegistration' column indicates it is a float type ranging from 1950 to 2018.

```

Working range- 100 and 150000
Variable powerPS
sum(cars['powerPS'].value_counts())
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(x='powerPS', y='price', fit_regmse=False, data=cars)
sum(cars['powerPS'] > 500)
sum(cars['powerPS'] < 10)
Working range- 10 and 500
=====
Working range of data
=====
Working range of data
=====
cars = cars[
 (cars.yearOfRegistration <= 2018) &
 (cars.yearOfRegistration >= 1950) &
 (cars.price >= 100) &
 (cars.price <= 150000) &
 (cars.powerPS >= 10) &
 (cars.powerPS <= 500)]
~6700 records are dropped

```

So, what we are going to do is we are going to subtract the yearOfRegistration from 2018 and add the monthOfRegistration.

(Refer Slide Time: 01:34)

The screenshot shows the Spyder Python 3.7 interface with two panes. The left pane contains Python code for data analysis, and the right pane shows the variable explorer with data frames and series.

**Code Editor:**

```
85 # Working range- 100 and 150000
86
87
88 # Variable powerPS
89 power_count=cars['powerPS'].value_counts()
90 sns.distplot(cars['powerPS'])
91 cars[['powerPS']].describe()
92 sns.boxplot(x=cars['powerPS'])
93 regplot(x='powerPS', y='price', s=9338)
94 sum(cars['powerPS'] > 500)
95 sum(cars['powerPS'] < 10)
100 # Working range- 10 and 500
101
102
103 # ****
104 # Working range of date
105 # ****
106
107
108 # Working range of data
109
110 cars = cars[
111 (cars.yearOfRegistration <= 2
112 & (cars.yearOfRegistration >= 1
113 & (cars.price >= 100)
114 & (cars.price <= 150000)
115 & (cars.powerPS >= 10)
116 & (cars.powerPS <= 500))
117 # ~6700 records are dropped
118]
```

**Variable Explorer:**

| Index | powerPS | model  | kilometer | monthOfRegistration | fuelType |
|-------|---------|--------|-----------|---------------------|----------|
| 30651 | others  | 60000  | 3         | nan                 | petrol   |
| 31173 | others  | 50000  | 6         | petrol              | petrol   |
| 34388 | nan     | 90000  | 9         | petrol              | petrol   |
| 45281 | others  | 150000 | 10        | petrol              | petrol   |
| 45747 | others  | 125000 | 6         | petrol              | petrol   |
| 42368 | others  | 60000  | 0         | petrol              | petrol   |
| 33752 | nan     | 100000 | 5         | petrol              | petrol   |
| 4255  | others  | 60000  | 6         | petrol              | petrol   |
| 45747 | others  | 150000 | 6         | petrol              | petrol   |
| 23487 | others  | 5000   | 7         | petrol              | petrol   |
| 12951 | others  | 100000 | 5         | petrol              | petrol   |
| 18533 | nan     | 20000  | 6         | petrol              | petrol   |
| 23562 | others  | 150000 | 3         | petrol              | petrol   |
| cars  | others  | 175000 | 8         | petrol              | petrol   |

Legend:  
Column names: seller, offerType, price, Attest, vehicleType, yearOfReg ...  
Column names: dateCreated, name, seller, offerType, price, attest, veh ...  
[name, offerType, dateCreated, 'petrol', '2018-01-01', 'UK', '06/09/2019']  
Series object of pandas.core.series module  
Series object of pandas.core.series module  
Series object of pandas.core.series module

Instructions:  
instruction <= 2018)  
instruction >= 1950)  
(>= 10)  
>= 500))

**Python console:**

```
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 111 Column: 43 Memory: 37%
```

**Command Prompt:**

```
ENG 0923 UK 06/09/2019
```

So, the monthOfRegistration we are going to divide it by 12 which gives you months in decimals, and I am going to create another variable called age that will contain the difference between the year 2018 and the yearOfRegistration. Now, that is the first step. So, I am going to subtract the yearOfRegistration from the year 2018 and then I am going to add the monthOfRegistration which is monthOfRegistration by 12. So, let us start by doing this.

(Refer Slide Time: 02:09)

The screenshot shows the Spyder Python 3.7 IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, Help, and a Help icon. Below the menu is a toolbar with icons for file operations like Open, Save, Run, Stop, and Help. The main area contains a Jupyter notebook titled 'Regression-Case Study.ipynb'. The code cell contains the following Python script:

```
Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

Reading the dataset
cars = pd.read_csv('cars.csv')

Displaying the first few rows of the dataset
cars.head()

Summary statistics of the dataset
cars.describe()

Checking for missing values
cars.isnull().sum()

Cleaning the data
Dropping rows with missing values
cars.dropna(inplace=True)

Selecting relevant columns
cars = cars[['name', 'yearOfRegistration', 'monthOfRegistration', 'powerPS', 'price', 'seller', 'offerType', 'attest', 'vehicleType', 'year', 'name', 'seller', 'offerType', 'price', 'attest', 'veh...', 'dateCreated', 'postcode1', 'lastseen']]

Creating new variables
Working range- 10 and 500
cars['powerPS'] = np.where(cars['powerPS'] > 500, 500, cars['powerPS'])
cars['powerPS'] = np.where(cars['powerPS'] < 10, 10, cars['powerPS'])

Working range- 10 and 500
cars['price'] = np.where(cars['price'] < 10000, 10000, cars['price'])
cars['price'] = np.where(cars['price'] > 50000, 50000, cars['price'])

Working range of data
Further to simplify- variable reduction
Combining yearOfRegistration and monthOfRegistration
cars['monthOfRegistration'] = pd.DatetimeIndex(cars['yearOfRegistration']).month

Creating new variable Age by adding yearOfRegistration and monthOfRegistration
cars['Age'] = 2018 - cars['yearOfRegistration'] + cars['monthOfRegistration']

Rescaling the data
cars['Age'] = round(cars['Age'], 2)
cars['PowerPS'] = round(cars['powerPS'], 2)
cars['Price'] = round(cars['price'], 2)

Describing the data
cars.describe()
```

The Variable explorer pane on the right shows the following data structures:

| Name           | Type      | Size        | Value                                                                                                                                                                      |
|----------------|-----------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cars           | DataFrame | (42772, 19) | Column names: seller, offerType, price, ..., attest, vehicleType, yearOfReg, ..., name, ..., offerType, price, attest, veh..., ..., dateCreated, ..., postcode1, lastseen' |
| cars_data      | DataFrame | (50000, 19) | Column names: datarawdata, name, seller, ..., offerType, price, attest, veh..., ..., dateCreated, ..., postcode1, lastseen'                                                |
| col            | list      | 5           | ['name', 'seller', 'offerType', 'dateCreated', 'postcode1', 'lastseen']                                                                                                    |
| power_count    | Series    | (400,)      | Series object of pandas.core.series module                                                                                                                                 |
| price_count    | Series    | (1000,)     | Series object of pandas.core.series module                                                                                                                                 |
| yearwise_count | Series    | (97,)       | Series object of pandas.core.series module                                                                                                                                 |

The IPython console pane shows the execution history:

```
In [28]: cars = cars[...]
 ...: (cars.yearOfRegistration <= 2018)
 ...: & (cars.yearOfRegistration >= 1950)
 ...: & (cars.price < 100)
 ...: & (cars.powerPS >= 10)
 ...: & (cars.powerPS <= 500)
 ...:] # ~6700 records are dropped
In [29]: cars = cars[...]
 ...: (cars.yearOfRegistration <= 2018)
 ...: & (cars.yearOfRegistration >= 1950)
 ...: & (cars.price < 10000)
 ...: & (cars.powerPS >= 10)
 ...: & (cars.powerPS <= 500)
 ...:]
```

The Command Prompt pane shows the current environment:

```
Permissions: RW End-of-lines CRLF Encoding: UTF-8 Line: 123 Column: 1 Memory: 37%
```

The bottom status bar indicates the current session is 'Spyder (Python 3.7)' with memory usage of 924 MB and a timestamp of 06/09/2019.

So, first we are going to divide the monthOfRegistration by 12. So, let us just do that and see if the changes are reflected.

(Refer Slide Time: 02:17)

The screenshot shows a Jupyter Notebook interface with several tabs open. The main notebook cell contains Python code for data processing and model creation. A 'Variable explorer' sidebar shows the data frame structure. The 'File' menu has 'Save' highlighted.

```
ipython 3.7.1
```

```
File Edit Search Source Run Debug Console Projects Tools View Help
```

```
E:\GDP\DATA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.ipynb
```

```
Editor - E:\GDP\DATA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.ipynb
```

```
Regression-Case Study.ipynb
```

```
Importing required libraries
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn import linear_model
```

```
from sklearn import metrics
```

```
from sklearn.model_selection import train_test_split
```

```
Reading the data
```

```
cars = pd.read_csv('cars.csv')
```

```
Displaying the first few rows of the data
```

```
cars.head()
```

```
Cleaning the data
```

```
cars.dropna(inplace=True)
```

```
Fitting regression model
```

```
fit_regr=False, data=cars)
```

```
99 sum(cars['powerPS'] > 500)
```

```
100 sum(cars['powerPS'] < 10)
```

```
101 # Working range- 10 and 500
```

```
102
```

```
103
```

```
104 # *****
```

```
105 # Working range of data
```

```
106 # *****
```

```
107
```

```
108 # Working range of data
```

```
109
```

```
110 cars = cars[
```

```
111 (cars.yearOfRegistration <= 2
```

```
112 & (cars.yearOfRegistration) >= 1
```

```
113 & (cars.price > 100)
```

```
114 & (cars.price <= 150000)
```

```
115 & (cars.powerPS >= 10)
```

```
116 & (cars.powerPS <= 500)]
```

```
117 # ~6700 records are dropped
```

```
118
```

```
119 # Further to simplify- variable reduction
```

```
120 # Combining yearOfRegistration and model
```

```
121
```

```
122 cars['monthOfRegistration'] = 12
```

```
123
```

```
124 # Creating new variable Age by adding yearOfRegistration and monthOfRegistration
```

```
125 cars['Age'] = 2018 - cars['yearOfRegistration'] + cars['monthOfRegistration']
```

```
126 cars['Age'] = round(cars['Age'], 2)
```

```
127 cars['Age'].describe()
```

So, whichever entries were 0 earlier in terms of integer, they become 0 with decimal. So, again when you add your yearOfRegistration the difference between 2018 and the yearOfRegistration, when you add it with the monthOfRegistration these entries will again be reflected as 0 under age. So, indirectly we are actually reducing the effect of 0 because if you try and use the monthOfRegistration as integer itself. Because if you try and use the monthOfRegistration as integer itself it is going to create some problems because 0 is not a valid integer to represent any of the month and the months are being represented by numbers 1 to 12, right.

So, because we wanted to get rid of 0s, we are trying to divide each of the entries by 12 which means any entry which has a monthOfRegistration of 0 will automatically remain a 0 and that when you add it with the age which is a difference between the year 2018 and the yearOfRegistration and that will eventually go off, right.

So, we are not letting go of observations which are marked as 0 under monthOfRegistration, at the same time you are also making sure that it does not affect our analysis. So, we have converted the monthOfRegistration as decimals. So, I have just divided by 12. Now, I am going to create a new variable called age in line 125 and it is

the difference between the year 2018 minus the yearOfRegistration for that particular record and I am adding the monthOfRegistration here.

(Refer Slide Time: 04:17)

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** Displays a script named "Regression-Case Study.py" containing Python code for data processing. The code includes filtering for yearOfRegistration (2018-10 and 500), creating a new variable "Age" by subtracting yearOfRegistration from 2018, rounding "Age" to 2 decimal places, and dropping columns "yearOfRegistration" and "monthOfRegistration".
- Data View:** A "cars - Dataframe" viewer window shows a sample of the car dataset. The columns are Index, Registration, fuelType, brand, otheRepairedDamage, Age, name, dateCrawled, name\_s, s, and dateCreated. The data includes rows for various car models like BMW, Volvo, Volkswagen, SEAT, etc., with their respective details.
- Console:** The "In [30]" and "In [31]" sections show the execution of the code. "In [30]" shows the creation of the "Age" column with values ranging from 13 to 21. "In [31]" shows the result of the "cars['Age'] = round(cars['Age'], 2)" command.
- Status Bar:** Shows the current memory usage (37%), encoding (UTF-8), and other system information.

You will see that an extra column gets added to cars and here you will see the different values of age, because we have added the months here you will see values which are in decimals. So, we are going to round off the decimal places to 2, using the round function. So, the input to the round function is the column that you want to round off and to how many places do you want to round it off to.

(Refer Slide Time: 04:40)

This screenshot is nearly identical to the previous one, showing the same code execution and data visualization. The key differences are:

- Code Editor:** The code remains the same, focusing on creating the "Age" column and rounding it to 2 decimal places.
- Data View:** The "cars - Dataframe" viewer shows the same data sample with the "Age" column now rounded to two decimal places (e.g., 13.00, 14.00, etc.).
- Console:** The "In [31]" section shows the result of the "cars['Age'] = round(cars['Age'], 2)" command, and the "In [32]" section shows the result of the "cars.drop(['yearOfRegistration', 'monthOfRegistration'], axis=1)" command, which has removed the specified columns from the DataFrame.
- Status Bar:** The status bar at the bottom right shows the current memory usage (37%), encoding (UTF-8), and other system information.

I am effecting these changes in the same column itself. So, again let us just open and see, yes. So, all these values have been rounded off to 2 decimal places. So, let us just quickly do a.describe.

(Refer Slide Time: 04:57)

```

Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - E:\GDPL-GITAAN\PTTEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study.py
101 # Working range- 10 and 500
102
103
104 # =====
105 # Working range of data
106 # =====
107
108 # Working range of data
109
110 cars = cars[
111 (cars.yearOfRegistration <= 2018)
112 & (cars.yearOfRegistration >= 1950)
113 & (cars.price >= 100)
114 & (cars.price <= 150000)
115 & (cars.powerPS >= 10)
116 & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118
119 # Further to simplify- variable reduction
120 # Combining yearOfRegistration and monthOfRegistration
121
122 cars['monthOfRegistration']/=12
123
124 # Creating new variable Age by adding yearOfRegistration and monthOfRegistration
125 cars['Age']=(2018-cars['yearOfRegistration']+cars['monthOfRegistration'])
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()
128
129
130 # Dropping yearOfRegistration and monthOfRegistration
131 cars=cars.drop(columns=['yearOfRegistration','monthOfRegistration'], axis=1)

```

In [31]: cars['Age'].round(cars['Age'],2)

In [32]: cars['Age'].describe()

Out[32]:

|      | count | mean   | std   | min   | 25%    | 50%    | 75%    | max    |
|------|-------|--------|-------|-------|--------|--------|--------|--------|
| Name | 42772 | 14.873 | 7.093 | 0.000 | 10.330 | 14.830 | 19.170 | 67.750 |
| Age  | 42772 | 14.873 | 7.093 | 0.000 | 10.330 | 14.830 | 19.170 | 67.750 |

In [33]:

So, you will see that the count is again 42772 and the mean and the mean age or the average of age is 14.87 with a standard deviation of 7 and minimum is 0, maximum is 67, and the median lies at 14.83. So, the mean and the median are not very far off, the mean is at 14.87 and the median is at 14.83. So, we know the data is not very skewed.

(Refer Slide Time: 05:27)

```

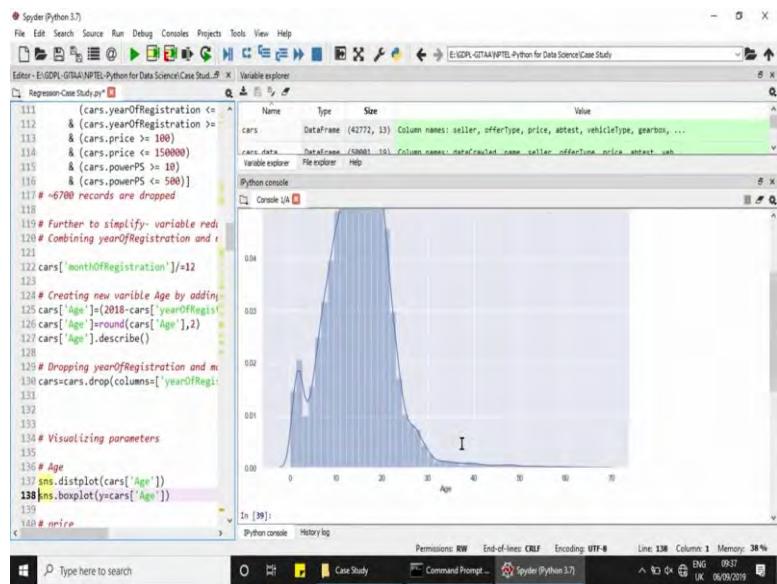
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - E:\GDPL-GITAAN\PTTEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study.py
102
103
104 # =====
105 # Working range of data
106 # =====
107
108 # Working range of data
109
110 cars = cars[
111 (cars.yearOfRegistration <= 2018)
112 & (cars.yearOfRegistration >= 1950)
113 & (cars.price >= 100)
114 & (cars.price <= 150000)
115 & (cars.powerPS >= 10)
116 & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118
119 # Further to simplify- variable reduction
120 # Combining yearOfRegistration and monthOfRegistration
121
122 cars['monthOfRegistration']/=12
123
124 # Creating new variable Age by adding yearOfRegistration and monthOfRegistration
125 cars['Age']=(2018-cars['yearOfRegistration']+cars['monthOfRegistration'])
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()
128
129
130 # Dropping yearOfRegistration and monthOfRegistration
131 cars=cars.drop(columns=['yearOfRegistration','monthOfRegistration'], axis=1)
132

```

In [34]:

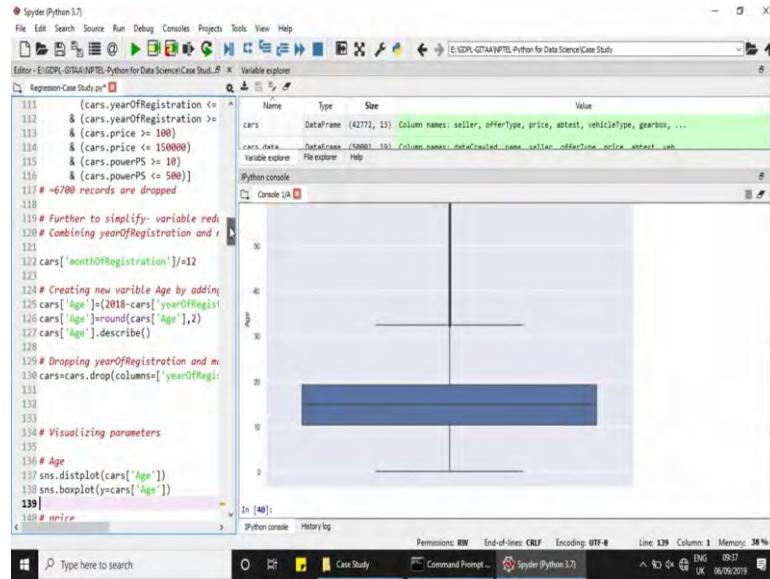
So, now that we have incorporated the information of yearOfRegistration and monthOfRegistration into 8. We can drop off these columns because adding them is only going to make it very redundant. So, I am going to use the cars.drop function to clear. So, I am going to use the cars.drop function to drop them and I am giving the column names as a list, and I am giving access is equal to 1 because these are because these are columns. Now, once you drop them you can see that the number of columns have reduced by 2 and the year and monthOfRegistration have disappeared, right.

(Refer Slide Time: 06:01)



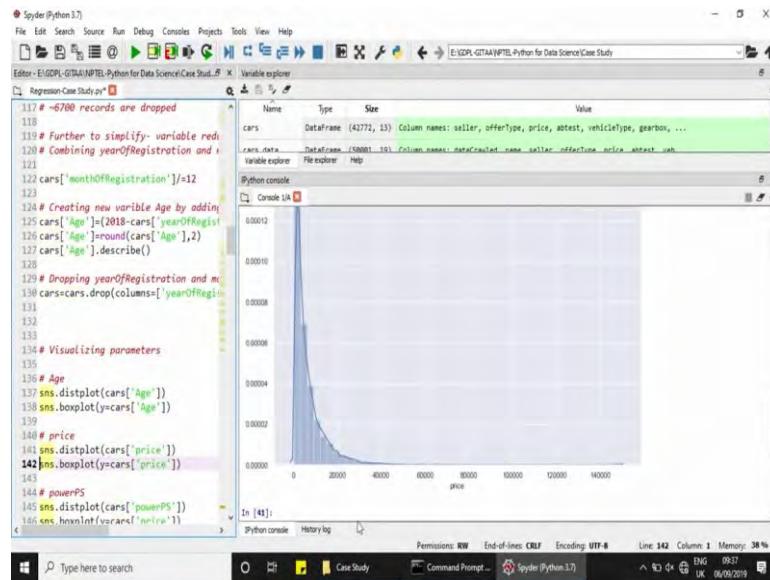
So, now let us start by visualizing parameters I am going to first start with visualizing age. So, let us see how does the histogram look for age. So, if you look at age here you can see a slightly skewed distribution, it is not extremely skewed, let me just drag let me just drag the box to show you how does the distribution look. Again you can fit a density curve over it. So, now, again let us plot a box plot here for age.

(Refer Slide Time: 06:37)



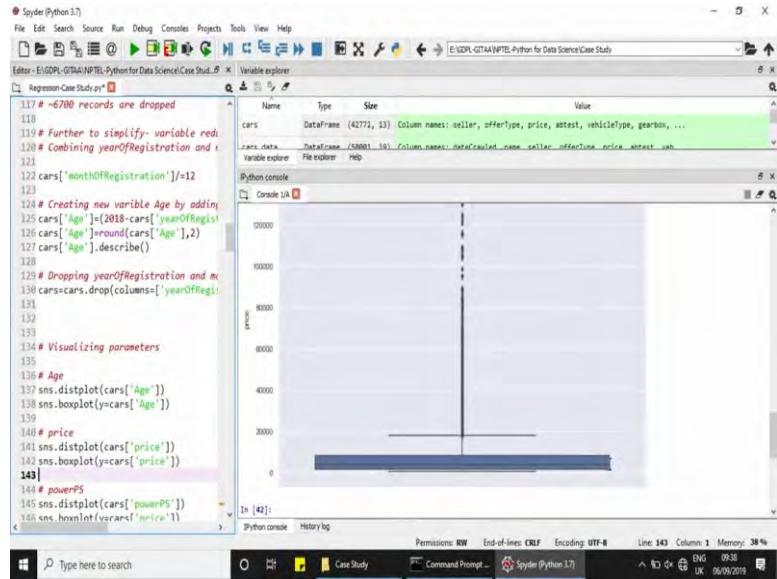
Now, we can see a clear cut box here, though above the whiskers there are a few extreme values or outliers, right. But even this was not possible in the earlier case.

(Refer Slide Time: 06:51)



So, let us now look at the histogram for price. Now, for price here you will see histogram, even this was not seen earlier, but now we are able to see a few more ranges.

(Refer Slide Time: 07:00)



The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains a script named 'Regression-Case Study.py' with the following content:

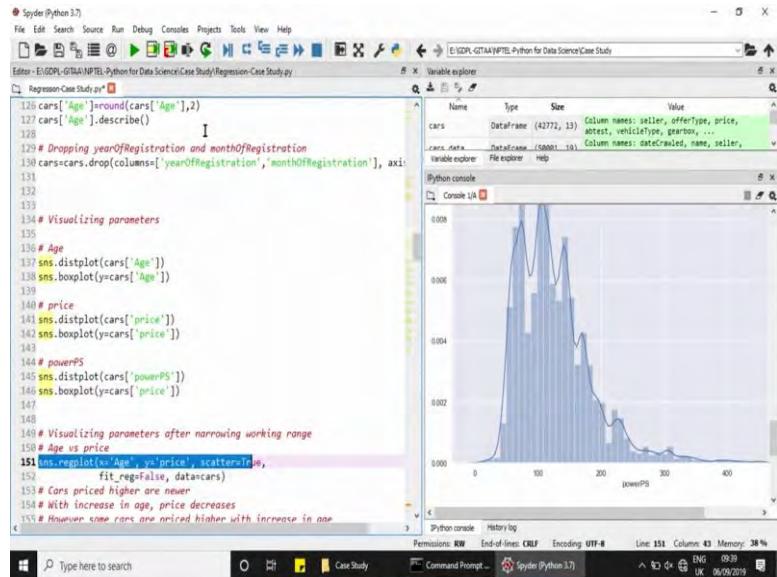
```
111 # ~6700 records are dropped
112
113 # Further to simplify- variable reduction
114 # Combining yearOfRegistration and monthOfRegistration
115
116 cars['monthOfRegistration']=12
117
118 # Creating new variable Age by adding yearOfRegistration
119 cars['Age']=2018-cars['yearOfRegistration']
120 cars['Age']=round(cars['Age'],2)
121 cars['Age'].describe()
122
123 # Dropping yearOfRegistration and monthOfRegistration
124 cars=cars.drop(columns=['yearOfRegistration'])
125
126 # Visualizing parameters
127
128 # Age
129 sns.distplot(cars['Age'])
130 sns.boxplot(y=cars['Age'])
131
132 # price
133 sns.distplot(cars['price'])
134 sns.boxplot(y=cars['price'])
135
136 # powerPS
137 sns.distplot(cars['powerPS'])
138 sns.histplot(var=cars['powerPS'])
```

The IPython console shows a box plot for the 'Age' variable, which has a median around 20 and some outliers extending beyond the whiskers.

And if you plot the box plot for the variable price you will see a small box here, earlier what we could really see was just a line, right. Now, this is better, but again you still have a few extreme values outside the whiskers.

If you look at powerPS; if you look at powerPS you will see that the histogram is now better, better than earlier definitely because earlier it was all bundled up to one side and you can see that the density curve is also being fit here and you will see more ranges to powerPS than it was earlier.

(Refer Slide Time: 07:29)



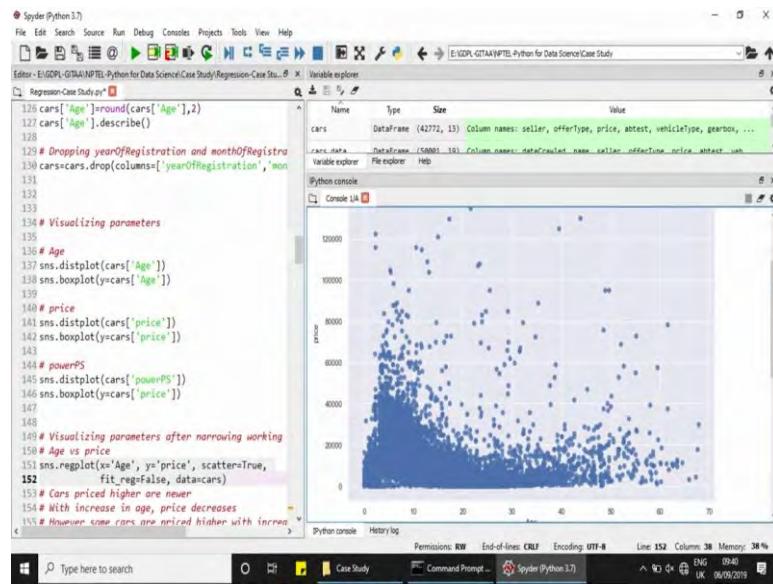
The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains a script named 'Regression-Case Study.py' with the following content:

```
128 cars['Age']=round(cars['Age'],2)
129 cars['Age'].describe()
130
131 # Dropping yearOfRegistration and monthOfRegistration
132 cars=cars.drop(columns=['yearOfRegistration','monthOfRegistration'], axis=1)
133
134 # Visualizing parameters
135
136 # Age
137 sns.distplot(cars['Age'])
138 sns.boxplot(y=cars['Age'])
139
140 # price
141 sns.distplot(cars['price'])
142 sns.boxplot(y=cars['price'])
143
144 # powerPS
145 sns.distplot(cars['powerPS'])
146 sns.boxplot(y=cars['powerPS'])
147
148
149 # Visualizing parameters after narrowing working range
150 # Age vs price
151 sns.regplot(x='Age', y='price', scatter=True,
152 fit_reg=False, data=cars)
153 # Cars priced higher are newer
154 # With increase in age, price decreases
155 # However some cars are priced higher with increase in age
```

The IPython console shows a density plot for the 'powerPS' variable, which is centered around 100 with a multi-modal distribution and a fitted density curve.

So, now, that we have seen the individual distributions and we have seen the individual histograms and box plot for the 3 variables, let us see what their effect is on price. So, let us start by plotting. So, let us start by plotting age versus price here. I am using the regplot that is under c1 package again here.

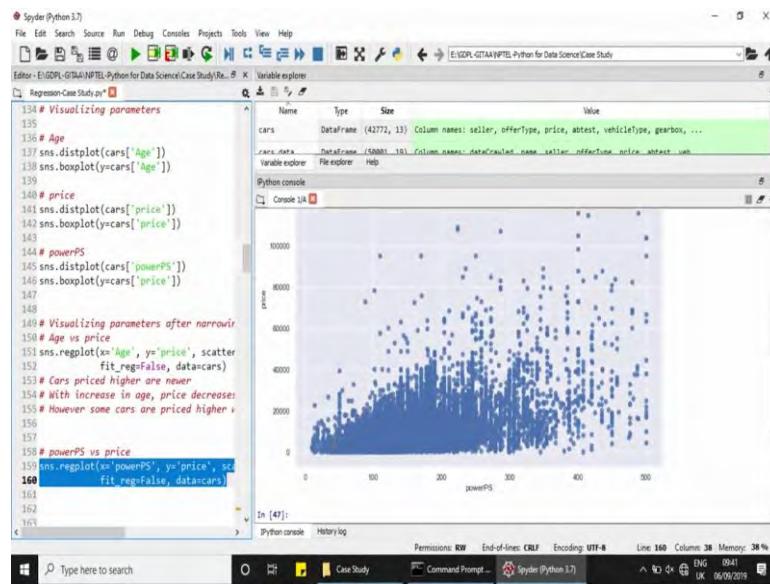
(Refer Slide Time: 08:08)



So, let me just drag this output window to the left to show you how does the plot look. So, you can see that it is a scatter plot and so you can see that it is a scatter plot; earlier when you try to plot yearOfRegistration versus price you would have seen that all these values were bunched up together in one end and we could not really see a clear cut effect of age on price, but here it is much better.

We can see that cars which are priced higher are fairly newer, so you have age on the x axis price on the y axis and cars which are priced higher are fairly new. But cars, but there are cars which are priced a little higher and are still older, so we can treat them as vintage. And on a very general note with increase in age the price also drops, right. So, this is the effect of age on price. Yes, it does affect a price to a great deal. But now, let us see the effect of powerPS versus price.

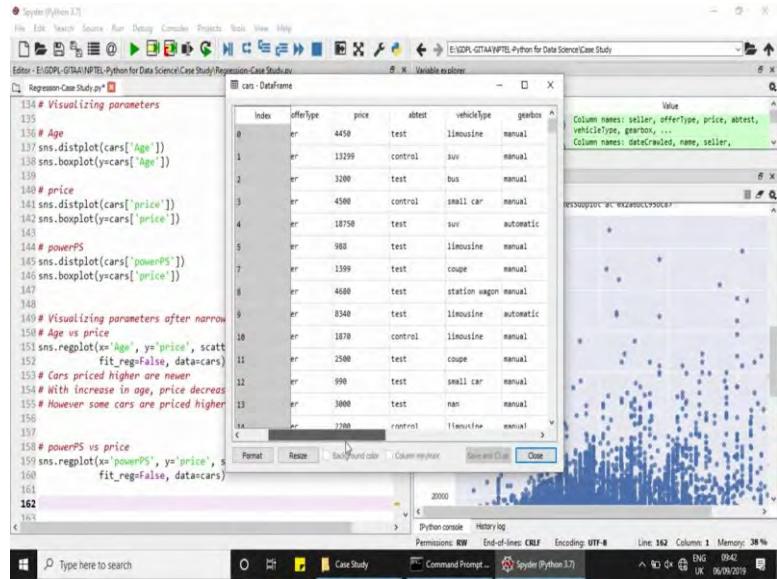
(Refer Slide Time: 09:32)



So, I am again using the same plot to plot powerPS with price. My power is on the x axis and my price is again on the y axis. So, here we can see with increase in power the price increases and that is very very clear from this plot because earlier we were not able to infer the effect of powerPS on price, we were not sure, but now that we have arrived at a very very narrow range of working values we can see an effect of powerPS on price. So, with increase in powerPS, the price eventually increases.

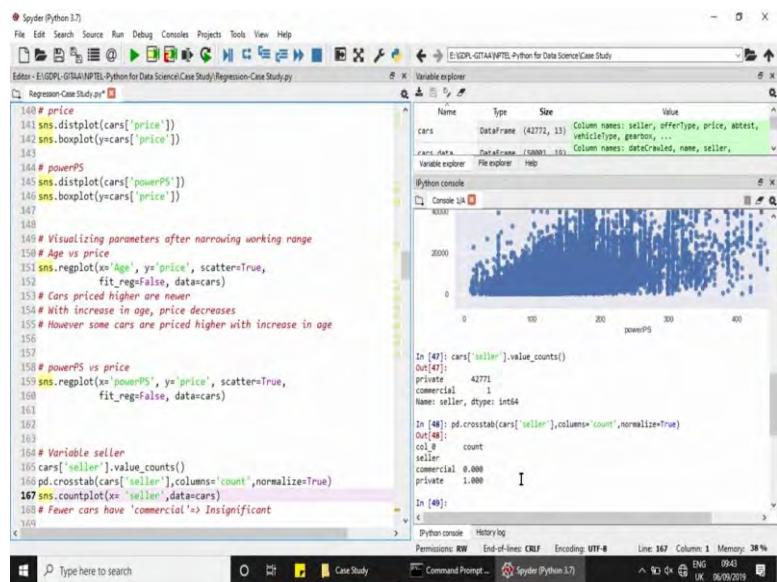
So, these were the effect of powerPS, age on price. Of course, age is a combination of yearOfRegistration and monthOfRegistration. So, two variables have been incorporated into a single in the form of a single variable. Now, having done this we are also going to check what are the effect of other variables on price, right.

(Refer Slide Time: 10:38)



But before that we have to even narrow down if these variables, if these, but mostly the categorical variables if they have enough if there are enough categories or if there is enough count under each category for us to even retain them. So, you again have seller offer type then you have abtest and vehicleType gearbox and there are many more categorical variables. So, now let us see what is the individual frequency count under each of these categories.

(Refer Slide Time: 11:12)



So, I first I am, so I first taken seller the variable seller and if you look at the value count you will see private is 42771 and commercial is only 1. So, the seller variable has two categories, private and commercial, but one of the category has only one observation. So, we are going to drop it and if you are interested from a proportion point of view then you can clearly see that private almost accounts for 100 percent of the proportion.

So, now that we have seen all this, we can clearly say that the variable seller is insignificant. So, we will also see how to remove it later on, but just let us identify all the variables which are insignificant and remove them together.

(Refer Slide Time: 12:00)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains several lines of Python code related to car data analysis, including plots for 'powerPS' vs 'price' and 'offerType' vs 'seller'. The variable explorer shows a DataFrame named 'cars' with 42772 rows and 13 columns, including 'offerType', 'seller', 'price', and 'abtest'. The IPython console displays the output of these analyses, such as the value counts for 'offerType' (offer: 42772, control: 0) and 'abtest' (test: 2138, control: 20644). A bar chart in the console shows the distribution of 'seller' types: private (~99%) and commercial (~1%).

```

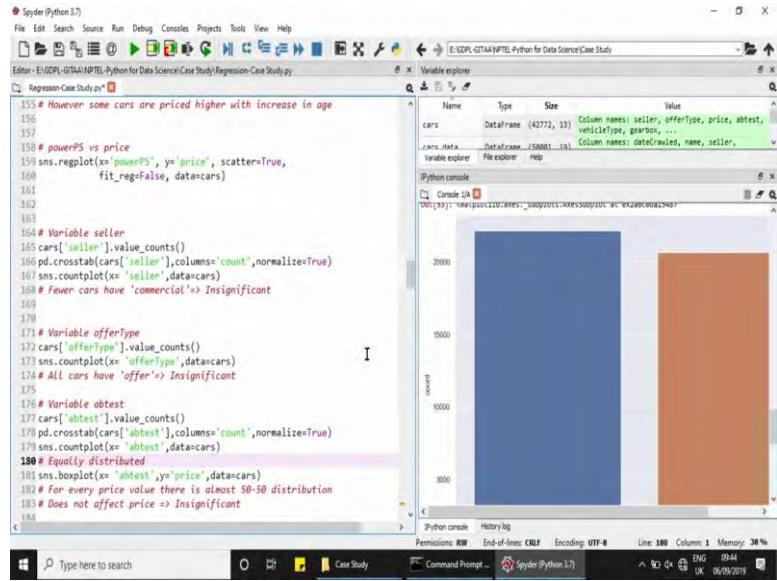
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor: E:\GDP\GITAA\INTEL Python for Data Science\Case Study\Regression-Case Study
Regression-Case Study.py
152 # fit_regrFalse, data=cars)
153 # Cars priced higher are newer
154 # With increase in age, price decreases
155 # However some cars are priced higher with increase in age
156
157
158 # powerPS vs price
159 sns.regplot(x='powerPS', y='price', scatter=True,
160 fit_regr=False, data=cars)
161
162
163
164 # Variable seller
165 cars['seller'].value_counts()
166 pd.crosstab(cars['seller'],columns='count',normalize=True)
167 sns.countplot(x='seller',data=cars)
168 # Fewer cars have 'commercial' > Insignificant
169
170
171 # Variable offerType
172 cars['offerType'].value_counts()
173 sns.countplot(x='offerType',data=cars)
174 # All cars have 'offer' > Insignificant
175
176 # Variable abtest
177 cars['abtest'].value_counts()
178 pd.crosstab(cars['abtest'],columns='count',normalize=True)
179 sns.countplot(x='abtest',data=cars)
180 # Equally distributed
181 sns.hvplot(x='abtest', y='neire', data=cars)

```

So, the next variable is offer type. So, in your earlier data you would have seen two offer type, one is offer, the other is request. And in this case all the 4 all the 42772 records are of the type offer, so we do not have any of the records under request. So, again this variable is again going to be insignificant.

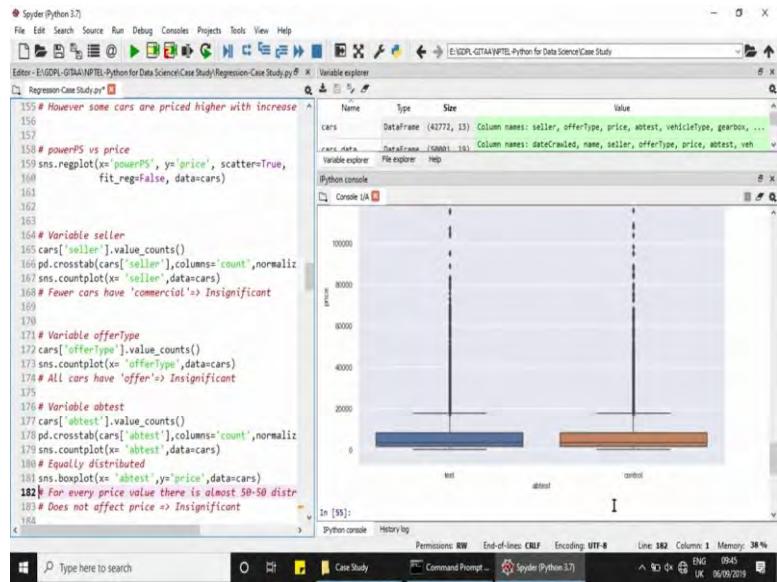
Now, moving further let us look at the variable abtest. So, under abtest if you do a value count you can see there is an equal split between the categories, test and control, right. And from a proportion point of view 51 percent, roughly about 52 percent is under test and the remaining 48 percent is under control, right. So, this is more or less, it is a closed cut its more or less equal and that can also be seen from the bar graph here, right.

(Refer Slide Time: 12:50)



It is cutting very very closely both the categories. So, both the categories are cutting very very closely. So, they are more or less equally distributed, but let us say their effect on price

(Refer Slide Time: 13:06)



But even from the box plot it seems like the effect is more or less the same. So, there is nothing much you can tell whether between categories test and control. They seem to have the same median or, they seem to have the same median for price. So, there is nothing much that we can infer even from the box plot. So, for every price value there is

almost a 50-50 distribution. So, we cannot really conclude anything from abtest. So, we are going to treat it in significant.

(Refer Slide Time: 13:37)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains a script named 'Regression-Case Study.py' with the following content:

```

161
162
163
164 # Variable seller
165 cars['seller'].value_counts()
166 pd.crosstab(cars['seller'],columns='count',normalize=True)
167 sns.countplot(x='seller',data=cars)
168 # Fewer cars have 'commercial' > Insignificant
169
170
171 # Variable offerType
172 cars['offerType'].value_counts()
173 sns.countplot(x='offerType',data=cars)
174 # All cars have 'offer' > Insignificant
175
176 # Variable abtest
177 cars['abtest'].value_counts()
178 pd.crosstab(cars['abtest'],columns='count',normalize=True)
179 sns.countplot(x='abtest',data=cars)
180 # Equally distributed
181 sns.boxplot(x='abtest',y='price',data=cars)
182 # For every price value there is almost 50-50 distribution
183 # Does not affect price > Insignificant
184
185
186 # Variable vehicleType
187 cars['vehicleType'].value_counts()
188 pd.crosstab(cars['vehicleType'],columns='count',normalize=True)
189 sns.countplot(x='vehicleType',data=cars)
190 sns.heatmap(~cars['vehicleType'].corr(),data=cars)

```

The variable explorer shows the following data frame:

| Name      | Type      | Size        | Value                                                                                                                                                     |
|-----------|-----------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| cars      | DataFrame | (42772, 13) | Column names: seller, offerType, price, abtest, vehicleType, gearbox, power, length, width, height, weight, extra, dateCreated, name, seller, vehicleType |
| cars.data | Dataframe | (42772, 13) | Column names: dateCreated, name, seller, vehicleType                                                                                                      |

The Python console displays the following output for 'vehicleType' value counts:

```

In [56]: cars['vehicleType'].value_counts()
Out[56]:
limousine 11746
small car 9285
station wagon 8976
bus 5597
cabrio 2782
coupe 2261
suv 1813
others 326
Name: vehicleType, dtype: int64

```

And the following output for 'vehicleType' correlation matrix:

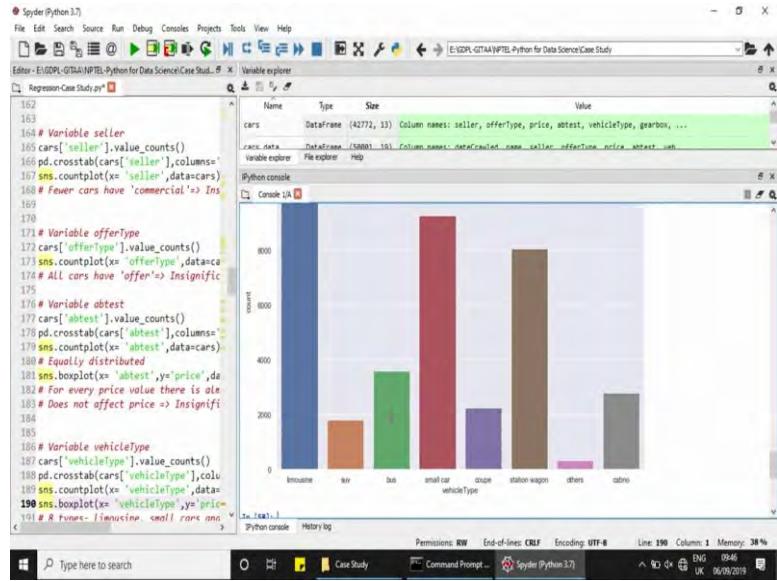
```

In [57]:
Out[57]:

```

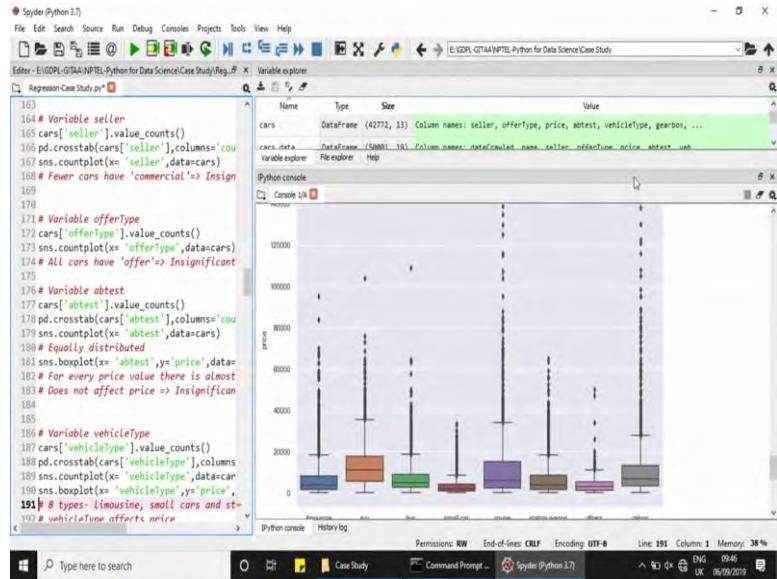
This is vehicleType and if you do a value counts for them you will see that the category topping the charts is limousine, followed by that you have small car, and then station wagon, and then you have bus, and cabrio, coupe and others, SUV, and then others, right. So, for a very proportion based account you can see that limousine is the highest with about 29 percent and let us just try to understand the proportion or the frequency better.

(Refer Slide Time: 14:04)



From the products see more easier to understand that limousine is contributing the most and followed by that you have small cars, station wagon and bus, right. So, this is the order. It is more easier to understand from the plot than from the proportion count when under each variable you have several categories, ok.

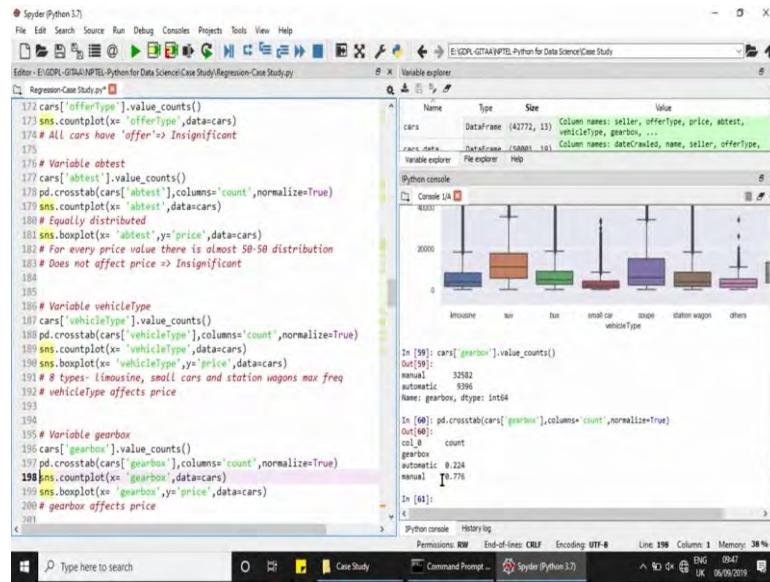
(Refer Slide Time: 14:28)



So, let us just see its effect on price. So, I am plotting a box plot of vehicleType with price. I have just dragged the output window to show you how does the screen look. So, for different categories under vehicleType, you can see that there are different price

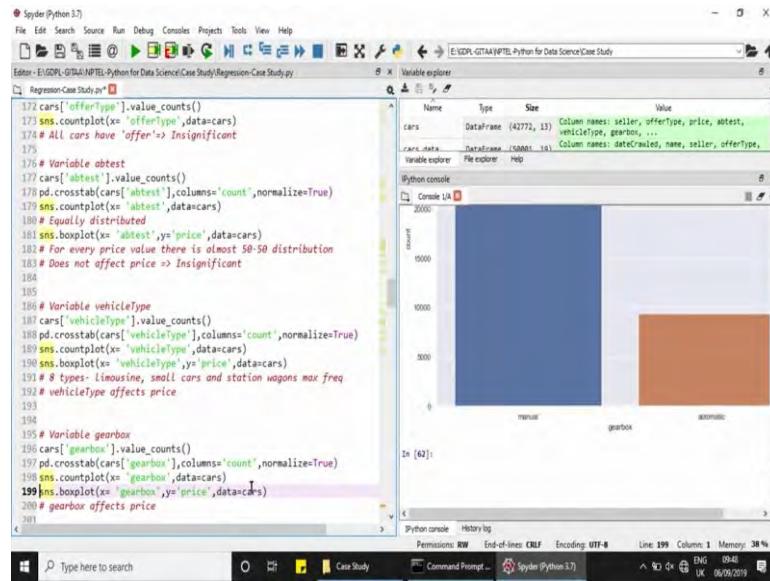
ranges, right. So, it is all it is all different, there is nothing constant about each category and limousine again is very very different from you know coupe or the bus or station wagon. So, the price are different for different categories. So, yes, vehicleType does affect price and we are going to retain it in our model.

(Refer Slide Time: 15:17)



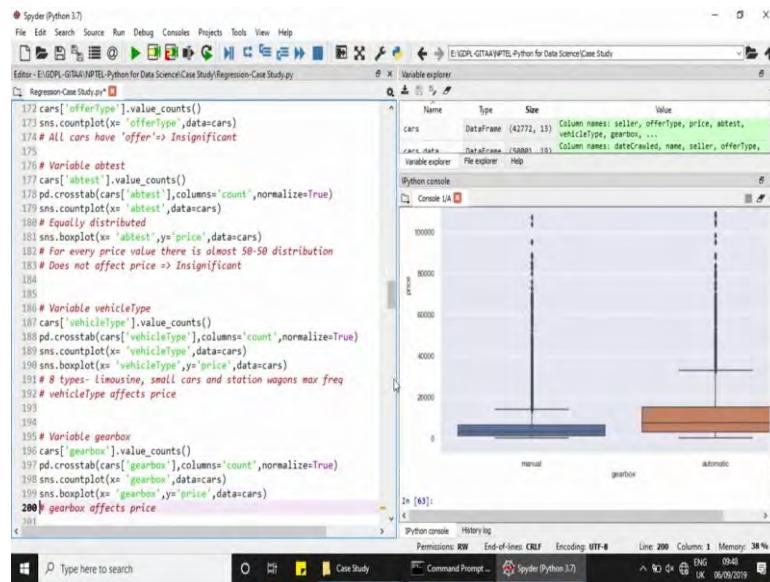
So, the next variable is gearbox. Let us look at the count for gearbox. You have 32582 records under manual and the remaining 9396 under automatic and these are the two categories that are available. And if you want a proportion based view you can see that manual counts for about 78 percent roughly and automatic accounts for roughly about 22 percent, right. And again we can; and again we can do a count plot for it which will also show you that manual gearbox are the most frequent.

(Refer Slide Time: 15:47)



And from your box plot let us see if the gearbox has an effect on price.

(Refer Slide Time: 15:53)



And yes, it does because manual is priced lower compared to automatic, right and that is evident from the box plot itself. So, yes, gearbox does affect price and we are going to retain gearbox in our final model.

(Refer Slide Time: 16:14)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the following content:

```
181 sns.boxplot(x='abtest',y='price',data=cars)
182 # For every price value there is almost 50-50 distribution
183 # Does not affect price => Insignificant
184
185
186 # Variable vehicleType
187 cars['vehicleType'].value_counts()
188 pd.crosstab(cars['vehicleType'],columns='count',normalize=True)
189 sns.countplot(x='vehicleType',data=cars)
190 sns.boxplot(x='vehicleType',y='price',data=cars)
191 # 8 types- limousine, small cars and station wagons max freq
192 # vehicleType affects price
193
194
195 # Variable gearbox
196 cars['gearbox'].value_counts()
197 pd.crosstab(cars['gearbox'],columns='count',normalize=True)
198 sns.countplot(x='gearbox',data=cars)
199 sns.boxplot(x='gearbox',y='price',data=cars)
200 # gearbox affects price
201
202
203 # Variable model
204 cars['model'].value_counts()
205 pd.crosstab(cars['model'],columns='count',normalize=True)
206 sns.countplot(x='model',data=cars)
207 sns.boxplot(x='model',y='price',data=cars)
208 # Cars are distributed over many models
209 # Considered in modelling
210
```

The variable explorer shows the DataFrame 'cars' with columns: seller, offerType, price, abtest, vehicleType, gearbox, dateCrawled, name, and seller, offerType. The IPython console displays a box plot for 'gearbox' and a frequency count for 'model'.

The next is the variable model which is the model of the car and if you look at the frequency count you can see that you have about 247 models, right and of course, we cannot know the value count for all, but the model golf seems to be the most frequently occurring. So, moving further we are going to look into the variable kilometer.

(Refer Slide Time: 16:36)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the following content:

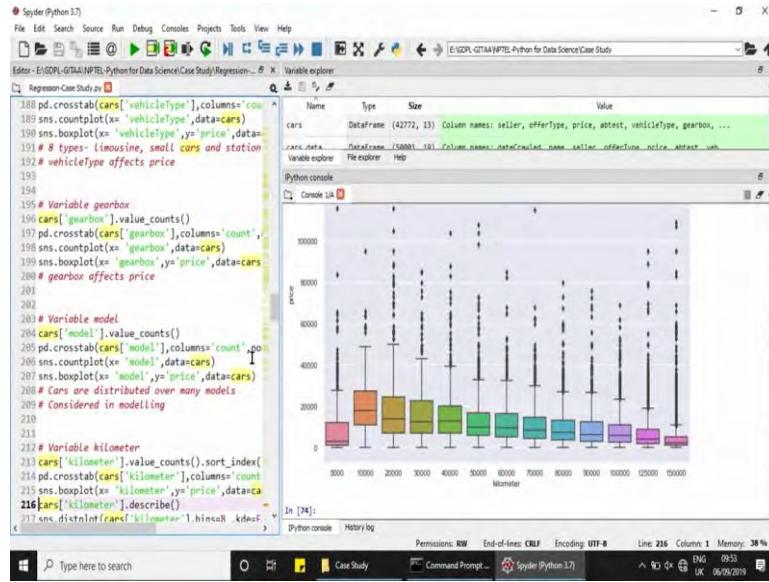
```
187 cars['vehicleType'].value_counts()
188 pd.crosstab(cars['vehicleType'],columns='count',normalize=True)
189 sns.countplot(x='vehicleType',data=cars)
190 sns.boxplot(x='vehicleType',y='price',data=cars)
191 # 8 types- limousine, small cars and station wagons max freq
192 # vehicleType affects price
193
194
195 # Variable gearbox
196 cars['gearbox'].value_counts()
197 pd.crosstab(cars['gearbox'],columns='count',normalize=True)
198 sns.countplot(x='gearbox',data=cars)
199 sns.boxplot(x='gearbox',y='price',data=cars)
200 # gearbox affects price
201
202
203 # Variable model
204 cars['model'].value_counts()
205 pd.crosstab(cars['model'],columns='count',normalize=True)
206 sns.countplot(x='model',data=cars)
207 sns.boxplot(x='model',y='price',data=cars)
208 # Cars are distributed over many models
209 # Considered in modelling
210
211
212 # Variable kilometer
213 cars['kilometer'].value_counts().sort_index()
214 pd.crosstab(cars['kilometer'],columns='count',normalize=True)
215 sns.boxplot(x='kilometer',y='price',data=cars)
216 cars['kilometer'].describe()
```

The variable explorer shows the DataFrame 'cars' with columns: seller, offerType, price, abtest, vehicleType, gearbox, dateCrawled, name, seller, offerType. The IPython console displays a frequency count for 'kilometer'.

Now, to check the value counts I have sorted out the kilometers in the increasing order based on the index using the sort index function and you can see that the kilometer which is equal to 150000 contributes the maximum and in terms of proportion if

you see it contributes to about roughly 64 percent. So, it is the most frequently occurring kilometer. So, most of the cars have a kilometer of 150000. And let us see if its effect on price and I am doing a boxplot for it.

(Refer Slide Time: 17:10)



So, you can see here that different kilometers affect the price in a very very different sense and kilometers, so if cars that have traveled lower kilometers like 10000 and 20000 are going for are being sold or go to rent, lease for a much higher price. But whereas, cars which have gone whereas, cars which have had traveled for a lot of kilometers eventually their price drops, except for this one variable, except for this one case where the kilometer is at 5000 and the median of the price is very very less.

So, this is the only outlier in terms of the behavior, but otherwise with increase in kilometer you will see that the price also decreases. So, eventually when the car has traveled more, you will see that there is more wear and tear, and hence the price drops, ok. So, this is a take home message with this variable.

(Refer Slide Time: 18:09)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains a script named 'Regression-Case Study.py' with the following content:

```
200 # gearbox affects price
201
202
203 # Variable model
204 cars['model'].value_counts()
205 pd.crosstab(cars['model'],columns='count',normalize=True)
206 sns.countplot(x='model',data=cars)
207 sns.boxplot(x='model',y='price',data=cars)
208 # Cars are distributed over many models
209 # Considered in modelling
210
211
212 # Variable kilometer
213 cars['kilometer'].value_counts().sort_index()
214 pd.crosstab(cars['kilometer'],columns='count',normalize=True)
215 sns.boxplot(x='kilometer',y='price',data=cars)
216 cars['kilometer'].describe()
217 sns.distplot(cars['kilometer'],bins=8,kde=False)
218 sns.regplot(x='kilometer',y='price',scatter=True,
219 fit_regmse=False,data=cars)
220 # Considered in modelling
221
222
223 # Variable fuelType
224 cars['fuelType'].value_counts()
225 pd.crosstab(cars['fuelType'],columns='count',normalize=True)
226 sns.countplot(x='fuelType',data=cars)
227 sns.boxplot(x='fuelType',y='price',data=cars)
228 # fuelType affects price
229
```

The variable explorer shows the 'cars' DataFrame with columns: seller, offerType, price, abtest, vehicleType, gearbox, ... and the 'cars\_data' DataFrame with columns: data, carId, name, seller, offerType, price, ... The console window displays the following output:

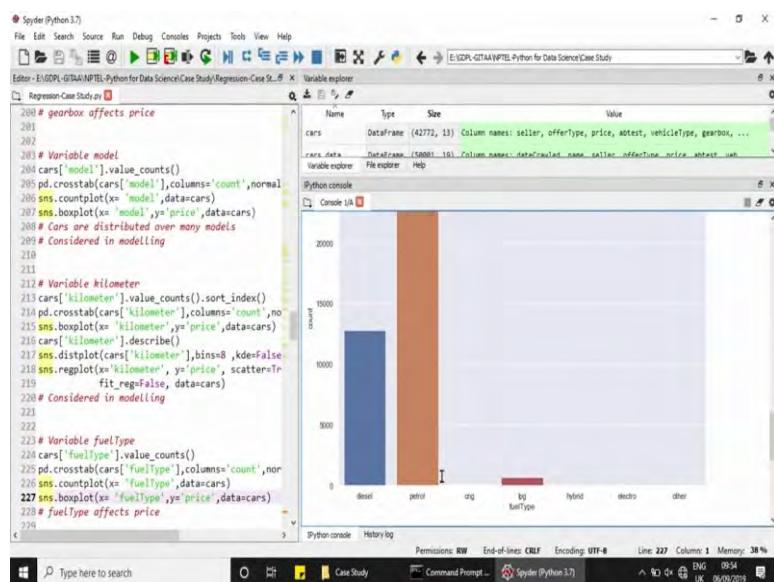
```
In [24]: cars['fuelType'].value_counts()
Out[24]:
petrol 26509
diesel 12054
lpg 1008
cng 79
hybrid 36
electro 10
other 6
Name: fuelType, dtype: int64

In [25]: pd.crosstab(cars['fuelType'],columns='count',normalize=True)
Out[25]:
col_0 count
fuelType
petrol 0.668
diesel 0.320
electro 0.000
hybrid 0.000
lpg 0.017
other 0.000
petrol 0.668
```

The bottom status bar shows the current line (Line 226), column (Column 1), and memory usage (38%).

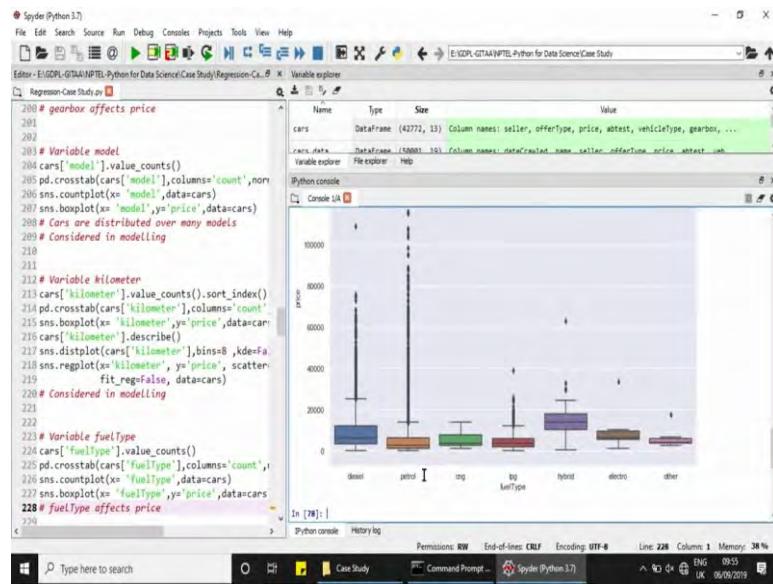
So, let us now look at the variable fuelType. So, we are going to retain kilometer and now let us look at the variable fuelType, and if you do a value counts for it you will see that petrol is topping with a frequency of over 26500, followed by that you have diesel and then you have LPG and then you have CNG, CNG hybrid electro and others, they all contribute for very very small fraction. And in terms of proportion, you will see petrol is leading with about 66 percent.

(Refer Slide Time: 18:45)



Now, let us just understand this better in terms of a bar plot. On my right you have the bar plot. So, you can see that here diesel, you can see here that petrol is contributing the maximum followed by that you have diesel and then you have LPG, but we cannot even see the bars for the others because they are very very less. And the effect of and the count of petrol and diesel is actually smearing out the effect of others which is obvious. So, now, let us just try to understand if fuelType affects price.

(Refer Slide Time: 19:18)



Now, definitely it does, so from the. So, definitely it affects price and that is clear from the box plot because different categories of fuelTypes has different price ranges and that is very very clear from the boxplot itself. So, yes, we are going to retain fuelType in our modeling. Let us move on to the variable brand.

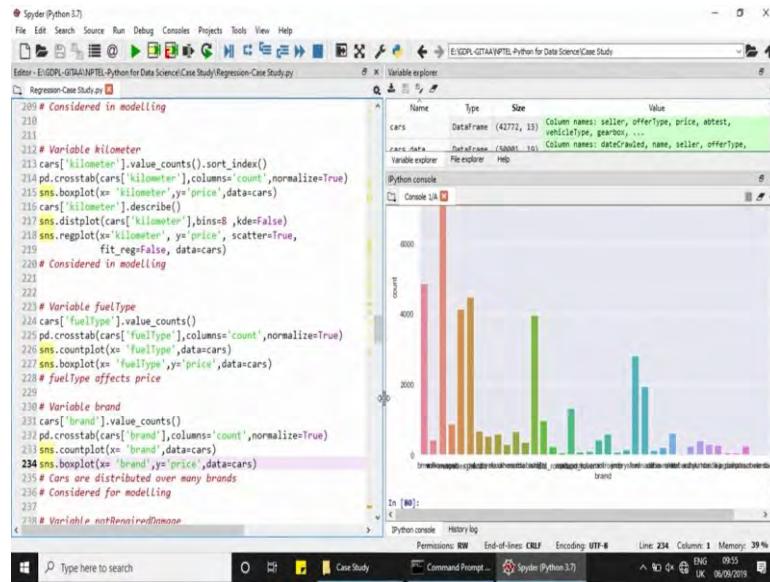
(Refer Slide Time: 19:39)

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** E:\DPL-GITA\AI\TEL\Python for Data Science\Case Study\Regression-Case Study.py
- Code:** The code block contains several lines of Python code related to car price analysis, including frequency counts for kilometers, fuel types, and brands.
- Variable explorer:** Shows a DataFrame named "cars" with columns: seller, offerType, price, abtest, vehicleType, gearbox, ... and a Dataframe named "dataCars" with columns: dateCrawled, name, seller, offerType, ...
- Console:** Displays the output of the command `cars['brand'].value\_counts()` which lists the count of occurrences for each brand: volkswagen (9134), bmw (4668), opel (4487), mercedes\_benz (4134), audi (2814), ford (2815), renault (1941), peugeot (1323), fiat (996), seat (886), skoda (698), mazda (663), smart (623), nissan (601), citroen (598), toyota (547), volvo (429), mini (428), hyundai (406).

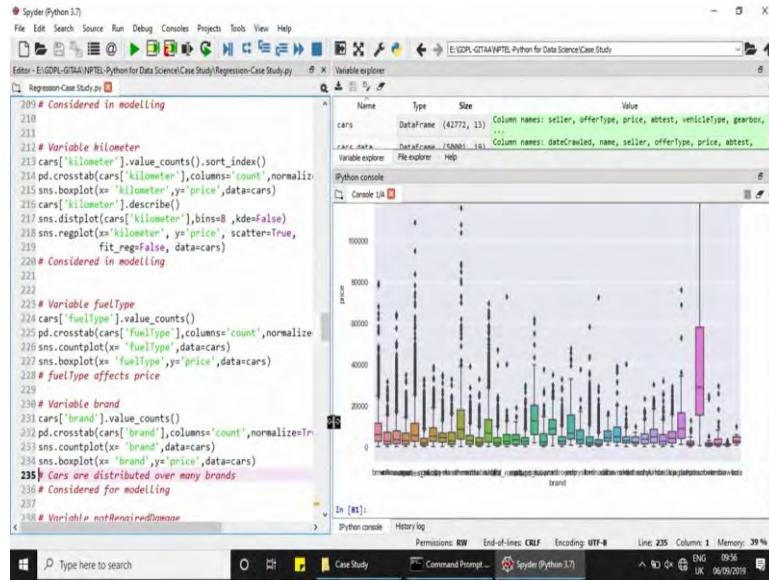
So, if you do a frequency count you will see a whole lot of brands here, but the one that stopping the chart is Volkswagen with about 9134 occurrences and followed by that you have BMW and Opel and Mercedes Benz and so on and so forth, right.

(Refer Slide Time: 20:01)



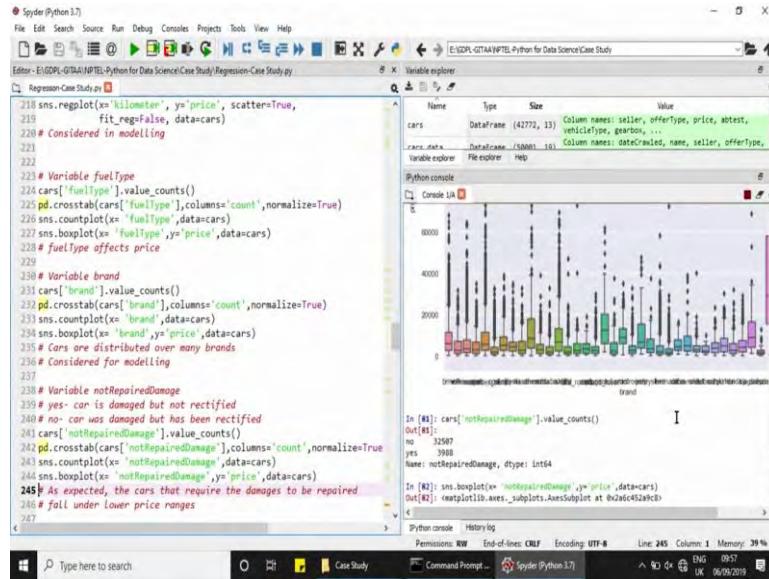
So, again if you do a count plot you will see different ranges for it. This is how the count plot looks for all the brands and to understand if brand would it definitely affect price, we are doing going for a box plot.

(Refer Slide Time: 20:14)



It is very very clear from the box plot that all the brands do have an effect, but were not able to see which of the boxes correspond to which brand that is not evident from the box plot, but on an overall note we can see that different brands have different price ranges. So, the last variable that we are going to look into is not repaired or damaged.

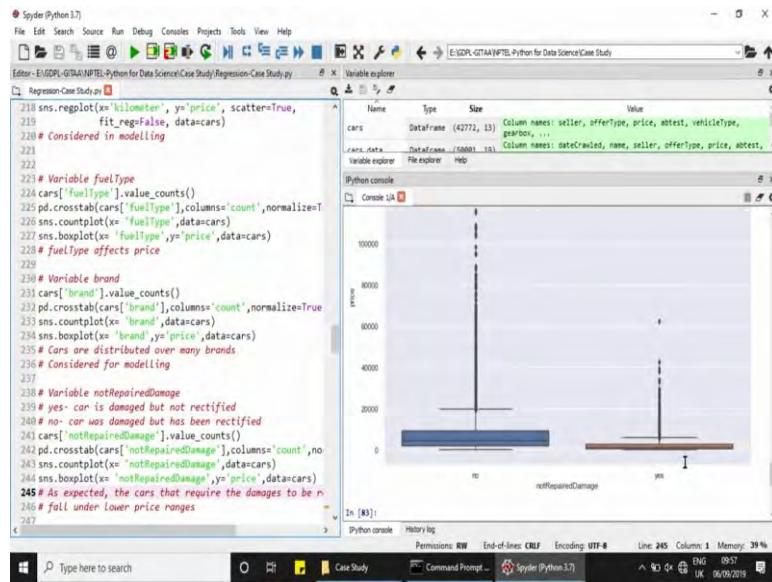
(Refer Slide Time: 20:33)



Now, since these are secondhand cars, there is already some damage that has happened to these cars. Now, this variable will tell you whether has that damage being rectified or has that damage is just being retained and not looked after, right. Now, if there are two

levels to, so there are two categories to this variable, one is yes, the other is no. When you have yes, it means that the car was damaged, but the damage has not been rectified and the second category says that no the car was damaged, but the damage has been looked after, right. So, these are the two levels. So, let us look at the value counts for this. So, the category with no is 32507 and the category yes is 30988.

(Refer Slide Time: 21:27)



But if you want to see their effect on price you will have to do a box plot which eventually tells you that cars where the damage has not been rectified are under the category yes and they have a lower price range. So, cars which where the damage has not been looked after too, are going are being sold or quoted for a lower price range compared to the cars where the damage has been rectified. So, there is a clear cut difference between the two categories and their effect on price, and hence we are going to retain the variable `notRepairedDamage`, right.

(Refer Slide Time: 22:00)

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** E:\ODP-GITA\INTEL Python for Data Science\Case Study\Regression-Case Study.py
- Variable explorer:** Shows data frames: cars (42772, 10), cars\_copy (42772, 10), and cars\_data (50000, 10). A list variable col is shown with values: ['seller', 'offerType', 'abtest'].
- Python console:** Displays a boxplot titled 'notRepairedDamage' with values ranging from 0 to 40000. The plot shows two distinct groups of data points, one centered around 0 and another centered around 20000.
- Console:** Shows the execution of code to drop columns and copy data frames.
- Bottom status bar:** Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 257, Column: 1, Memory: 39 %.

So, now, I am going to just remove this insignificant variables here which are seller offer type and abtest from the car. So, I am just collecting their names as a list and I am using that cars.drop function, and you can see that from 13 we are down to 10 and that is from the cars data frame, right. So, before we go further let us just make a copy of this data here. So, this is just to make sure that we do not end up losing the analysis that we have done till now.

(Refer Slide Time: 22:30)

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** E:\ODP-GITA\INTEL Python for Data Science\Case Study\Regression-Case Study.py
- Variable explorer:** Shows data frames: cars (42772, 10), cars\_copy (42772, 10), and cars\_data (50000, 10). A data frame cars\_select1 is also listed.
- Python console:** Displays the creation of cars\_select1 and its dtypes.
- Console:** Shows the execution of code to calculate correlations and sort them.
- Bottom status bar:** Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 264, Column: 1, Memory: 100 %.

So, before we move further let us look at the correlation between our numerical variables and I am using `.select_dtypes` which will select a certain ranges of data based on the data type that we have, and for this under the exclude parameter I am excluding all variables which are of the type object.

(Refer Slide Time: 22:52)

```

230 # Variable notRepairedDamage
231 # year car is damaged but not rectified
232 # no - car was damaged but has been repaired
233 cars['notRepairedDamage'].value_counts()
234 pd.crosstab(cars['notRepairedDamage'],
235 sns.countplot(x='notRepairedDamage',
236 # As expected, the cars that require repair fall under lower price ranges
237
238
239 # =====
240 # Removing insignificant variables
241 # =====
242 col=['seller','offerType','attest']
243 cars=cars.drop(columns=col, axis=1)
244 cars_copy=cars.copy()
245
246 # Correlation
247 # =====
248
249 cars_select1=cars.select_dtypes(exclude=[object])
250 correlation=cars_select1.corr()
251 round(correlation,3)
252 cars_select1.corr().loc[:, 'price'].abs().sort_values(ascending=False)[1:]
253
254
255 # =====
256 # Removing insignificant variables
257 # =====
258 col=['seller','offerType','attest']
259 cars=cars.drop(columns=col, axis=1)
260 cars_copy=cars.copy()
261
262 # Correlation
263 # =====
264 cars_select1=cars.select_dtypes(exclude=[object])
265 correlation=cars_select1.corr()
266 round(correlation,3)
267 cars_select1.corr().loc[:, 'price'].abs().sort_values(ascending=False)[1:]
268
269 # =====
270 # OMITTING MISSING VALUES
271 # =====
272
273

```

Once I do that you can see that `cars.select_dtypes` gets displayed and we have 4, columns which are price, powerPS, kilometer and age and I want to calculate the correlation between them.

(Refer Slide Time: 22:57)

```

243 sns.countplot(x='notRepairedDamage',data=cars)
244 sns.countplot(x='notRepairedDamage',y='price',data=cars)
245 # As expected, the cars that require the damages to be repaired fall under lower price ranges
246
247
248
249
250 # =====
251 # Removing insignificant variables
252 # =====
253
254 col=['seller','offerType','attest']
255 cars=cars.drop(columns=col, axis=1)
256 cars_copy=cars.copy()
257
258
259 # =====
260 # Correlation
261 # =====
262
263 cars_select1=cars.select_dtypes(exclude=[object])
264 correlation=cars_select1.corr()
265 round(correlation,3)
266 cars_select1.corr().loc[:, 'price'].abs().sort_values(ascending=False)[1:]
267
268
269 # =====
270 # OMITTING MISSING VALUES
271 # =====
272
273

```

So, I am using the.corr function for this and I am rounding off the correlation to 3 decimal places. So, you have the variables again on the x and on the y axis, you have the same variables, it is a matrix, it is a correlation matrix and from here it is little hard to interpret which of the variables have the highest correlation value.

So, I have cars underscore select.corr which returns the correlation and I am from there I am doing a.location function based on the price, right and this just returns the price and the first column. So, all that returns is the first column, right. And I am just taking the absolute value of it and I am just sorting them in descending order.

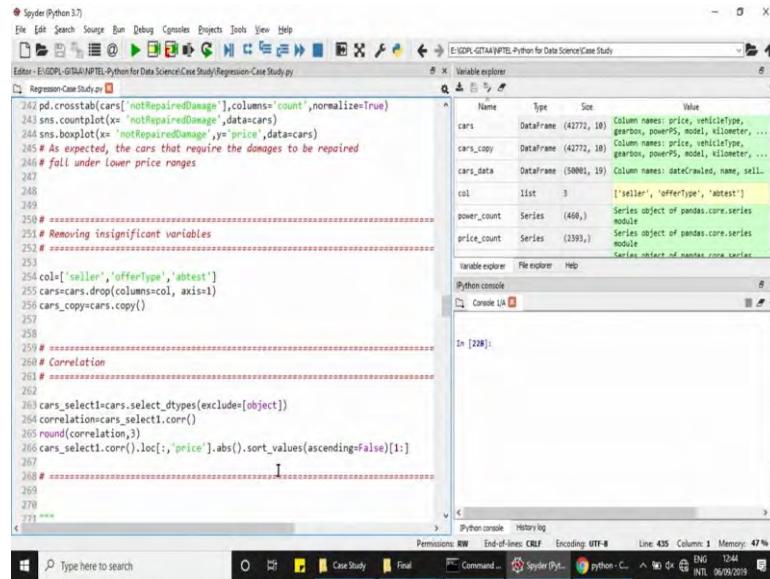
So, I want values after price and you can see that this is the correlation and powerPS has the highest correlation which is about 0.575 followed by that you have kilometer at 0.44 and age is correlated to price by the value of 0.336, right. These values are not high except for powerPS which shows a reasonable correlation. This is only, this method was to only check if there is some, if there is a heavy dependency of one of these variables or one of these numerical variables with price and that was the only reason to check the correlation.

So, except for probably powerPS, the other two variables do not have a very very heavy influence on price. Again the value of the powerPS, again the correlation value of powerPS and price is less, but it is not too less, it is referring about 0.57 and that is ok, but it is not very very strongly correlated to price.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 28**  
**Case study on regression Part III**

(Refer Slide Time: 00:15)



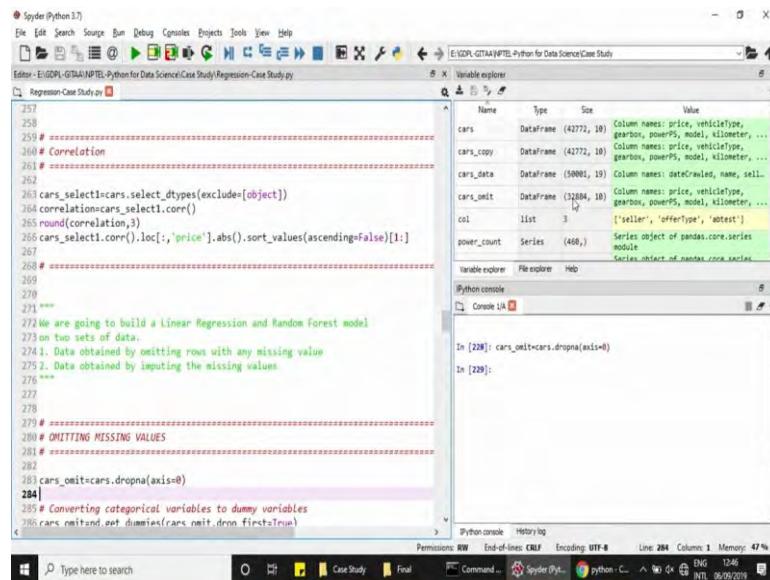
```

Spyder (Python 3.7)
File Edit Search Source Run Debug Cognos Projects Tools View Help
Editor - E:\GDP\GITAA\INTEL\Python for Data Science\Case Study\Regression-Case Study.py
Regresson-Case Study.py
242 pd.crosstab(cars['notRepairedDamage'],columns='count',normalize=True)
243 sns.countplot(x='notRepairedDamage',data=cars)
244 sns.boxplot(x='notRepairedDamage',y='price',data=cars)
245 # As expected, the cars that require the damages to be repaired
246 # fall under lower price ranges
247
248
249
250 # ****
251 # Removing insignificant variables
252 # ****
253
254 cols=['seller','offerType','abitest']
255 cars=cars.drop(columns=cols, axis=1)
256 cars_copy=cars.copy()
257
258
259 # ****
260 # Correlation
261 # ****
262
263 cars_select1=cars.select_dtypes(exclude=[object])
264 correlations=cars_select1.corr()
265 round(correlation,3)
266 cars_select1.corr().loc[:, 'price'].abs().sort_values(ascending=False)[1:]
267
268 # ****
269
270
271 ***

```

So, now we have seen how to compute correlation. So, let us go ahead with model building.

(Refer Slide Time: 00:19)



```

Spyder (Python 3.7)
File Edit Search Source Run Debug Cognos Projects Tools View Help
Editor - E:\GDP\GITAA\INTEL\Python for Data Science\Case Study\Regression-Case Study.py
Regresson-Case Study.py
257
258
259 # ****
260 # Correlation
261 # ****
262
263 cars_select1=cars.select_dtypes(exclude=[object])
264 correlations=cars_select1.corr()
265 round(correlation,3)
266 cars_select1.corr().loc[:, 'price'].abs().sort_values(ascending=False)[1:]
267
268 # ****
269
270
271 ***
272 We are going to build a Linear Regression and Random Forest model.
273 on two sets of data.
274 1. Data obtained by omitting rows with any missing value
275 2. Data obtained by imputing the missing values
276 ***
277
278
279 # ****
280 # OMITTING MISSING VALUES
281 # ****
282
283 cars_omit=cars.dropna(axis=0)
284
285 # Converting categorical variables to dummy variables
286 cars_mitnd=pd.get_dummies(cars_mit,dropna=True)

```

So, we are going to be looking into two algorithms; one is linear regression and other is the random forest regression and for each of these algorithms we are going to use two sets of data. Now, the first data is obtained by removing any missing value. So, any row with any number of missing value gets omitted. So, even if it is a single row with one missing value the entire row gets omitted. Now, the other data is where I impute the missing values. So, I am not going to let go of the missing cells. I am going to come up with some algorithm that is going to help me impute the missing value.

So, these are the two sets of data and for each of these set we are going to use a linear regression model and a random forest model. So, let us start with the first theta which is by omitting missing rows right. So, the command that you are going to use to drop any data with a missing value is `dropna()`. I am setting the axis as equal to 0. Because I want to drop any row that contains any number of missing value be it 1 or be it all cells missing right.

And once the command is executed so, the remaining data gets saved on to `cars OMIT`. We started with 42772 records and after removing all rows which were missing it came down to 32884 and that is roughly about 10000 records that is been removed. So, once it is omitted `cars OMIT` and you can see that around 10000 records have been removed.

(Refer Slide Time: 01:53)

The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```

260 # Correlation
261 # *****
262 cars=cars.select_dtypes(exclu
263 correlation=cars.select1.corr()
264 round(correlation,3)
265 cars_select1.corr().loc[:, 'price'].ab
266
267 # *****
268
269
270
271 ***
272 We are going to build a Linear Regre
273 on two sets of data.
274 1. Data obtained by omitting rows wit
275 2. Data obtained by imputing the miss
276 ***
277
278
279 # *****
280 # OMITTING MISSING VALUES
281 # *****
282
283 cars OMIT=cars.dropna(axis=0)
284
285 # Converting categorical variables to
286 cars OMIT=pd.get_dummies(cars OMIT,drop_Tinst=True)
287
288 # *****
289 # IMPORTING NECESSARY LIBRARIES

```

The variable explorer window shows the following data frame:

| Index | price | vehicleType   | gearbox   | powerPS | model    |
|-------|-------|---------------|-----------|---------|----------|
| 1     | 13239 | SUV           | manual    | 155     | cc_reith |
| 3     | 4500  | small car     | manual    | 86      | ibiza    |
| 4     | 18750 | SUV           | automatic | 185     | cc_reith |
| 5     | 968   | limousine     | manual    | 98      | passat   |
| 7     | 1399  | coupe         | manual    | 136     | clk      |
| 8     | 4600  | station wagon | manual    | 122     | vectra   |
| 9     | 8340  | limousine     | automatic | 140     | octavia  |
| 10    | 1878  | limousine     | manual    | 92      | a_klass  |
| 11    | 2580  | coupe         | manual    | 105     | astr     |
| 12    | 950   | small car     | manual    | 68      | yaris    |
| 14    | 2100  | limousine     | manual    | 83      | meriva   |
| 16    | 1300  | SUV           | manual    | 181     | others   |
| 18    | 600   | small car     | manual    | 75      | golf     |
| 19    | 600   | limousine     | manual    | 88      | volvo    |

The status bar at the bottom indicates the current line is 284, column 1, memory usage is 47%, and the encoding is UTF-8.

So, we have columns that are categorical in nature for instance vehicle type, gearbox or model. Now, the values under these columns are strings in nature and none of the

machine learning algorithms under scikit learn except columns which are strings by nature. So, I am going to dummy encode these categorical columns and I am going to generate newer columns out of it.

(Refer Slide Time: 02:15)

The screenshot shows the Spyder Python IDE interface. The code editor on the left contains Python code for data processing, including correlation analysis and handling missing values. The main area displays a DataFrame titled 'cars\_omit' with columns: Index, kilometer, fuelType, brand, and rmprepareddamg. The DataFrame viewer shows several rows of data, such as a row for index 1 with kilometer=150000, fuelType=diesel, brand=valvo, and rmprepareddamg=13.1. To the right of the DataFrame, there is a 'Variable explorer' window showing various variables like 'gearbox', 'powerPS', 'model', 'kilometer', etc., with their corresponding data types and values.

```

Correlation

cars_select1=cars.select_dtypes(exclude=[object])
correlation=cars_select1.corr()
round(correlation,3)
cars_select1.corr().loc[:,['price']].abs

We are going to build a Linear Regression on two sets of data.
Data obtained by omitting rows with missing values
Data obtained by imputing the miss

OMITTING MISSING VALUES

cars_omit=cars.dropna(axis=0)
Converting categorical variables to dummies
cars_omitpd.get_dummies(cars_omit,drop_first=True)

IMPORTING NECESSARY LIBRARIES

```

For instance, let us say if your fuelType is diesel, petrol and you have CNG and LPG you have four categories here, then I am going to generate 4-1 which is 3 more columns out of this; one column will have fuelType\_diesel, the other column will be fuelType\_petrol and similarly, you have columns generated on it. Now, the value under each of these column will be 0 or 1; now, 0 if it is not present and 1 if the value is present.

For instance, let us take the column fuelType. Now, this column has several values. Let us say you have a column called fuelType\_diesel and if diesel is present under this then that column will be then that cell will be marked as 1, otherwise it will be marked as 0. So, ideally we are trying to encode all of these categories as 0s or 1s alone and we are going to get rid of the strings. So, in order for us to do that I am going to use the get dummies function from the panda's library and I am going to drop the first category.

So, if I run it, you will see there are 301 columns in total. So, we started with 10 and then we have 301 more. And, the reason is because you have several categories under each column and especially variables like brand and model have several categories under them and hence you are getting more number of columns.

(Refer Slide Time: 03:47)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
275 # Data obtained by imputing the missing values
276 ...
277
278
279 # *****
280 # OMITTING MISSING VALUES
281 #
282 cars_omit=cars.dropna(axis=0)
283
284 # Converting categorical variables to dummy variables
285 cars_omit=pd.get_dummies(cars_omit,drop_first=True)
286
287
288 # *****
289 # IMPORTING NECESSARY LIBRARIES
290 #
291
292 from sklearn.model_selection import train_test_split
293 from sklearn.linear_model import LinearRegression
294 from sklearn.ensemble import RandomForestRegressor
295 from sklearn.metrics import mean_squared_error
296
297
298
299 # *****
300 # MODEL BUILDING WITH OMITTED DATA
301 #
302
303 # Separating input and output features
304 v1 = cars_omit.drop(['name'], axis='columns', inplace=False)
```

The variable explorer shows several data frames and series:

- cars: DataFrame (42772, 10) Column names: price, vehicleType,...
- cars\_copy: DataFrame (42772, 10) Column names: price, vehicleType,...
- cars\_data: DataFrame (50000, 19) Column names: dateCrawled, name,...
- cars\_omit: DataFrame (32084, 30) Column names: price, powerPS, kilome...
- col: list [ 'seller', 'offerType', 'adtest' ]
- power\_count: Series (400,) Series object of pandas.core.series module

The IPython console shows the execution history:

```
In [208]: cars_omit=cars.dropna(axis=0)
In [209]: cars_omit=pd.get_dummies(cars_omit,drop_first=True)
In [210]: from sklearn.model_selection import train_test_split
In [211]: from sklearn.linear_model import LinearRegression
In [212]: from sklearn.ensemble import RandomForestRegressor
In [213]: from sklearn.metrics import mean_squared_error
In [214]:
```

The status bar at the bottom indicates the current line (Line 296), column (Column 1), and memory usage (47%).

So, before we proceed let us just begin by importing some of the necessary libraries that we might need. Now, I am importing the train test split from `sklearn.model_selection`. Now, in now because we are going to go about building a model we will need a train set on which we are going to build a model and you also need a test set of data on which we are going to test it. So, given this data `cars_omit`, I am going to reserve a certain part of it for training and building my model I am going to reserve the remaining part to test my model on and for that you need the `train_test_split` function.

And I need the `LinearRegression` function from `sklearn.linear_model` and like how the name suggests this is for building a linear regression model. And similarly for a random forest model I want the `RandomForestRegressor` and this is a part of `sklearn.ensemble` package and apart from these two there are a few heuristics that I am going to calculate and one of it is mean squared error and that is a part of the `sklearn.metrics` package.

So, I am going to import these four libraries of packages before I proceed. So, let us begin by separating the input features and the output features.

(Refer Slide Time: 05:05)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
285 # Converting categorical variables to dummy variables
286 cars_omit=pd.get_dummies(cars_omit,drop_first=True)
287
288 # **** IMPORTING NECESSARY LIBRARIES ****
289
290 from sklearn.model_selection import train_test_split
291 from sklearn.linear_model import LinearRegression
292 from sklearn.ensemble import RandomForestRegressor
293 from sklearn.metrics import mean_squared_error
294
295 # **** MODEL BUILDING WITH OMITTED DATA ****
296
297 # Separating input and output features
298 x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
299 y1 = cars_omit['price']
300
301 # Plotting the variable price
302 prices = pd.DataFrame({'1. Before':y1, '2. After':np.log(y1)})
303 prices.hist()
304
305 # Transforming price as a Logarithmic value
306 y1 = np.log(y1)
307
```

The Variable explorer pane shows the following data structures:

| Name        | Type      | Size         | Value                                                                      |
|-------------|-----------|--------------|----------------------------------------------------------------------------|
| cars_omit   | DataFrame | (32884, 301) | Column names: price, powerPS, kilome...                                    |
| col         | list      | 3            | ['seller', 'offerType', 'test']                                            |
| power_count | Series    | (400,)       | Series object of pandas.core.series module                                 |
| price_count | Series    | (2993,)      | Series object of pandas.core.series module                                 |
| x1          | DataFrame | (32884, 300) | Column names: powerPS, kilometer, Age, vehicleType, carID, vehicleType ... |
| y1          | Series    | (32884,)     | Series object of pandas.core.series module                                 |

The Python console pane shows the execution history:

```
In [288]: cars_omit=cars.dropna(axis=0)
In [289]: cars_omit=pd.get_dummies(cars_omit,drop_first=True)
In [290]: from sklearn.model_selection import train_test_split
In [291]: from sklearn.linear_model import LinearRegression
In [292]: from sklearn.ensemble import RandomForestRegressor
In [293]: from sklearn.metrics import mean_squared_error
In [294]: In [295]: x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
In [296]: y1 = cars_omit['price']
In [297]: In [298]:
```

Now, my input features I am going to call them as x1 and my output feature which is price I am going to call it as y1. So, from cars.omit I am going to drop only price and I am dropping column. So, I am just mentioning it as columns and I am just saying in place equal to false I do not want the changes to be reflected under cars\_omit right.

So, let us just drop only price. So, basically x1 has everything apart from price. So, we so, cars\_omit had 301 columns and one column alone we have removed which is price and hence you have 300 columns under x1 contrary to that we want y1 to contain only one column which is price. So, I am just saving cars\_omit and within square indices I am just giving price which makes it y as only column.

(Refer Slide Time: 06:01)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the following content:

```
285 # Converting categorical variables to
286 cars_omit=pd.get_dummies(cars_omit,drop='category')
287
288 # **** IMPORTING NECESSARY LIBRARIES ****
289 # ****
290 from sklearn.model_selection import train_test_split
291 from sklearn.linear_model import LinearRegression
292 from sklearn.ensemble import RandomForestRegressor
293 from sklearn.metrics import mean_squared_error
294
295 # MODEL BUILDING WITH OMITTED DATA
296 # ****
297 # Separating input and output feature
298 x1 = cars_omit.drop(['price'], axis=1)
299 y1 = cars_omit['price']
300
301 # Plotting the variable price
302 prices = pd.DataFrame({'1. Before':y1,
303 '2. After':np.log(y1)})
304
305 # Transforming price as a logarithmic value
306 y1 = np.log(y1)
307
```

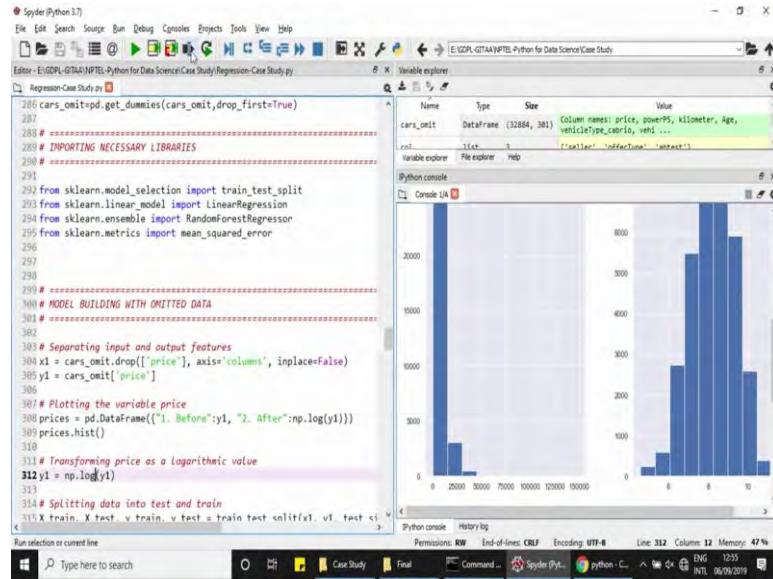
The IPython console shows the creation of a DataFrame 'prices' with two columns: '1. Before' and '2. After'. The '2. After' column contains the natural logarithm of the original 'price' values.

| Index | 1. Before | 2. After |
|-------|-----------|----------|
| 1     | 13299     | 9.49944  |
| 3     | 4580      | 8.41183  |
| 4     | 18750     | 9.83995  |
| 5     | 980       | 6.09950  |
| 7     | 1399      | 7.24351  |
| 8     | 4000      | 8.45105  |
| 9     | 8340      | 9.82082  |
| 10    | 1870      | 7.53369  |
| 11    | 2500      | 7.82485  |
| 12    | 990       | 6.8977   |
| 14    | 2200      | 7.69621  |
| 16    | 1380      | 7.17912  |
| 18    | 600       | 6.39693  |
| 20    | 600       | 6.52289  |

And, you have the same number of observations for price. Now, this is. So, x1 is a collection of all input features and y1 is my output feature which is price. Now, I am just going to plot the variable price now before I plot it I am just converting it to a data frame which says before and after. So, y1 and then I have transformed y1 as a log as a natural logarithm.

So, let me just run this command. So, basically you will have two column under prices one is before, the other is after. Before is what you have as y1 and after is after taking the natural logarithm. Now, the reason we are doing it, it will be clear when I plug the histogram which is below.

(Refer Slide Time: 06:53)



Let me just drag the screen to show you the difference between plotting histogram with  $\pi$  and plotting histogram with logarithmic of  $\pi$ . So, here you can see there is a nice bell shaped curve the moment you plot the x-axis that is natural logarithm of y whereas, on my left I have just done a histogram on just the price whereas, on the right I have a logarithm of price. Now, here you can see that the shape the right hand side is more bell shaped normal compared to the left; the left is most cute and this itself tells you that you know we should consider going further with our natural log on the prices and not just with price.

So, we are not going to regress price with the input features, rather we are going to regress the log of price with input features and this log is the natural log. So, let us now transform the price as a logarithmic value. So, I do `np.log(y1)`. So, my entire `y1` is now transformed right. This is because the range of price is very very huge because you want to avoid it you are taking the log.

(Refer Slide Time: 08:19)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
Transforming price as a Logarithmic value
y1 = np.log(y1)
Splitting data into test and train
X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_size=0.3, random_state=3)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

BASELINE MODEL FOR OMITTED DATA

We are making a base model by using test data mean value
This is to set a benchmark and to compare with our regression model

finding the mean for test data value
base_pred = np.mean(y_test)
print(base_pred)
Repeating same value till Length of test data
base_pred = np.repeat(base_pred, len(y_test))
finding the RMSE
base_root_mean_square_error = np.sqrt(mean_squared_error(y_test, base_pred))
print(base_root_mean_square_error)

```

The variable explorer shows the following data frames:

- X\_test: DataFrame (9866, 300) Column names: powerPS, kilometer,...
- X\_train: DataFrame (23018, 300) Column names: powerPS, kilometer,...
- base\_pred: float64 (9866,) [8.24961579 8.24961579 ...]
- cars: DataFrame (42772, 10) Column names: price, vehicleType,...
- cars\_copy: DataFrame (42772, 10) Column names: price, vehicleType,...
- cars\_data: DataFrame (50001, 19) Column names: dateCreated, name,...
- cars\_omit: DataFrame (32884, 301) Column names: price, powerPS, kilometer,...

The Python console shows the execution of the code:

```
In [298]: y1 = np.log(y1)
In [299]: X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_size=0.3)
In [300]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
In [301]: (23018, 300) (9866, 300) (23018,) (9866,)

In [302]: 0.699956720593082
Out[302]: 0.699956720593082
```

Now, I am splitting my data into testing chain, I am just dragging the output window to the right just to show you the entire command. So, the train underscores test\_split is the function which we have already imported. The input to it is the input set of features, the output set of features which is x1 and y1 respectively. Now, the test side I am giving it as 0.3 and which means 70 percent of the data will go to the train set and the remaining 30 percent will be retained as the test set and I am setting a random state of equal to 3.

Now, random state is a predefined algorithm it is called pseudo random number generator; you can give any value to this parameter and I am giving a value of 3. So, every time you run the same algorithm the same set of records will go to train and test. So, the output I am saving it as X\_train X\_test, y\_train and y\_test. So, the X's represent the input set of features and the y's that represent the output feature which is price. So, let me just run this command and once I run the command I am just printing the shape of each of these. So, my X\_train is 23,018 and I can have 300 columns my y\_train is 23018 again.

Similarly, my X\_test is 9866 and I can have 300 columns and my y\_test is 9866. So, this is the shape of the data if you do 23818 divided by 32884 that is roughly about 70 percent right. Similarly, if you do 9866 divided by 32884 that is a free about 30 percent right. You can infer the test train split ratio from here itself based on the number of observations.

So, before we move further we want to build a baseline model for this data and in the baseline model the predicted value is basically replaced with the mean value of the test data and this is to actually set the benchmark for us and to compare our models that we are going to build in future. So, for this data we are going to build a regression and a random forest model and in order to compare the metrics from that model we also need a base value or a base metric. And, the base metric that we are going to use is the RMSE value for the base model.

So, the base model is where you replace the predicted value as the mean from the test value and that I am saving it as `base_pred` right and that comes out to be around 8.25 and I am going to. So, this is only one value and I wanted repeated to 9866 values which is the length of `y_test`. So, I am just replicating this value over 9866 records.

Now, I want to find the RMSE value. Now, RMSE is root means squared error and it is a like the name suggests is the square root of the mean squared error. So, it computes a between the test value and the predicted value squares them and divides them by the number of sample. Now, that is mean square error; the moment you take a root of that value you get the RMSE value which is the root mean square error. So, you already have a function called `mean_square_error` which we have imported earlier.

Now, the input to it is the test data right the `y_test` and the base predictions right. So, these are the two value that I am going to give and once I computed I am just going to print it. So, the base RMSE turns out to be 1.13 roughly. So, this is a benchmark for comparison. Any other models that are going to build in future should give you an RMSE that is less than this. So, that is the objective for us.

(Refer Slide Time: 12:11)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor contains Python code for linear regression. The code includes importing pandas, setting intercept to true, fitting the model, predicting on test data, computing MSE and RMSE, and performing regression diagnostics. A Jupyter Notebook cell is also visible, showing the execution of the code and the resulting output, which includes a DataFrame named 'cars' with columns 'price', 'seller', and 'offerType', and a variable 'lin\_rmse' with a value of 0.5454.

```
LINEAR REGRESSION WITH OMITTED DATA
Setting intercept as true
lgr=linearRegression(fit_intercept=True)
Model
model_lin=lgr.fit(X_train,y_train)
Predicting model on test set
cars_predictions_lin1 = lgr.predict(X_test)
Computing MSE and RMSE
lin_mse1 = mean_squared_error(y_test, cars_predictions_lin1)
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)
R squared value
r2_lin_test1=model_lin.score(X_test,y_test)
r2_lin_train1=model_lin.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)
Regression diagnostics- Residual plot analysis
residualsly_tst=cars_predictions_lin1
sns.residplot(x=cars_predictions_lin1, y=residualsly_tst, scatter=True,
 fit_reg=False, data=cars)
residuals1.describe()
RANDOM FOREST WITH OMITTED DATA
```

In [281]: lin\_mse1 = mean\_squared\_error(y\_test, cars\_predictions\_lin1)  
In [282]: lin\_rmse1 = np.sqrt(lin\_mse1)  
In [283]: print(lin\_rmse1)  
0.545401266513864  
In [284]: r2\_lin\_test1=model\_lin.score(X\_test,y\_test)  
In [285]: r2\_lin\_train1=model\_lin.score(X\_train,y\_train)  
In [286]: print(r2\_lin\_test1,r2\_lin\_train1)  
0.765815891649225 0.7800936978183916  
In [297]:

So, now let us start with linear regression for this data. Now, I am going to call this the omitted data because I have removed all missing values from here. So, the function that you are going to invoke first is called the linear regression function. I am setting the intercept as true because I want an intercept for this and I am saving it on to an object called lgr. Once you do that you need to fit the model on the train X and train y; train X being the input features, train y being the output features for the train set of data. And, I am storing the model as model\_linear\_1; linear has represented by lin.

And, once I do that I want to predict my model on the test set of input features. So, my\_test set of input features are the X\_test and I am going to use the.predict function and I am saving my predictions on to cars\_predictions\_linear\_1; linear again is represented by lin1. So, here you can see you have 9866 predictions right. Now, these are my predictions. Now, I am again going to compute my mean squared error.

Now, instead of giving a base predicted value, I am going to pass on the actually predicted values here and again the other parameter that I am passing is the test set of features that I have from my\_test set. Now, again I need RMSE value. So, I am just computing the RMSE by taking a square root of this and if you print it you will see that my RMSE for this model which is a linear regression model is 0.54.

So, we started at a value of 1.12 with the base RMSE and we came down to about 0.54 that is roughly about a 50 percent drop right there is a 50 percent change from this value.

So, we have come down a long way from there we have started. So, we are at 0.54. Now, let us see if we can compute the R squared value. So, the R squared value tells you, so how good is your model able to explain the variability in the y. Now, let us do this for test and train.

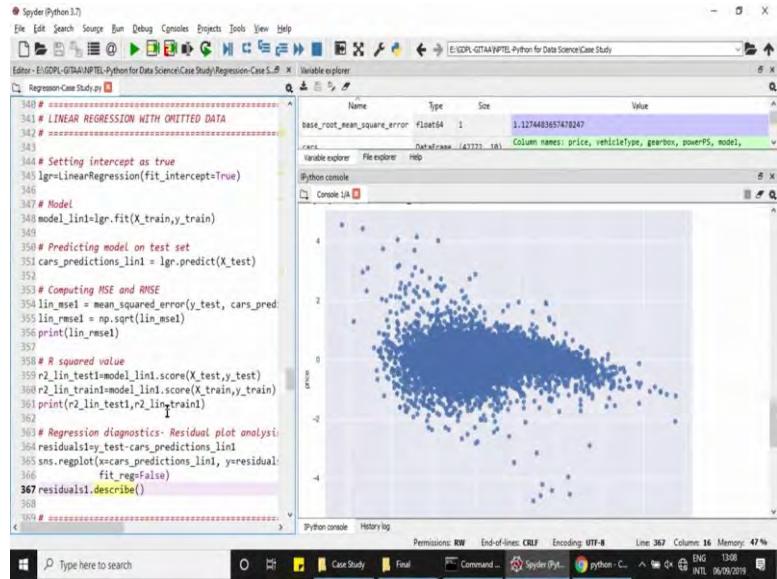
Now, I am going to call the function.score from the model\_lin1 and the input to it is X\_test and x y\_test and similarly, if you want to r squared value for your train you just give the train input. Now, I am just calculating the value of for test and for train and let us just print it out.

So, if you print out both these values at test side for the test set my R square is 0.766 and for my\_train set the R square is 0.780; cutting quite close, but the train seems to be definitely better and the test is not far behind, it is about 0.766. So, that is a close cut. So, this tells you that the model is good. As much as the variability that the model was able to capture in the train data, it is able to capture the same amount of variability if not more on the test data itself, on the test data as well.

Now, let us just do a quick regression diagnostics. Now, I am going to calculate something called residuals here. Now, the residual is nothing, but the difference between your test data and your prediction. Now, the residual is nothing, but the difference between your predicted value and your actual value. Now, your actual values under y\_test and your predicted value is under is under cars\_predictions\_linear1.

Now, once you do this you can plot I am once you do this I am going to generate a residual plot using a simple scatter plot where my x is the predicted the fitted value, that is, cars\_predictions\_linear1 and my y is nothing, but the residuals.

(Refer Slide Time: 16:25)



The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```

LINEAR REGRESSION WITH OMITTED DATA

Setting intercept as true
lgr=linearRegression(fit_intercept=True)
Model
model_lini=lgr.fit(X_train,y_train)
Predicting model on test set
cars_predictions_lini = lgr.predict(X_test)

Computing MSE and RMSE
lin_msel = mean_squared_error(y_test, cars_predictions_lini)
lin_rmse = np.sqrt(lin_msel)
print(lin_rmse)

R squared value
r2_lini_testi=model_lini.score(X_test,y_test)
r2_lini_traini=model_lini.score(X_train,y_train)
print(r2_lini_testi,r2_lini_traini)

Regression diagnostics- Residual plot analysis:
residuals=y_test-cars_predictions_lini
sns.regplot(x=cars_predictions_lini, y=residuals,
 fit_reg=False)
residuals1.describe()
```

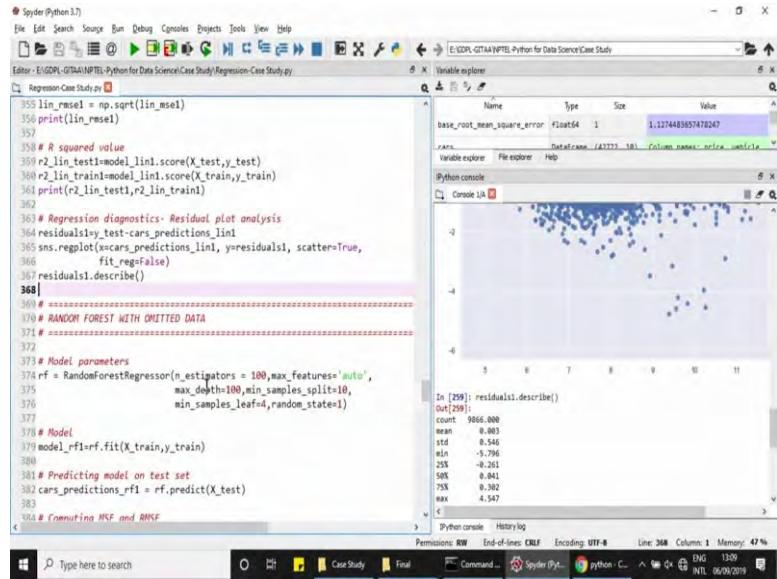
The variable explorer window shows a table with one row and four columns, labeled 'base\_root\_mean\_square\_error'. The value is 1.1274483657478247. The data frame is named 'cars' and contains columns: price, vehicleType, gearbox, powerPS, model, and residuals.

The console window displays a scatter plot of residuals. The x-axis is labeled 'cars\_predictions\_lini' and ranges from approximately -4 to 4. The y-axis is labeled 'residuals' and ranges from -4 to 4. The data points are scattered randomly around the zero line, indicating a good fit of the linear regression model.

So, let us just quickly do a regression diagnostic. I am going to use the residual plot analysis. Now, I am going to generate a separate set of numbers or values that is called residuals1. Now, it is the difference between the y\_test and the cars\_prediction\_linear1. Now, basically residuals are nothing, but the difference of your actual value or the observed value and the predicted value. So, that is called residuals and what I am going to do is I am going to generate a simple scatter plot where my x-axis is basically the fitted or the predicted values and my y-axis contains the residual values.

So, let us just do a quick scatter plot here. So, I am going to drag the output window little to show you how does the plot look. This is good because you want the residuals to be close to 0. Since, residuals are more or more of errors you want the errors to be less and that tells you that your predicted and your actual values are very very closer and this can also be seen from the describe function. So, if you do a residuals1.described this gives you the summary of the data.

(Refer Slide Time: 17:37)



```
355 lin_rmse1 = np.sqrt(lin_mse1)
356 print(lin_rmse1)
357
358 # R squared value
359 r2_lin_test1= model_lini.score(X_test,y_test)
360 r2_lin_train1= model_lini.score(X_train,y_train)
361 print(r2_lin_test1,r2_lin_train1)
362
363 # Regression diagnostics- Residual plot analysis
364 residuals1iy_test=cars_predictions_lini
365 sns.regplot(x=cars_predictions_lini, y=residuals1, scatter=True,
366 fit_reg=False)
367 residuals1.describe()
368
369 # *****
370 # RANDOM FOREST WITH OMITTED DATA
371 # *****
372
373 # Model parameters
374 rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
375 max_depth=100,min_samples_split=10,
376 min_samples_leaf=4,random_state=1)
377
378 # Model
379 model_rf=rf.fit(X_train,y_train)
380
381 # Predicting model on test set
382 cars_predictions_rf1 = rf.predict(X_test)
383
384 # Computing MSE and RMSE
```

And you can see that the mean of the data is around 0.003 that is very very less, it is almost 0 right and this itself a good this itself is a good indicator that your predictor and your actual value is a very very closer now. And this is the conclusion that it can actually draw from your residual analysis. So, I have just shown you for one linear regression model and you can just do it with the others as well.

So, let us move ahead with random forest model. Now, a random forest model has several inputs and we have already imported the function called `RandomForestRegressor`. So, I have a couple of parameters here as input arguments. Let us just quickly go over what these mean.

(Refer Slide Time: 18:23)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the following content:

```
355 lin_mse1 = np.sqrt(lin_mse1)
356 print(lin_mse1)
357
358 # R squared value
359 r2_lin_test1=model_lini.score(X_test,y_test)
360 r2_lin_train1=model_lini.score(X_train,y_train)
361 print(r2_lin_test1,r2_lin_train1)
362
363 # Regression diagnostics- Residual plot analysis
364 residuals1y_test=cars_predictions_lini
365 sns.residplot(x=cars_predictions_lini, y=residuals1, scatter=True,
366 fit_reg=False)
367 residuals1.describe()
368
369 # *****
370 # RANDOM FOREST WITH OMITTED DATA
371 # *****
372
373 # Model parameters
374 rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
375 max_depth=100,min_samples_split=10,
376 min_samples_leaf=4,random_state=1)
377
378 # Model
379 model_rf1rf.fit(X_train,y_train)
380
381 # Predicting model on test set
382 cars_predictions_rf1 = rf.predict(X_test)
383
384 # Computing MSE and RMSE
```

The 'Parameters' pane on the right shows the 'n\_estimators' parameter with its default value of 100 and a note about a change in version 0.20. The 'Console' pane shows a scatter plot of residuals.

So, I in my help toolbox I have typed a `sklearn.ensemble.RandomForestRegressor` and it gives you a quick description of what does the function do. But, if you come down it also tells you what are the input parameters and what do they mean. Also mentioned under the parameters are also the default values. So, let us begin with the first parameter it is `n_estimators`. Now, because this is a random forest tree you can have several trees and that is what `n_estimators` does. It basically tells you the number of trees in your forest.

(Refer Slide Time: 19:01)

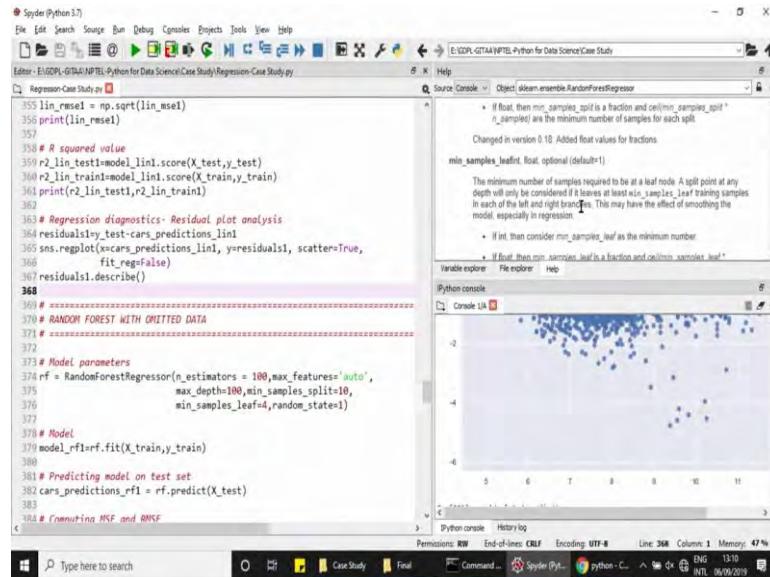
The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the same content as the previous screenshot:

```
355 lin_mse1 = np.sqrt(lin_mse1)
356 print(lin_mse1)
357
358 # R squared value
359 r2_lin_test1=model_lini.score(X_test,y_test)
360 r2_lin_train1=model_lini.score(X_train,y_train)
361 print(r2_lin_test1,r2_lin_train1)
362
363 # Regression diagnostics- Residual plot analysis
364 residuals1y_test=cars_predictions_lini
365 sns.residplot(x=cars_predictions_lini, y=residuals1, scatter=True,
366 fit_reg=False)
367 residuals1.describe()
368
369 # *****
370 # RANDOM FOREST WITH OMITTED DATA
371 # *****
372
373 # Model parameters
374 rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
375 max_depth=100,min_samples_split=10,
376 min_samples_leaf=4,random_state=1)
377
378 # Model
379 model_rf1rf.fit(X_train,y_train)
380
381 # Predicting model on test set
382 cars_predictions_rf1 = rf.predict(X_test)
383
384 # Computing MSE and RMSE
```

The 'Parameters' pane on the right shows the 'n\_estimators' parameter with its default value of 100 and a note about a change in version 0.20. The 'Console' pane shows a scatter plot of residuals.

The next is maximum depth which is the depth of each tree; how deep do you want each tree to go and that is given by maximum depth.

(Refer Slide Time: 19:11)



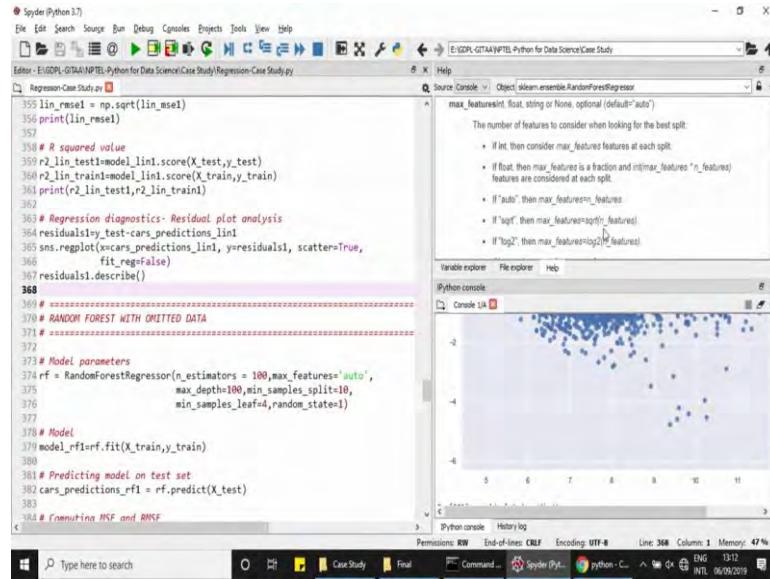
The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the following content:

```
355 lin_msel = np.sqrt(lin_mse)
356 print(lin_msel)
357
358 # R squared value
359 r2_lin_test1=model_lini.score(X_test,y_test)
360 r2_lin_train=model_lini.score(X_train,y_train)
361 print(r2_lin_test1,r2_lin_train)
362
363 # Regression diagnostics- Residual plot analysis
364 residualslyy_test=cars_predictions_lini
365 sns.resplot(x=cars_predictions_lini, y=residuals1, scatter=True,
366 fit_reg=False)
367 residuals1.describe()
368
369 # *****
370 # RANDOM FOREST WITH OMITTED DATA
371 # *****
372
373 # Model parameters
374 rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
375 max_depth=100,min_samples_split=10,
376 min_samples_leaf=4,random_state=1)
377
378 # Model
379 model_rfisrf.fit(X_train,y_train)
380
381 # Predicting model on test set
382 cars_predictions_rf1 = rf.predict(X_test)
383
384 # Computing MSE and RMSE
```

The IPython console window shows a scatter plot of data points. The x-axis ranges from 5 to 11, and the y-axis ranges from -6 to -2. The data points are scattered across this range.

The minimum\_samples\_split is the minimum number of samples that is required for a node to split. So, let us say if you have only two observations in a node and you do not want that node to split any further right because it might not make any sense. So, in that case if you want to impose such conditions you can do it here. The next is minimum\_samples\_leaf and it is the minimum number of samples which is required to be at a leaf node.

(Refer Slide Time: 19:41)



The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
355 lin_mse1 = np.sqrt(lin_mse1)
356 print(lin_mse1)
357
358 # R squared value
359 r2_lin_test1= model_lini.score(X_test,y_test)
360 r2_lin_train1= model_lini.score(X_train,y_train)
361 print(r2_lin_test1,r2_lin_train1)
362
363 # Regression diagnostics- Residual plot analysis
364 residuals1iy_test=cars_predictions_lini
365 sns.regplot(x=cars_predictions_lini, y=residuals1, scatter=True,
366 fit_reg=False)
367 residuals1.describe()
368
369 # *****
370 # RANDOM FOREST WITH OMITTED DATA
371 #
372
373 # Model parameters
374 rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
375 max_depth=100,min_samples_split=10,
376 min_samples_leaf=4,random_state=1)
377
378 # Model
379 model_rf1=rf.fit(X_train,y_train)
380
381 # Predicting model on test set
382 cars_predictions_rf1 = rf.predict(X_test)
383
384 # Computing MSE and RMSE
```

The code includes imports from numpy and scikit-learn, performs linear regression analysis, and then moves on to a Random Forest model with 100 estimators. A scatter plot of residuals is shown in the IPython console.

And, the next input is maximum features which is basically the number of features the algorithm wants to consider to build the model or to build the tree right. Now, there are different options for this. If you give auto, then it automatically chooses otherwise you can give square root which is basically the square root of the number of features. So, each tree is built on different features.

So, there are other parameters as well which you can explore one by one and these are the main parameters that you usually start with. Again random state is equal to 1, because I have just chosen one random state, so that every time I run the model I get the same output.

(Refer Slide Time: 20:29)

The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```
Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
 max_depth=100,min_samples_split=10,
 min_samples_leaf=4,random_state=1)

Model
model_rf1=rf.fit(X_train,y_train)

Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

Computing MSE and RMSE
rf_mse1 = mean_squared_error(y_test, cars_predictions_rf1)
rf_rmse1 = np.sqrt(rf_mse1)
print(rf_rmse1)

R squared value
r2_rf_test1=model_rf1.score(X_test,y_test)
r2_rf_train1=model_rf1.score(X_train,y_train)
print(r2_rf_test1,r2_rf_train1)

MODEL BUILDING WITH IMPUTED DATA

```

The variable explorer shows the following data frames and their sizes:

| Name                 | Type      | Size         | Value                             |
|----------------------|-----------|--------------|-----------------------------------|
| cars_data            | Dataframe | (50000, 19)  | Column names: dateCrawled, n...   |
| cars_out             | Dataframe | (32084, 301) | Column names: price, powerPS...   |
| cars_predictions_rf1 | float64   | (9866,)      | [0.43257524 9.70032094 0.341...   |
| col                  | list      | 3            | ['seller', 'offerType', 'adtest'] |
| lin_mse1             | float64   | 1            | 0.3974227049492871                |

The Python console shows the execution of the code:

```
In [262]: cars_predictions_rf1 = rf.predict(X_test)
In [263]: rf_mse1 = mean_squared_error(y_test, cars_predictions_rf1)
In [264]: rf_rmse1 = np.sqrt(rf_mse1)
In [265]: print(rf_rmse1)
0.4325752391792225
In [266]: r2_rf_test1=model_rf1.score(X_test,y_test)
In [267]: r2_rf_train1=model_rf1.score(X_train,y_train)
In [268]: print(r2_rf_test1,r2_rf_train1)
0.850401814775962 0.9203494705146291
In [269]:
```

So, now let us just load the function. Now, rf is the model; now, I do rf.fit to fit the train data. Now, this will take some time, now the model was built using the train set of input and output features. Now, I am going to predict this model on the test set. Now, if you see under cars\_predictions\_rf1 you again have 9866 predictions ok. Now, we are again to compute the mean square error for this and the rmse turned out to be 0.44 roughly.

Now, this itself is a good indicator that your random forest is performing better than your linear regression model because the RMSE has come down further. Computer the R squared value here. So, let us just print out the R squared value. So, I have the R squared value here now. Now, the R squared value for the test is 0.85 and the a squared value for the train when the train is 0.92. So, definitely random forest is working a lot better compared to the linear regression model at least for this omitted data.

So, now let us go ahead and impute the missing cells in our data and let us now build the same linear and regression, now let us and let us build this linear regression model and random forest model on the data where these missing values have been imputed. So, I am using the same cars data that we are earlier started with. I am using a.apply function and within the apply function I am declaring a lambda function.

Now, the lambda function will fill the missing values with median if the data type of the column is float otherwise it will fill the missing cell with the most frequent category. Now, this the second condition will be applicable wherever your column type is object

and the first condition will be applicable wherever your column type is float; either ways both cells will be imputed.

(Refer Slide Time: 22:27)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays a script named 'Regression-Case Study.py' with the following content:

```

387 print(rf_mse)
388
389 # R squared value
390 r2_rf_test1=model_rf1.score(X_test,y_test)
391 r2_rf_train1=model_rf1.score(X_train,y_train)
392 print(r2_rf_test1,r2_rf_train1)
393
394
395 # =====
396 # MODEL BUILDING WITH IMPUTED DATA
397 # =====
398
399 cars_imputed = cars.apply(lambda x:x.fillna(x.median()))
400 if x.dtype=='float' else \
401 x.fillna(x.value_counts().index[0]))
402 cars_imputed.isnull().sum()
403
404 # Converting categorical variables to dummy variables
405 cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)
406
407
408 # =====
409 # MODEL BUILDING WITH IMPUTED DATA
410 # =====
411
412 # Separating input and output feature
413 x2 = cars_imputed.drop(['price'], axis='columns', inplace=False)
414 y2 = cars_imputed['price']
415
416

```

The variable explorer window shows the following data frames:

| Name                 | Type      | Size         | Value                             |
|----------------------|-----------|--------------|-----------------------------------|
| cars_data            | Dataframe | (50000, 19)  | Column names: dateCrawled, n...   |
| cars_imputed         | Dataframe | (42772, 304) | Column names: price, powerPS...   |
| cars_out             | Dataframe | (32804, 301) | Column names: price, powerPS...   |
| cars_predictions_l1  | float64   | (9866,)      | [0.43257549, 7.88032094, 8.841... |
| cars_predictions_rf1 | float64   | (9866,)      | [0.70182348, 9.57250428, 8.195... |
| col                  | list      | 3            | ['seller', 'offerType', 'adtest'] |

The Python console window shows the following output:

```

In [270]: cars_imputed.isnull().sum()
Out[270]:
price 0
vehicleType 0
gearbox 0
model 0
powerPS 0
odo 0
kilometer 0
fuelType 0
brand 0
notRepairedDamage 0
Age 0
dtype: int64

In [271]: cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)
In [272]:

```

Now, if you impute and then you can check the number of null values under each you will see that none of the columns are missing values. So, in our data we had missing values only for the categorical columns and in that sense you are all these categorical columns are imputed with the most frequently occurring category.

Now, let us convert the categorical columns into dummy variables. So, I am going to use the same get dummies function. So, if you look at cars\_imputed you again have 10 columns and when you do a dummy variable encoding, you get 304 columns. So, these are the number of columns that have been converted to dummy variables.

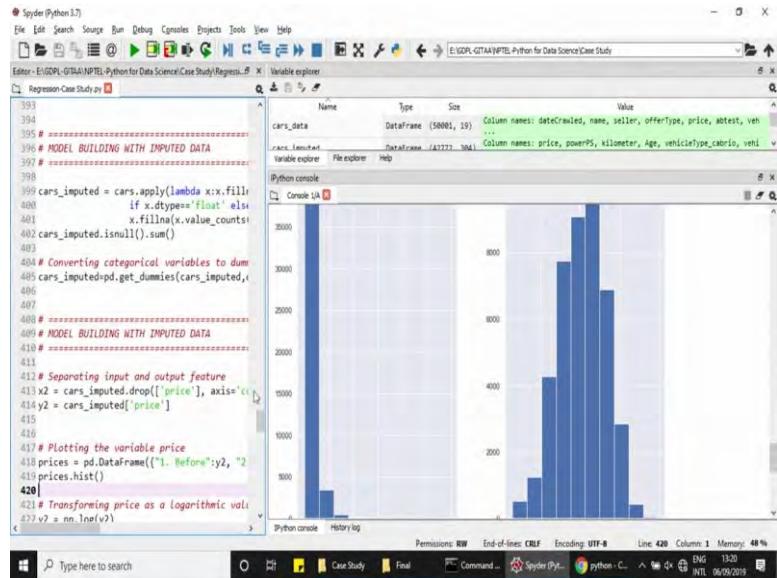
(Refer Slide Time: 23:13)

The screenshot shows the Spyder Python IDE interface. The code editor displays a script named 'Regression-Case Study.py' with several sections of code. The variable explorer on the right shows various data frames and their columns, such as 'cars\_data', 'cars\_imputed', 'cars\_mean', 'cars\_predictions\_lini', 'cars\_predictions\_rf1', and 'col'. The Python console at the bottom shows the execution of code to plot histograms and log-transformed price values.

```
393
394
395 # ****
396 # MODEL BUILDING WITH IMPUTED DATA
397 # ****
398
399 cars_imputed = cars.apply(lambda x:x.fillna(x.median()))
400 if x.dtype=='float' else \
401 x.fillna(x.value_counts().index[0]))
402 cars_imputed.isnull().sum()
403
404 # Converting categorical variables to dummy variables
405 cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)
406
407
408 # ****
409 # MODEL BUILDING WITH IMPUTED DATA
410 # ****
411
412 # Separating input and output feature
413 x2 = cars_imputed.drop(['price'], axis='columns', inplace=False)
414 y2 = cars_imputed['price']
415
416
417 # Plotting the variable price
418 prices = pd.DataFrame(("1. Before":y2, "2. After":np.log(y2)))
419 prices.hist()
420
421 # Transforming price as a Logarithmic value
422 v2 = np.log(y2)
```

So, let us again repeat the same drill where I separate the input features and the output set of features. Now, I am again plotting y2 what you get and np.log of y2.

(Refer Slide Time: 23:21)



And, if you plot a histogram you can again see that value for price is normally distributed and this is the same as before. So, all our predictions are going to be for a log transformed value of price. So, I am going ahead and I am transforming it to a natural logarithm.

(Refer Slide Time: 23:35)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window displays Python code for data processing and model building. The code includes:

```
437 # finding the mean for test data value
438 base_pred = np.mean(y_test1)
439 print(base_pred)
440
441 # Repeating same value till length of test data
442 base_pred = np.repeat(base_pred, len(y_test1))
443
444 # finding the RMSE
445 base_root_mean_square_error_imputed = np.sqrt(mean_squared_error(y_test1, base_pred))
446 print(base_root_mean_square_error_imputed)
447
448 # ****LINEAR REGRESSION WITH IMPUTED DATA****
449 # ****LINEAR REGRESSION WITH IMPUTED DATA****
450 # Setting intercept as true
451 lgr2=LinearRegression(fit_intercept=True)
452
453 # Model
454 model_lin2=lgr2.fit(X_train1,y_train1)
455
456 # Predicting model on test set
457 cars_predictions_lin2 = lgr2.predict(X_test1)
458
459 # Computing MSE and RMSE
460 lin_mse1 = mean_squared_error(y_test1, cars_predictions_lin2)
461 lin_rmse2 = np.sqrt(lin_mse2)
462 print(lin_rmse2)
```

The variable explorer shows a DataFrame named 'Dataframe' with one column 'Value' containing the value 1.18843491128. The Python console shows various imports and function calls related to data shapes and printing.

And then I am going to split the data into test and train again and I am using the same split ratio which is 0.3 and if you print the shape you can see 70 percent of it has gone to train and remaining 30 percent have gone to test. And, here are the individual split up of the test set and the train set for the input and output features.

So, let us again start by building a baseline model. Now, here I am going to use the `y_test1` I am again going to find the base RMSE for this. This is the same function as before and just that by instead of having `y_test`, I am doing `y_test1`. So, that is the only change that is here and in this case we have an RMSE value of 1.18.

So, again let us go ahead with linear regression I am going to repeat the same set of steps. I am building a model I am fitting the model with the train data and then I am predicting the test data. So, this is the RMSE value. So, I am going to use the same function again and if you print the RMSE you can see it is 0.648. Now, after imputing you can see that the RMSE value for the linear regression model has gone up right. It was earlier at around 0.56 for the same data with, but without cells which were missing. So, the error has gone slightly up. Now, let us go ahead with random forest model for the same data.

(Refer Slide Time: 25:01)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the following content:

```
461 # R squared value
462 r2_lin_train= model_1lin2.score(X_test1,y_test1)
463 r2_lin_test2= model_1lin2.score(X_train1,y_train1)
464 print(r2_lin_test2,r2_lin_train)
465
466 # *****
467 # RANDOM FOREST WITH IMPUTED DATA
468 # *****
469
470 # Model parameters
471 rf2 = RandomForestRegressor(n_estimators = 100,max_features='auto',
472 max_depth=100,min_samples_split=10,
473 min_samples_leaf=4,random_state=1)
474
475 # Model
476 model_rf2=rf2.fit(X_train1,y_train1)
477
478 # Predicting model on test set
479 cars_predictions_rf2 = rf2.predict(X_test1)
480
481 # Computing MSE and RMSE
482 rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)
483 rf_rmse2 = np.sqrt(rf_mse2)
484 print(rf_rmse2)
485 r2_rf_test2= model_rf2.score(X_test1,y_test1)
486 r2_rf_train2= model_rf2.score(X_train1,y_train1)
487
488 # *****
489
```

The variable explorer shows a single entry: 'base\_root\_mean\_square\_error\_imputed' of type float64 with a value of 1.18843481128. The Python console shows the execution of the code, including the calculation of the RMSE, which is printed as 0.494352670399689.

I am again building a model with all parameters I have kept the parameters value the same as before just to standardize both the methods and I am again fitting the model on the train and predicting on the test. Again, this will take some time to run. So, the model has run and it is predicted on the test set. So, let us just compute the RMSE again and RMSE that you get in this case is 0.494. That is a lot less compared to what the linear regression model k with imputed data.

(Refer Slide Time: 25:33)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor displays a script named 'Regression-Case Study.py' with the same content as the previous screenshot:

```
471 # *****
472 # Model parameters
473 rf2 = RandomForestRegressor(n_estimators = 100,max_features='auto',
474 max_depth=100,min_samples_split=10,
475 min_samples_leaf=4,random_state=1)
476
477 # Model
478 model_rf2=rf2.fit(X_train1,y_train1)
479
480 # Predicting model on test set
481 cars_predictions_rf2 = rf2.predict(X_test1)
482
483 # Computing MSE and RMSE
484 rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)
485 rf_rmse2 = np.sqrt(rf_mse2)
486 print(rf_rmse2)
487
488 # R squared value
489 r2_rf_test2= model_rf2.score(X_test1,y_test1)
490 r2_rf_train2= model_rf2.score(X_train1,y_train1)
491 print(r2_rf_test2,r2_rf_train2)
492
493 # *****
494
495 # Final output
496
497 print("Metrics for models built from data where missing values were omitted")
498 print("R squared value for train from Linear Regression: %r" % r2_lin_train)
499 print("R squared value for test from Linear Regression: %r" % r2_lin_test1)
```

The variable explorer shows the same entry: 'base\_root\_mean\_square\_error\_imputed' of type float64 with a value of 1.18843481128. The Python console shows the execution of the code, including the calculation of the RMSE, which is printed as 0.494352670399689.

So, again let us compute the R squared value for this and R squared value for the random forest regression module after imputing turns out to be 0.83 for the test set and on the train set it is about 0.9. Now, that is a very good R squared value compared to the linear regression model that we had got earlier and similarly, if even if you look at the RMSE value we had got about 0.49 and that is a good value compared to the linear regression model on the imputed data. So, let us just print all of these and condense it together.

(Refer Slide Time: 26:07)

The screenshot shows the Spyder Python 3.7 IDE interface. The code editor window contains a script named 'Regression-Case Study.py' with the following content:

```

056 print(r2_rf_test2,r2_rf_train2)
057
058 #####
059
060 # Final output
061
062 print("Metrics for models built from")
063 print("R squared value for train from rf")
064 print("R squared value for test from rf")
065 print("R squared value for train from rf")
066 print("R squared value for test from rf")
067 print("Base RMSE of model built from")
068 print("RMSE value for test from Lin")
069 print("RMSE value for test from Rand")
070 print("\n")
071 print("Metrics for models built from")
072 print("R squared value for train from rf")
073 print("R squared value for test from rf")
074 print("R squared value for train from rf")
075 print("R squared value for test from rf")
076 print("Base RMSE of model built from")
077 print("RMSE value for test from Lin")
078 print("RMSE value for test from Rand")
079
080 #####
081 # END OF SCRIPT
082 #####

```

The variable explorer window shows a single entry:

| Name                                | Type    | Size | Value              |
|-------------------------------------|---------|------|--------------------|
| base_root_mean_square_error_imputed | float64 | 1    | 1.1884349112889792 |

The Python console window displays the output of the printed statements. It includes several R-squared values and RMSE values for both linear regression and random forest models, comparing them across training and testing datasets. The output is too long to list here but follows the pattern established in the code.

So, basically I am just printing these values, so that we can infer the message. So, what I am printing here is basically a condensed form of whatever we have computed till now. I have taken two sets; one is a set of metric for the omitted data and the other set of metrics for the imputed data. Now, for the omitted data you can see that the R squared value for train from the linear regression is 0.78, again on test it is 0.76. So, it is more or less close and similarly, for random forest the R square on train is 0.92 and the R squared value on point on test is 0.85.

Now, if you look at the base RMSE value it is for 0.12 and for this data for linear regression the RMSE value turns out to be 0.54 and the RMSE value from random forests turns out to be 0.44 and these are also the heuristics that we have computed earlier. So, this tells you that definitely on at least we omitted data, the random forest is definitely performing a lot better compared to the linear regression model.

Now, moving further to the imputed set of data if you look at the R squared value from the linear regression on train it is 0.70 and it is more or less the same even on the test. And if you look at the r squared value from random forest on the train it is 0.9 and on the test it is a 0.826. So, that is again a good value and it is a lot better compared to your linear regression model. And the base RMSE for this data is at 1.18 and the RMSE value from the linear regression that you record this 0.64, but however, if you look at the RMSE from the random forest it is at 0.49.

So, definitely for the imputed data, there is a huge remarkable change in the RMSE value between your linear regression and random forest model. So, your random forest model is definitely a lot better than your linear regression model and this is very important because it is if the models are performing good for data where missing values are omitted because in that case you have a clean data you are ready full, but most often. But, more often than not problem comes only when you start impute in sells because you are not sure about with which value you suppose to impute and how the results are going to turn out right.

So, at least in this case the imputation looks like the imputation has not gone really wrong and random forests definitely is a lot better than linear regression. And, this is also very and it is also very important to impute values because you do not want to let go of information and you do not want to let go of the number of records that you have. In the earlier case where we did omit we lost about 10000 record, we had around 33000 on records. But, after imputing, you can see that random forest definitely able to capture the variations much better than linear regression.

So, it is a necessary that you do not let go of information that you already have, but at the same time there is a tradeoff between how much you can retain and how much you can leave.

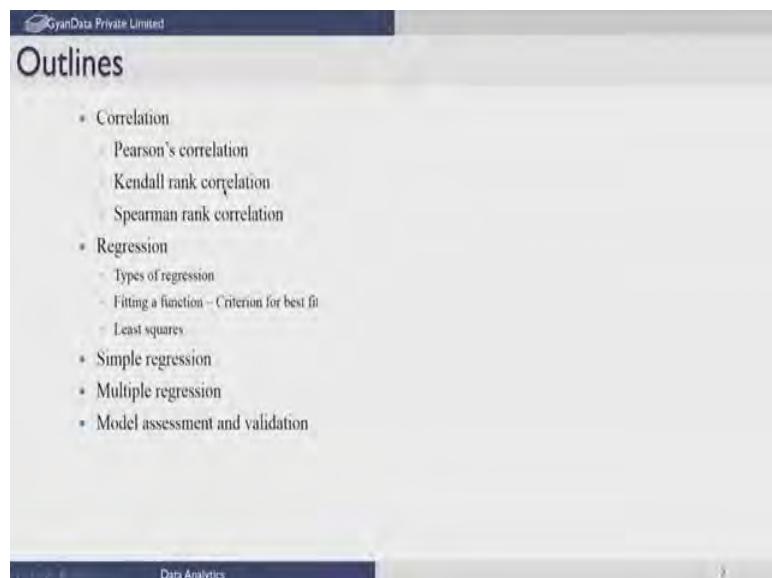
So, thank you.

**Python for Data Science**  
**Prof. Raghunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 29**  
**Module: Predictive Modelling**

Welcome to the lectures in Data Analytics, in this series of lectures I am going to introduce to you Model Building. In particular I will be talking about building linear models using a techniques called regression techniques. So, let us start with some basic concepts, we are going to introduce the notion of correlation.

(Refer Slide Time: 00:36)



Different types of correlation coefficients that have been defined in the literature, what they are useful for. This is a preliminary check you can do, before you start building models. Then I will talk about regression, specifically linear regression and I will introduce the basic notions of regression and then take the case of two variables; before taking going through multi linear regression where there are several input variables and one dependent output variable.

Finally after building the model, we would like to assess how well the model performs, how to validate some of the assumptions we have made and so on. So, this is called model assessment and validation. So, let us first look at some measures of correlation, we have already seen one in the basic introductory lectures to statistics.

(Refer Slide Time: 01:22)

Preliminaries

- $n$  observations for  $x$  and  $y$  variables ( $x_i, y_i$ )
- Sample means  $\bar{x}$  and  $\bar{y}$   
$$\bar{x} = \frac{\sum x_i}{n} \quad \bar{y} = \frac{\sum y_i}{n}$$
- Sample variances  $S_{xx}$  and  $S_{yy}$   
$$S_{xx} = \frac{1}{n} \sum (x_i - \bar{x})^2 \quad S_{yy} = \frac{1}{n} \sum (y_i - \bar{y})^2$$
- Sample covariance  $S_{xy}$   
$$S_{xy} = \frac{1}{n} \sum (x_i - \bar{x})(y_i - \bar{y})$$

So, let us consider  $n$  observations of 2 variables  $x$  and  $y$ . So, denoted by the samples  $x_i, y_i$  and we can of course compute the sample means, we have seen this in statistics, which is just the summation of all values divided by  $n$  for  $x$  which is denoted by  $\bar{x}$ .

And similarly we can do the sample mean of  $y$ , which is denoted by  $\bar{y}$ . We can also define sample variances which is nothing, but the sum square deviation of the individual values from the respective means,  $x_i - \bar{x}$  whole square summed over all the values divided by  $n$  or  $n - 1$  as the case may be. So, we define these sample variances  $S_{xx}$  and  $S_{yy}$  corresponding to the variance of sample variance of  $x$  and  $y$ . We can also define the cross covariance which is denoted by  $S_{xy}$  which is nothing, but the deviation of  $x_i$  from  $\bar{x}$  and the corresponding deviation of  $y_i$  from  $\bar{y}$ .

And the product of this we take and sum over all values and divide by  $n$ . Notice again that this  $x_i$  and  $y_i$  order pairs in the sense that, corresponding to the experimental condition  $i$  th experiment we have obtained values for  $x$  and  $y$  and that is what we have to think we cannot shuffle these values any which way, they have known assumed to be corresponding to some experimental conditions that you have set. So, there are  $n$  experimental observations you have made.

(Refer Slide Time: 03:02)

• Correlation: the strength of association between two variables

• Correlation does not imply causation

• Visual representation of correlation: Scatter grams

Positive trend      Negative trend      Little or no correlation

Quantitative Metric?

Now let us define what we call correlation, correlation is nothing but the indicates the strength of association between the two variables; of course, if you find a strong correlation it does not mean that the one variable is a cause and the other is an effect. You cannot treat correlation as a causation, because there can be a third variable which is basically triggering these two and therefore, you can only find the correlation; but cannot assume that one of the variables is a cause and the other is an effect.

We can also before we actually do numerical computation, we can check whether there is an association between variables by using what is called the scatter plot. We have done this before, so we can plot the values of  $x_i$  on the x axis,  $y_i$  on the y axis and for each of these points and we can see whether these points are oriented in any particular direction. For example, the figure on the left here, indicates that the  $y_i$  increases as  $x_i$  increases there seems to be a trend in this; in particular we can say there is even a linear trend as  $x_i$  increases  $y_i$  correspondingly increases in the linear fashion.

This is a positive trend, because when  $x_i$  increases  $y_i$  increases. In the next figure, the middle figure we show a case where  $x_i$  is as  $x_i$  increases  $y_i$  seems to decrease and again there seems to be a pattern association between  $x_i$  and  $y_i$  and this is a negative trend. Whereas, if you look at the third figure, the data that we find seems to be having no bearing on each other; that is  $y_i$  values do not seem to depend in any particular manner on the  $x_i$  values, when  $x_i$  increases maybe  $y_i$  increases for sometimes some cases and  $y_i$  decreases,

that is why it is spread in all over the place. So, we can say there is little or no correlation. This is a qualitative way of looking at it we can quantify this; and there are several measures that have been proposed depending on the type of variable and the kind of association you are looking for.

(Refer Slide Time: 05:01)

Pearson's Correlation

- $n$  observations for  $x$  and  $y$  variables ( $x_i, y_i$ )
- Pearson's product-moment correlation coefficient ( $r_{xy}$ )

$$r_{xy} = \frac{\sum x_i y_i - n \bar{x} \bar{y}}{\sqrt{(\sum x_i^2 - n \bar{x}^2)} \sqrt{(\sum y_i^2 - n \bar{y}^2)}} = \frac{S_{xy}}{\sqrt{S_{xx}} \sqrt{S_{yy}}}$$

- $r_{xy}$  takes a value between -1 (negative correlation) and 1 (positive correlation)
- $r_{xy} = 0$  means no correlation

So, let us look at the most common type of correlation, which is called the Pearson's correlation. Here as we started with, we have  $n$  observations for the variables  $x$  and  $y$  and we define the Pearson's correlation coefficient denoted by  $r_{xy}$  or sometimes denoted by  $\rho_{xy}$ . By this quantity defined, where we are essentially taking same thing that we did before the covariance between  $x$  and  $y$ , divided by the standard deviation of  $x$  and the standard deviation of  $y$ .

The numerator represents the covariance between  $x$  and  $y$  can also be computed in this manner; we can expand that definition we have for the covariance and we can find that it is nothing but the product of  $x_i y_i - n$  times the mean of  $x$  and the mean of  $y$ , which represents the covariance of  $x$  and  $y$ . And the denominator represents the standard deviation; we can look at this division by the denominator as what is called normalization.

So, this value is now bounded, we can show that  $r_{xy}$  will take any value between -1 and +1 -1, if it takes a value we say that the two variables are negatively correlated; if  $r_{xy}$  it takes a value close to 1, we say they are positively correlated. On the other hand if  $r_{xy}$  happens

to value close to 0, it indicates that  $x$  and  $y$  have no correlation between them. Now what how we can use this, let us see.

(Refer Slide Time: 06:30)

Pearson's Correlation (Cont.)

- A measure for the degree of linear dependence between  $x$  and  $y$
- Cannot be applied to ordinal variables
- Sample size: Moderate (20-30) for good estimate
- Robustness: Outliers can lead to misleading values

Data Analytics

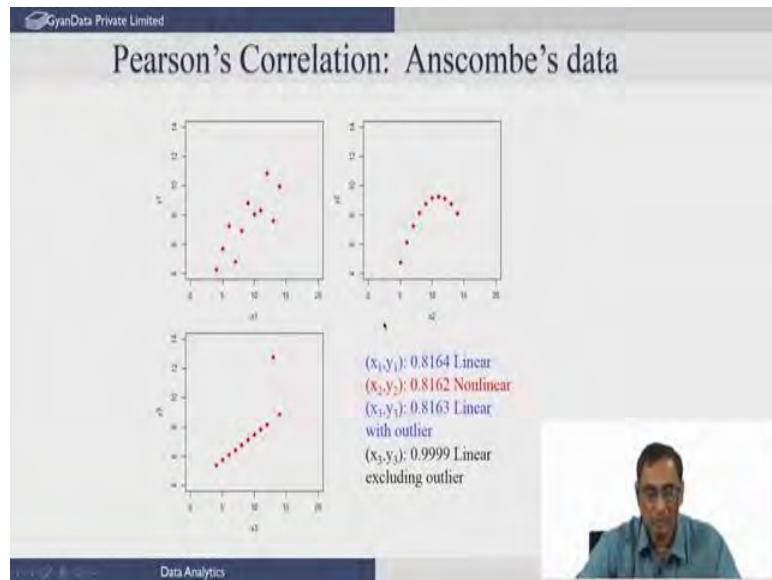
So, this correlation Pearson's correlation actually is useful to indicate there is a linear dependency between  $x$  and  $y$ ; for example, if  $y$  is a linear function of  $x$ , then we the correlation coefficient of correlation coefficient or the Pearson's correlation coefficient will turn out to be either close to +1 or close to -1, depending on whether  $y$  is increasing with  $x$ , then we say it is a positive correlation, as we saw in a figure. If  $y$  is decreasing with  $x$  linearly, then we say it is a the correlation coefficient will be close to -1.

On the other hand if the correlation coefficient is close to 0, all we can conclude from them is perhaps there is no linear relationship between  $y$  and  $x$ ; but perhaps there is non-linear association. So, we will come to that a little later. The way it is defined we can also not apply it to ordinal variables which means ranked variables. So, suppose you have a variable where you have indicated your scale, on a scale of say 0 to 10 ok; the let us say the course, the those kind of variables are typically you do not apply a Pearson's correlation, there are other kinds of correlation coefficients defined for what we call ranked or ordered variable, ordinal variables.

Typtically in order to get a good estimate of the correlation coefficient between  $y$  and  $x$  you need at least 20-30 points that is generally recommended. And then like your sample mean or the sample variance, standard deviation; if there are outliers, if there is one bad data

point or experimentally you know, experimental point which is wrongly recorded for example, that can lead to misleading values of this correlation coefficient. So, it is not robust with respect to outliers, just like the sample mean and sample variance we saw earlier.

(Refer Slide Time: 08:23)



So, let us look at some examples, this is a very famous data set called the Anscombe's data set. Here there are 11 data points, for each of this there are 4 data sets I have only shown 3 of them. Each of them contains exactly 11 data points corresponding to  $x_i$  and  $y_i$ , these points have been carefully selected. In the first one if you look at it, if you plot the scatter plot, you will see that there seems to be a linear relationship between  $y$  and  $x$  in the first data.

In the second data if you look at this figure, you can conclude that there is a non-linear relationship between  $x$  and  $y$ . And the third one you can say well, there is a perfect linear relationship for all the data points except one which seems to be an outlier, which is indicated way far away from that line. So, if I apply Pearson's, compute Pearson's correlation coefficient for each of these data sets, we find that it is identical; it does not matter whether you actually apply the first data set, or second data set, the third data set. In fact, the fourth data set there is no relationship between  $x$  and  $y$  and it turns out to be they have the same correlation coefficient.

So, what it seems to indicate, is that if we apply the Pearson's correlation and we find a high correlation coefficient close to 1 in this case; it does not immediately you cannot conclude there is a linear relationship. For example, this is a non-linear relationship and still gives rise to a high value. So, it is not confirmatory in the sense there is a linear, we can say it is one way; if there is a linear relationship between x and y, then the correlation Pearson's correlation coefficient will be high, when I say high it can be -1 or +1.

But if there is a non-linear relationship between x and y, it may be high or it may be low; we will see some data sets to actually show this, illustrate this point.

(Refer Slide Time: 10:14)

The screenshot shows a presentation slide with the title "Pearson's Correlation (Cont.)". Below the title, there are two bullet points under the heading "Example: Nonlinear".

- Example: Nonlinear
  - $x = 125$  equally spaced values between  $[0, 2\pi]$
  - $y = \cos(x)$
  - $r_{xy} = -0.0536$
- Example: Nonlinear
  - $x = 0:0.5:20; y = x^2; r_{xy} = 0.967$
  - $x = -10:0.5:10; y = x^2; r_{xy} = 0.0$

In the bottom right corner of the slide, there is a small video player window showing a man speaking. The video player has a play button and a progress bar. The overall background of the slide is light blue.

Here are three examples, in the first example I have taken 125 equally spaced values between 0 and  $2\pi$  for x; and I have actually computed y as  $\cos(x)$ . So, there is a relationship between y and x, in this case it is a sinusoidal relationship. So, if we apply the Pearson's correlation coefficient, compute the Pearson correlation coefficient for this data set we get a very low value close to 0, indicating as if there is no association between x and y.

But clearly there is a relationship because it is non-linear; in fact, it is symmetric, the points above the 0 line, when it is excess between 0 and  $\pi$  by 2 and when it is between  $\pi$  by 2 and  $\pi$  and between  $\pi$  and  $3\pi$  by 2,  $3\pi$  by 2 and  $2\pi$  they all seem to cancel each other out. And finally, give you a correlation coefficient which is very small does not indicate imply that there is no relationship between y, all you can conclude from this, is perhaps there is no linear relationship between x and y. Similarly let us look at another case, where this non-

linear  $y = x^2$  where I have chosen  $x$  between 0 and 20 with equally spaced point of 0.5 each 40 points you get; and I compute a  $y$  equals  $x$  square and then computer Pearson's correlation for this  $xy$  data, you find it is a very high correlation coefficient.

You can not immediately conclude there is a linear relationship between  $x$  and  $y$  ok, you can only say there is a relationship; perhaps it is linear, maybe it is even long linear we have to explore further. It is close to 0, I would have said there is no linear relationship ok; but it is in this case it is very high. So there is some association, but perhaps the association you cannot definitely conclude it is linear, it may be non-linear.

On the other hand if the data, if I chosen my  $x$  data between -10 and 10 symmetrically ok; for between -10 and 0,  $y = x^2$  will have positive values; between 0 and 10  $y$  equals  $x$  square will have positive values again, although  $x$  is positive,  $y$  is positive in this range and between negative values for  $x$ ,  $y$  is still positive. So, these will cancel each other out exactly and will turn out that the correlation coefficient in this case is 0, although there is a non-linear relationship between  $y$  and  $x$ .

So, all we are saying is this, if there exists a linear relationship between  $y$  and  $x$ ; then the Pearson's correlation coefficient will be either close to 1 or -1, perfect relationship, linear relationship. On the other hand if it is close to 0, you cannot dismiss a relationship between  $y$  and  $x$ . Similarly if value is high, looking at this just the value we cannot conclude that there definitely exists a linear relationship between  $y$  and  $x$ , you can only say there exists a relationship between  $y$  and  $x$ .

So, let us actually look at other correlation coefficients, you should note that Pearson's correlation coefficient can be applied only to what we called not ranked variables, ordinal variables, real valued variables like we have here.

(Refer Slide Time: 13:15)

GyanData Private Limited

## Spearman Rank Correlation

- Degree of association between two variables
- Linear or nonlinear association
- $x$  increases,  $y$  increases or decreases monotonically

Lecture Score

Score

Data Analytics

So, let us look at other correlation coefficients that can be applied even to ordinal variables. Now, here is a case, where we only look at the again look for degree of association between two variables; but this time the relationship may be either linear or non-linear, if  $x$  increases  $y$  increases or decreases monotonically, then the Spearman's rank correlation will tend to be very high.

So, here is a case when  $x$  increases  $y$  increases, this also is a case when  $x$  increases  $y$  increases monotonically; but in this case the right hand figure is a non-linear relationship, the left hand figure is indicates a linear relationship. Let us apply define Spearman's rank correlation, apply it to the same data set and see what happens.

(Refer Slide Time: 13:53)

Spearman Rank Correlation

- Spearman rank correlation computation for n observations:  
$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$
- $d_i$  is the difference in the ranks given to the two variables values for each item of the data
- Example:

| Number              | 1   | 2 | 3    | 4 | 5  | 6    | 7  | 8 | 9 | 10 |
|---------------------|-----|---|------|---|----|------|----|---|---|----|
| X <sub>i</sub>      | 7   | 6 | 4    | 5 | 8  | 7    | 10 | 3 | 9 | 2  |
| Y <sub>i</sub>      | 5   | 4 | 5    | 6 | 10 | 7    | 9  | 2 | 8 | 1  |
| Rank X <sub>i</sub> | 6.5 | 5 | 3    | 4 | 8  | 6.5  | 10 | 2 | 9 | 1  |
| Rank Y <sub>i</sub> | 4.5 | 3 | 4.5  | 6 | 10 | 7    | 9  | 2 | 8 | 1  |
| $d^2$               | 4   | 4 | 2.25 | 4 | 4  | 0.25 | 1  | 0 | 1 | 0  |

$r_s = 0.88$

So, in the Spearman's rank correlation what we do is convert the data, even if it is real value data to what we call ranks. So, for example, let us say I have 10 data points, in this case  $x_i$  is like somebody has taken a scale between let us say 2 and 10, 1 and 10 and similarly  $y$  is also a value between 1 and 10. So, what we have done is looked at all the individual values of  $x$  and assign the rank to it.

For example, the lowest value, in this case  $x$  value is 2 and it is given a rank 1, the next highest  $x$  value is 3 that is given a rank 2 and so on and so forth. So, we have ranked all of these points; notice that the 6th and the 1st value both are tied. So, they get the rank 6 and 7 which is the midway the half of it. So, we have given it a rank of 6.5, because there is a tie. Similarly if there are more than two values which are tied, we take all these ranks and average them by the number of data points which have equal values and correspondingly we obtain the rank.

We also rank the corresponding  $y$  values. For example in this case the 10th value as a rank 1 and so on so forth, 8th value as a rank 2 and so on; so we are given a rank in the similar manner. Now once you have got the rank, you compute the difference in the ranks. So, in this case the difference in the rank for the first data point is 2 and we square it; similarly we take the difference in the second data point in the ranks between  $x_i$  and  $y_i$  which is 2 and square it and we will get 4.

So, like this we take the difference in the ranks square it and we get the final, what we call the d squared values, we sum over all values and then we compute this coefficient. It turns out, that this coefficient also will be lie between -1 and +1; and -1 indicating a negative association, and +1 indicating a positive association between the variables. And in this particular case the rank, the Spearman rank correlation turns out to be 0.88.

(Refer Slide Time: 16:00)

GyanData Private Limited

## Spearman Rank Correlation

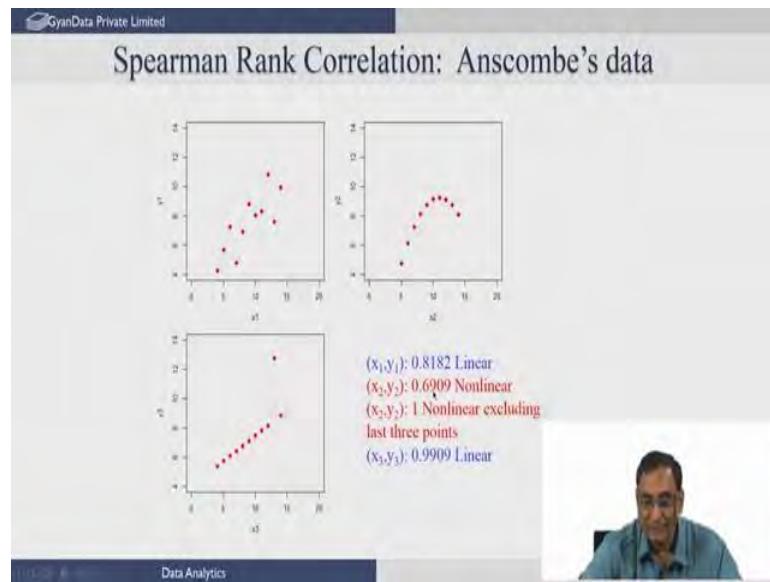
- $r_s$  takes a value between -1 (negative association) and 1 (positive association)
- $r_s = 0$  means no association
- Monotonically increasing  $r_s = 1$
- Monotonically decreasing  $r_s = -1$
- Can be used when association is nonlinear
- Can be applied for ordinal variables

Data Analytics

Let us look at some of the things, as I said 0 means no association when there is the positive association between y and x, then the  $r_s$  value a Spearman's thing will be +1 like the Pearson's correlation. And similarly when y decreases with x then we say that you know the Spearman's rank correlation is likely to be close to -1 and so on.

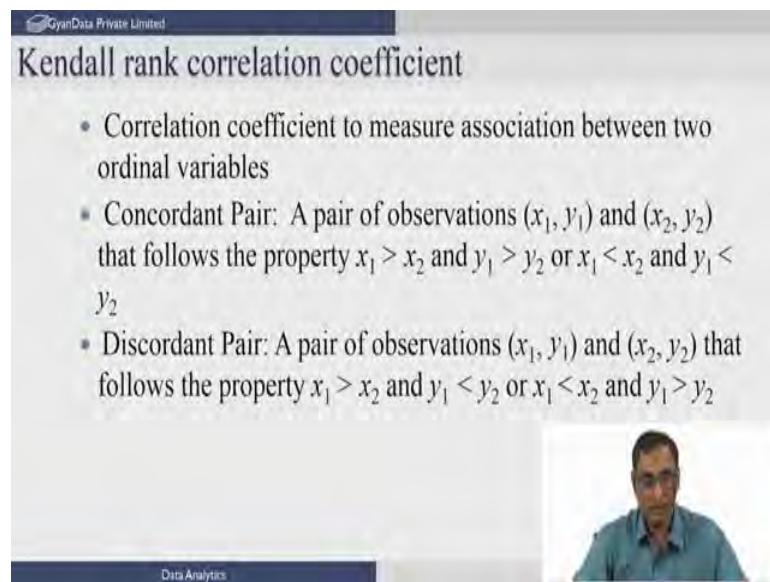
The differences between Pearson's and Spearman is not only can it be applied to ordinal variables; even if there is a non-linear relationship between y and x, the Spearman rank correlation can be high ok, it is not likely to be 0, it will have a reasonably high value. So, that can be used to distinguish maybe, to look for the kind of relationship between y and x.

(Refer Slide Time: 16:49)



So, let us apply it to the Anscombe's data set. In this case also we find that the, for the first one the Spearman rank correlation is quite high, in the second one also reasonably high. In fact, the Pearson also was high; notice that the Pearson was same for all this and the third one also is fairly high, 0.99. So, all of these things it is indicating that there is a really strong association between x and y.

(Refer Slide Time: 17:16)



Suppose we had applied I would suggest that you apply it to the cos x example and y squared x = y squared example; you will find that the Spearman rank correlation for these

will be reasonably high. It may not be close to 1, but will be high indicating there is some kind of a non-linear relationship between them, even though Pearson's correlation might be low. So, a third type of correlation coefficient that is used for ordinal variables is called the Kendall's rank correlation and this correlation coefficient also measures the association between ordinal variables.

In this case, what we define is a concordant and a discordant pair. If you look at the values compare two observations; let us say  $x_1$   $y_1$  and sorry here, it should be  $x_1$   $y_1$  and  $x_2$   $y_2$ . If  $x_1$  is greater than  $x_2$  and the corresponding  $y_1$  is greater than  $y_2$ , then we say it is a concordant pair; that means, if  $x$  increases and  $y$  also correspondingly increases then these two data points are known said to be concordant. Similarly if  $x$  is increasing and  $x_1$  less than  $x_2$ , and correspondingly  $y$ ,  $y_1$  is less than  $y_2$  then also we say it is a concordant pair; that means, when  $x$  increases  $y$  increases or  $x$  decreases or  $y$  decreases then we say these two data pairs are concordant.

On the other hand if there is an opposite kind of relationship. So, if you take two data points  $x_1$   $y_1$  and  $x_2$   $y_2$  and we say that you know look, we look at the data points and find that if  $x_1$  is greater than  $x_2$ ; but the corresponding  $y_1$  is less than  $y_2$  or if  $x_1$  is less than  $x_2$ , but  $y_1$  is greater than  $y_2$ , then we say it is a discordant pair. So, we take every pair of observations in your sample and then assign whether there is a concordant or discordant pair.

(Refer Slide Time: 19:09)

GyanData Private Limited

## Kendall rank correlation coefficient

- Kendall rank correlation coefficient

$$\tau = \frac{\text{Number of concordant pairs} - \text{Number of discordant pairs}}{n(n-1)/2}$$

- The pair for which  $x_1=x_2$  and  $y_1=y_2$  are not classified as concordant or discordant and are ignored.

Data Analytics

Let us take an example and look at it. So, once we have the number of concordant pairs and number of discordant pairs, we take the difference between them divide by the n into n -1 by 2 and that is called the Kendall stop.

(Refer Slide Time: 19:20)

GyanData Private Limited

### Kendall rank correlation coefficient

Example: Two experts ranking on food items

| Items | Expert 1 | Expert 2 |   |               |  |  |  |
|-------|----------|----------|---|---------------|--|--|--|
| 1     | 1        | 1        | 2 | C             |  |  |  |
| 2     | 2        | 3        | 3 | C C           |  |  |  |
| 3     | 3        | 6        | 4 | C D D         |  |  |  |
| 4     | 4        | 2        | 5 | C C C C       |  |  |  |
| 5     | 5        | 7        | 6 | C C D C D     |  |  |  |
| 6     | 6        | 4        | 7 | C C D C D C   |  |  |  |
| 7     | 7        | 5        |   | 1 2 3 4 5 6 7 |  |  |  |

$$\tau = \frac{15 - 6}{21} = 0.42857$$

Data Analytics

We can take an item here, there are about seven observations; let us say seven different wines or tea or coffee and there are two experts who rank the taste of the tea or coffee or wine on a scale between 1 to 10. For the first, the expert number 1 gives it a rank of 1 and expert 2 also ranks it 1; for the second 1 the expert 1 ranks it 2; while the expert 2 ranks it 3 in a scale or gives it the value of 3 and so on so forth for the seven different types of things. Now you compare data point 1 and data point 2, in this case experts opinion is that 2 is better than 1; expert 2 also says 2 is better than 1.

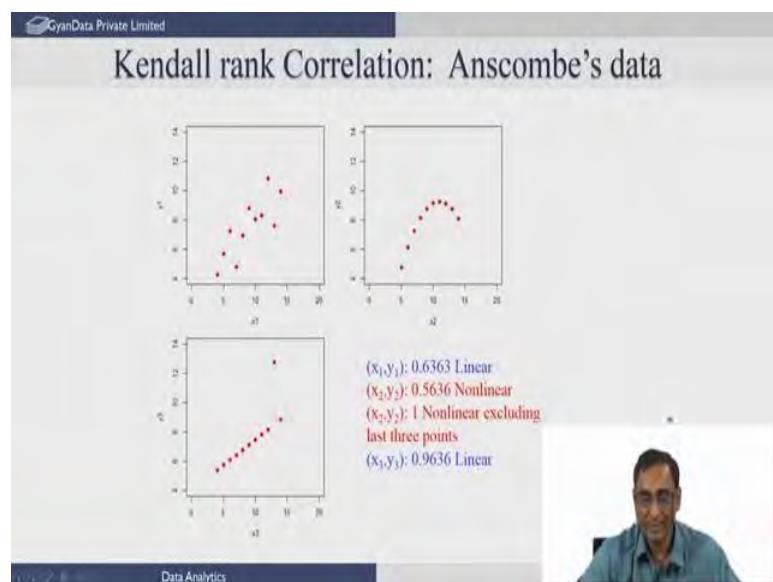
So, it is a concordant pair. So, 1 and 2 are concordant; that is what is indicated here. Similarly if I look at the data point 1 and 3, expert 1 says it is better, 3 is better than 1; expert 2 also says 3 is better than 1. So, it is a concordant pair. Similarly if you look at 2 and 3 both are agree in agreement, 3 is better than 2, 3 is better than 2. So, it is a concordant. Let us look at the 4th and the 1st one, looks like expert 1 says it is better, expert 2 also says better, concordant.

But the 2nd and 4th if you compare, expert 1 says it is better 4th one is better than the 2nd; but expert 2 disagrees, he says the 4th thing is worse than the 2nd one. So, there is a discordant pair of data and that is indicated by D. So, 4 and 2 are discordant, 4 and 3 are

discordant. Similarly here it says 5 and 1 are concordant; 5, 2 are concordant and so on so forth. So, between every pair n into n -1 by 2 pairs, you will get and we have classified all of these pairs as either concordant or discordant. And we find there are 6 discordant pairs and 15 concordant pairs and we can compute the Kendall's tau coefficient ok. This basically says, If this is high then there is broad agreement between the two experts, right.

So, basically we are saying y and x are associated with each other or there is a strong association; otherwise it is not strongly associated or completely, if the expert 2 completely disagrees with expert 1 you might get even negative values ok. So, the high negative value or high positive value indicates that the two variables x and y in this case are associated with each other. Again this can be used for ordinal variables, because it can work with ranked values here, as we have seen in this.

(Refer Slide Time: 21:53)



So, again if we apply it to Kendall's rank to this Anscombe's dataset, we find that although it has decreased for this linear case; the value has decreased as compared to the Pearson and Spearman's correlation coefficient; it still has a reasonably high value. High in this case typically you should, in experimental data you cannot expect to get a value I mean beyond 0.6 0.7, you should consider yourself fortunate typically; because we rarely know the nature of the relationship between variables, we are only trying to model them.

So, in this case non-linear relationship, you find again a reasonably high correlation coefficient for Kendall's rank and the last one again it is linear, perfectly linear, so you are

getting very high association between them. So, really speaking you can actually use these to get a preliminary idea before you even build the model; of course, for two variables it is easy, you can plot it, you can compute these correlation coefficient and try to get a preliminary assessment regarding the type of association that is likely to be; and then try go ahead and choose the type of models you want to bid.

And this is the first thing that we do before we jump into linear regression. So, see you next class about how to build the linear regression model.

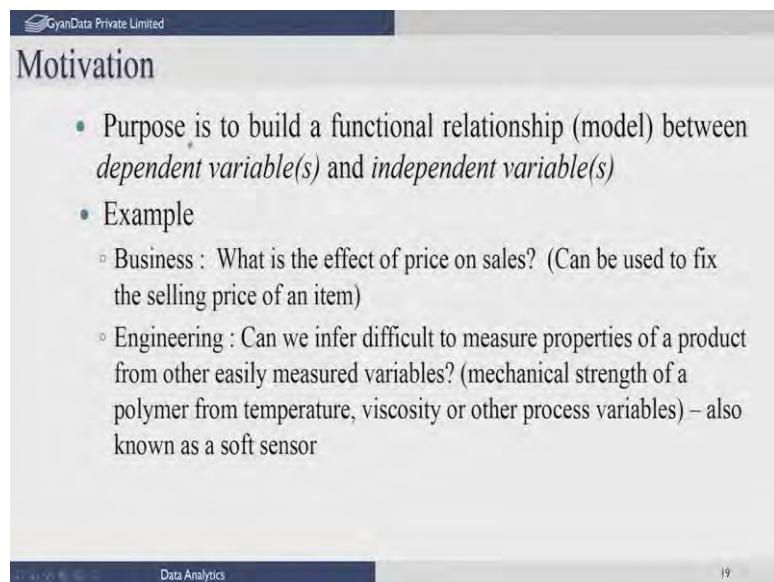
Thank you.

**Python for Data Science**  
**Prof. Raghunathan Rengasamy**  
**Department of Computer Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 30**  
**Linear Regression**

Welcome to this lecture on regression techniques. Today we are going to introduce to you the method of Linear Regression which is very popular technique for analyzing data and building models. We will start with some motivating examples. What is it that regression does?

(Refer Slide Time: 00:38)



Motivation

- Purpose is to build a functional relationship (model) between *dependent variable(s)* and *independent variable(s)*
- Example
  - Business : What is the effect of price on sales? (Can be used to fix the selling price of an item)
  - Engineering : Can we infer difficult to measure properties of a product from other easily measured variables? (mechanical strength of a polymer from temperature, viscosity or other process variables) – also known as a soft sensor

It is used to build a functional relationship or what we call model between a dependent variable or variables. There may be many of them and an independent variable. Again there might be more than one independent variable. We will define these variables and how you choose them for the intended purpose little later, but essentially we are building a relationship between two variables. You can take it in the simplest case and that relationship we also call it as a model.

So, in literature this is known as building a regression model. We can also call it as identification of a model. Sometimes this goes by the name of identification. Most popular term is regression. So, let us take some examples, let us take a business case. So, suppose we are interested in finding the effect of price on the sales volume.

Why do we want to actually find this effect? We may want to determine what kind of price we want to set the selling price of an item in order to either boost sales or get a better market share. So, that is why we are interested in finding what effect does price have on the sales. So, the purpose has to first define. What, why are we doing this in the first place?

In this case our ultimate aim is to fix the selling price. So, as to increase our market share. That is the reason we are trying to find this relationship. So, similarly let us take an engineering example. In this case I am looking at a problem where I am trying to measure or estimate the properties of a product which cannot be measured directly by means of an instrument easily.

However by measuring other variables we are trying to kind of infer or estimate this difficulty to measure property case in point is the mechanical strength of a polymer. This is very difficult to measure on line continuously or the other hand process conditions such as temperature viscosity of the medium can be measured and from this it is possible to infer provided we have a model that relates the mechanical strength to these variables temperature viscosity and so on. First you develop a model, then you can use the model to predict mechanical strength given temperature viscosity.

So, such a model is also known in the literature as a soft sensor or the software sensor and this model is very useful in practice to continuously infer values of variables which are difficult to measure using an instrument. Indirectly, you are always inferring it through this model and other variables. So, these are cases where we have the purpose is very clear. We are building the model for a given purpose and the purpose is defined depending on the area that you are working in.

(Refer Slide Time: 03:43)

The slide is titled "Regression - Basics" and is presented by GyanData Private Limited. It features a bulleted list defining regression terms:

- One of the widely used statistical techniques
- Dependent variables also known as *Response variable, Regressand, Predicted variable, output variable* - denoted as variable/s  $y$
- Independent variable also known as *Predictor variable, Regressor, Exploratory variable, input variable* - denoted as variable/s

A small video player window in the bottom right corner shows a person speaking. The slide has a "Data Analytics" footer.

So, regression happens to be one of the most widely used statistical techniques with data and typically there are two ah concepts here. The idea of a dependent variable which is known also in the literature as a response variable or a regressand or a predicted variable or simply the output variable. The variable whose output we desire to predict based on the model.

So, the symbolic way of denoting this output variable is by the symbol  $y$ . On the other hand we have what is called the independent variable. This is also known in the literature sometimes as the predictor variable the or the regressor variable as opposed to predicted and regressand or it is also known as the exploratory variable or very simply as the input variable. We will use the term independent variable for this and dependent variables for the response variable. We will not use the other terms in this talk.

So, the independent variable is denoted by the symbol  $x$  typically. So, we have let us for the simple case assume that we have only one variable which we denote by the variable  $x$ , the independent variable and we have another variable called the dependent variable which we wish to predict and we will denote it as  $y$ .

(Refer Slide Time: 05:04)

The slide is titled "Regression types" and is part of a course on "Data Analytics". It includes a navigation bar with icons for back, forward, search, and refresh, and a footer with the course name and slide number (21). The main content is a bulleted list under the heading "Classification of Regression Analysis":

- Classification of Regression Analysis
  - Univariate vs Multivariate
    - *Univariate*: One dependent and one independent variable
    - *Multivariate*: Multiple independent and multiple dependent variables
  - Linear vs Nonlinear
    - *Linear*: Relationship is linear between dependent and independent variables
    - *Nonlinear*: Relationship is nonlinear between dependent and independent variables
  - Simple vs Multiple
    - Simple: One dependent and one independent variable (SISO)
    - Multiple: One dependent and many independent variables (MISO)

There are several different classifications and we are I was just going to give you a brief idea of that we can have what is called the univariate regression problem or a multivariate regression problem. The univariate is the simplest regression problem you can come up cross which consists of only one dependent variable and one independent variable.

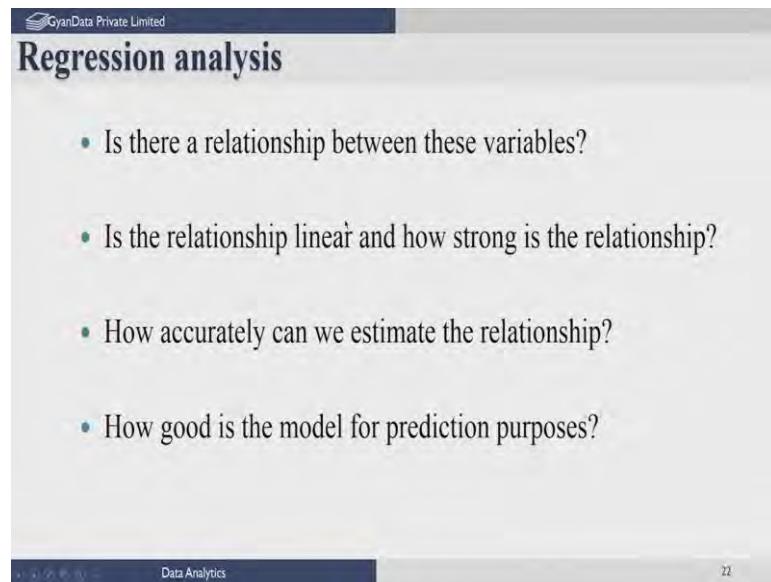
On the other hand if you talk about a multivariate regression problem, you have multiple independent variables and multiple dependent variables. So, to understand the subject it is better to take the simplest problem, understand it thoroughly and then you will see the extensions are fairly easy to follow. We can also have what is called linear versus non-linear regression.

Linear regression the relationship that we seek between the dependent and the independent variable is a linear function. Whereas, in a non-linear regression problem the functional relationship between the dependent and independent variable can be arbitrary, can be quadratic, can be sinusoidal or can be any arbitrary non-linear function.

And we will wish to discover that non-linear function that best describes this relationship that forms part of non-linear regression. We could also classify regression as simple versus multiple. Simple regression is the case of this single dependent and single depend independent variable also called the SISO system and multiple regression.

Linear regression is the case when we have one dependent variable and many independent variables or what is called the MISO case, Multiple Input Single Output. So, these are various ways of denoting the regression problem. We will always look at the simplest problem to start with which is the simple linear regression which consists of only one independent, one dependent variable and analyze it thoroughly.

(Refer Slide Time: 06:51)



GyanData Private Limited

## Regression analysis

- Is there a relationship between these variables?
- Is the relationship linear and how strong is the relationship?
- How accurately can we estimate the relationship?
- How good is the model for prediction purposes?

Data Analytics 22

So, the first thing that the various questions that we want to first ask where before we start the exercise is do we really think there is a relationship between these variables and if we believe there is a relationship, then we would not want to find out whether such a relationship is linear or not.

Of course in linear regression we are going ahead with the assumption there exists a linear relationship, but you really want to know whether such a relation, linear relationship exists and how strong is this, how strongly the independent variable affects the response of the dependent variable. Also we are interest since we are dealing with data that that has ran errors or stochastic in nature and we only have a small sample that we can gather from there from the particular application.

We want to ask this question what is the accuracy of our model in terms of how accurately we can estimate the relationship or the parameters in this model and if we use this model for prediction purposes subsequently how good it is? So, these are some of the questions that we would like to answer even in the process of developing the regression model.

(Refer Slide Time: 08:06)

GyanData Private Limited

## Regression methods

- Linear regression methods
  - Simple linear regression
  - Multiple linear regression
  - Ridge regression
  - Principal component regression
  - Lasso
  - Partial least squares
- Nonlinear regression methods
  - Polynomial regression
  - Spline regression
  - Neural networks

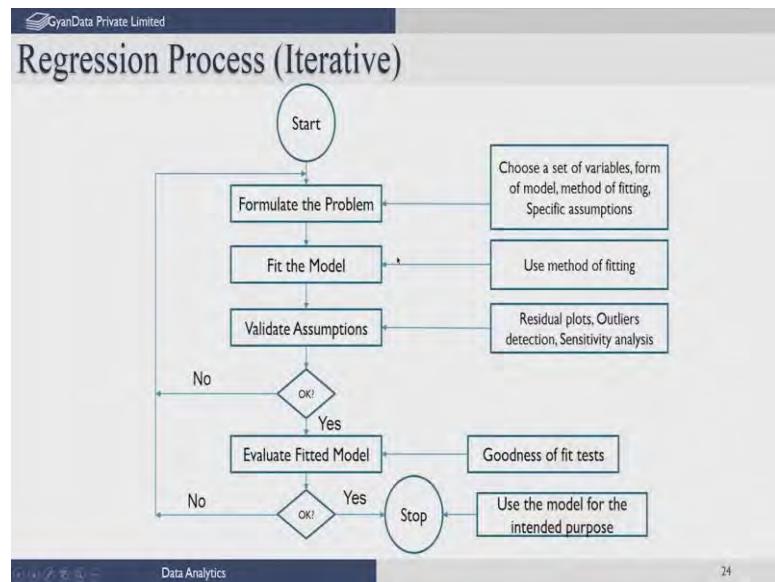
Data Analytics

So, there are several methods also that are available in the literature for performing the regression depending on the kind of assumptions you make and the kind of problems that may you may encounter. As I said the simple linear regression is the very very basic technique which we will discuss thoroughly.

Multiple linear regression is an extension of that for multiple independent variables, but there are other kinds of problems you may encounter when you have several variables, independent variables and those have to be tackled differently and there are techniques such as Ridge regression or Principal Component regression, Lasso regression, Partial Least Squares and so on and so forth which deals with these kind of difficulties that you might encounter in multi linear regression. Of course, in non-linear regression there is again a plethora of methods. I have only listed a just a few examples.

You could have polynomial or spline regression where the type of equations or functional relationship you specify a priori. You can have neural networks or today a support vector regression. These are methods that are used to develop non-linear models between the dependent and independent variable. Now, let us take only the simple linear regression and go further.

(Refer Slide Time: 09:15)



So, you have to understand that the regression process. It itself is not a once through process; it is iterative in nature. So, the first question that you should ask is the purpose. Before we even start the regression, you ask what is the purpose what are you trying to develop the model for like I said in the business case we are developing the model in order to determine set the price selling price of a thing.

So, you are really interested in how this selling price affects sales. That is the purpose that you have actually got in the case of the engineering case. We said the purpose is to replace a difficult to measure variable by other easily measured variables and this model using a combination of the model and other easily measured variables. We are predicting a variable which is difficult to measure online and then obviously we can monitor the process using that that parameter.

So, the purpose for each thing has to be well defined, then that leads you to the selection of variables which is the output variable that you want to predict and what are the input variables that you think are going to affect the output variable. And so you choose the set of variables and take measurements, get a sample, do design of experiments which is not talked about in this whole what we called lecture.

So, we will do proper design of experiments in to get the what we call meaningful data and once you have the data, we have to decide the type of model. When we say type of model, it is a linear model or non-linear model. So, let us say we have chosen one type of

model, then you have to actually choose the type of method that you are going to use in order to derive the parameters of the model or identify the model as we call it.

Once you have actually done that unfortunately when we use a method, it comes with a bunch of assumptions associated. You would like to validate or verify whether the assumptions we are made in deriving this model are correct or perhaps they are wrong what this is done by using what we call residual analysis or residual plots. So, we will examine the residual plots to kind of judge whether the assumptions we have made in developing the model are acceptable or not.

Sometimes you might have also a few data, experimental data. That data may be very bad, but you do not know this a priori you like to throw them out. They might affect the quality of your model and therefore, you would like to get rid of these bad data points and only use those good experimental observations for building the model.

How to identify such outliers or bad data is also part of the regression. You remove them and then you actually have to redo this exercise finally once develop the model. You want to actually do sensitivity analysis is there a if we have a small error in the data how well how much it affects the response variable and so on. So, this is sensitive analysis you do or if there are many variables you like to ask this question are all variables equally important or should I discard one of the input variables and so on.

So, these are the things that you would do and once you have built the best model that you can from the given data and the set of variables you have chosen, then you proceed further. So, the data that you used in building this model or regression model is also called the training data. You have used the model to train you to use the data to train the model or estimate the parameters of the model such that the data set is also denoted as the training data set.

Now, once you have built the model, you would like to see how well does it generalize, can it predict the output variable or the dependent variable for other values of the independent variable which you have not seen before. So, that comes to the testing phase of the model. So, you are evaluating the fitted model using data which is called test data. This test data is different from the training data.

So, when you do experimental observations if you have a lot of data, you set apart a sum for training and remaining for testing typically 70 or 80 percent of the data experimental data is used for training or fitting the parameters and the remaining 20 are used to test the model. This is typically done if you have a large number of data points. If you have fewer number of observations, then there are other techniques. We will actually explain or how to evaluate fitted models with small samples that you have.

So, you first evaluate find out how well the model predicts on data that it has not seen before and once you have satisfied with it, then you can stop otherwise if this model that you have developed even the best model that you have developed under whatever assumptions linear, linear model and so on and so forth that you have assumed it is not adequate for your purpose. You go ahead and change the model type. You may want to actually now consider a non-linear model maybe introduce a quadratic term or you might want to more look at a more general form and redo this entire thing, ok.

It may also turn out that whatever you do you are not getting a good model, then maybe you should once look at the set of variables you have chosen and also the type of experiments that you have conducted. So, there could be problems with those that is probably affecting the model development phase. So when you all your atoms are failed, you may want to even look at your experimental data that you have gathered. What, how did the, how did you conduct the experiments, whether there was any problem with that or the variables when you selected, did you miss out some important ones.

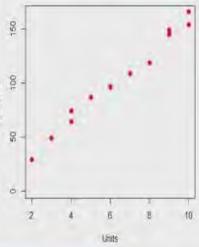
So, there was quite a lot to regression. What we are going to describe only a small part, we are not giving you the entire story. We are only providing a short story on how to formulate or fit a model for a linear case. How to validate assumptions and how to evaluate the fitted model, this is basically going to be the focus of the lectures.

(Refer Slide Time: 15:12)

 GyanData Private Limited

## Ordinary Least Squares (OLS)

- Fourteen observations obtained on time taken in minutes for service calls and number of units repaired
- Objective is to find relationship between these variables (useful for judging service agent performance)



Data Analytics 25

So, let us take one small example which we will use throughout. This is a data, fourteen observations small sample which we have taken on a servicing problem service agents. These service agents let us say its like Forbes Aquaguard service agent that comes to your house, they go visit several houses and they take a certain amount of time to kind of service the unit or repair it if it is down. So, they we will report the total amount of time taken in minutes let us say for through that they have spent on servicing different customers and the number of units that they have serviced in a given day.

So, let us assume that every day the service agent goes out on his rounds and notes the total amount of time he has actually spent and tells at the end of the day reports to his boss the number of units that he has repaired, he or she has repaired. Let us say that there are several such agents roaming around the city and so on and each of them come back and report let us say there are fourteen such data points that of the same person or multiple persons that you have actually gathered and from this the question that we want to actually answer let us say is given this data.

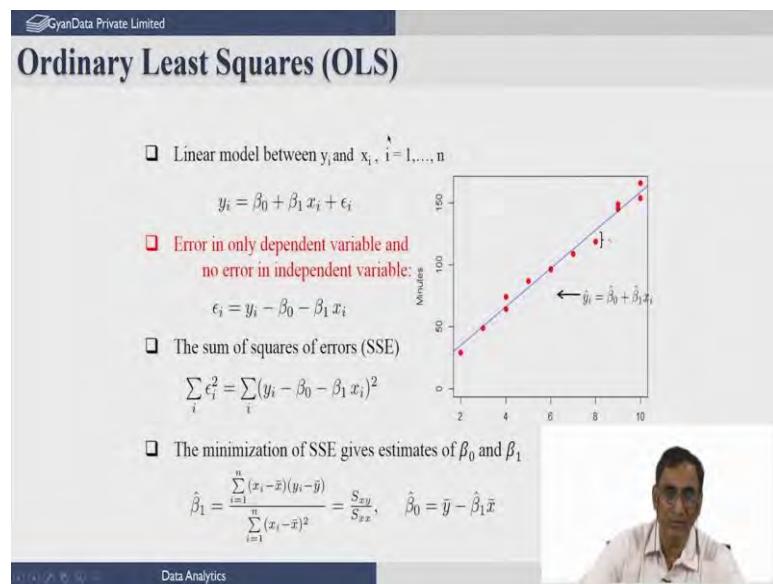
Suppose as an agent gives you data, you monitor him for week or month on how many how much time they spending and how many units he is repairing every day and want to judge the performance of that agent, service agent in order to reward or appropriately kind of you know improves productivity.

So, if you know a relationship between the time taken and number of units repaired which you believe should happen. If somebody takes more time and he is doing nothing not repairing much, then there is some inefficiency in the maybe is wasting too much time in between travel or whatever. So, we need to find out right. So, the purpose is to actually judge the service agent performance and do performance incentives in order to improve productivity of these agents. So, we are interested in developing a relationship between number of units and the time taken by thing or vice versa.

Now, for the sake of argument right now I have plotted as we said in the two variable cases you can visually plot the data scatter plot. So, you plot the data. First I have taken units on the x axis and the minutes on the y axis. I will discuss shortly whether we should choose units as the independent variable or minutes as the dependent variable or vice versa.

But for the time being I have just plotted it on the x and y axis and look at the spread of data and it looks like there is a linear relationship between the two variables, ok. Now, we want to build this linear relationship because from the trend of the data we believe a relationship exists. Let us go ahead with an assumption and just try to build this linear model ok.

(Refer Slide Time: 18:09)



Now, comes the exact mathematical form in which we state this problem. We have data points  $x_i$  of the independent variable, we have  $n$  data points in the example  $n$  is 14 and  $y$  is the dependent variable which we want to use for prediction or whatever purpose that we

want for this model. In this case as I said given both  $x$  and  $y$ , I would like to rate if some service agent comes and tells this is the time I spent and this is a number of units I have ai have repaired and based on its performance for a month or a week you would like to rate the service agent. That is the purpose for building this model. So, what we do is we have these fourteen observations we have taken and we are going to set up the model form.

So, as I said we are decided to go with a linear model and the linear model in general can be written like this between  $y$  and  $x$ , and this is said that every data point whatever be  $y$  whether it is time or units in the previous case which you have taken as the dependent variable can be written in terms of the independent variable as  $\beta_0$  a constant called the intercept term plus  $\beta_1 x_i$  where  $\beta_1$  represents what we call the slope.

So,  $\beta_1$  tells you how change in  $x_i$  effects the response variable  $y$ , ok. So,  $\beta_0$  is a constant. It is an offset term, but given  $x_i$  the independent variable it is not possible to get whatever observations we have. Observations always contain some error and that error is denoted by  $\epsilon_i$ . So,  $\epsilon_i$  represents an error.

Now, we have to ask this question. What is this error due to? There could be several reasons for this error. This could be because we have not measured  $x_i$  precisely or  $y_i$  precisely or it could be that the model form we have chosen is perhaps inadequate to explain the relationship between  $x_i$  and  $y_i$  and therefore, whatever we are unable to explain is denoted as  $\epsilon_i$  and it is called a Modeling error.

So, in this particular ordinary least squares regression that we are going to deal with, we will assume whenever we set up the problem  $x_i$  the independent variable is assumed to be completely error free in the sense we have measured  $x_i$  exactly. There is no error in reporting of  $x_i$ .

On the other hand the dependent variable  $y_i$  could contain error. We allow for errors in the reporting of  $y_i$ , but the error is not what we call a systematic error. It is a random error that is how we are modeling  $\epsilon_i$  or you could also look at  $\epsilon_i$  as a modeling error. In this particular case where you can say this linear model is only an approximation of the truth and anything that we are not able to explain perhaps can be treated like a random error modeling error.

Whatever be the reason the most important thing to note is that this particular model form or for ordinary least squares methodology, a formulation does not allow error in the

independent variable. So, when you choose the independent variable one of the things you should do carefully is that you should ensure that this thing is the most accurate among the two variables. If you have a two variable case, you should choose the independent variable as the one which is the most accurate one.

In fact, it should be probably error free. So, let us take the case of the units and minutes ok. Typically the number of units repaired by a service agent will be reported exactly because he will have a receipt from each customer and saying that the unit was serviced, you give this back and the total number of receipts that the service agent has gathered precisely represents the number of units serviced.

So, there cannot be an error unless somebody transcribes this thing the error in transcription. This can be exactly counted and you would have a precise idea. It is an integral number. They cannot be an error in this. On the other hand the amount of time taken could vary because of several reasons.

One because this guy reported the total time he actually started out on the day and when he returned to the office end of the day and this could involve not just the service time, but also travel time and depending on the location the travel time could vary, it could vary from time of day depending on the traffic, it could also vary because of congestion or a particular event that has happened. So, the time that has been reported contains other factors that we may not have precisely considered unless the service agent goes with that stopwatch and measures exactly the time for repair.

Typically you will report the total time spent in servicing all of these units including travel time and so on. That is the kind of data that you might get. So, you should regard the minutes as only an approximation. You cannot say that is only due to servicing, but also could have other factors which you treat as random disturbance or random error.

So, it is better in this case to choose units as the x variable because that is precise, that has no error and y minutes as the dependent variable notice. There might be an argument saying that you know you should always choose the variable which you wish to predict as the dependent variable need not once you build a model, you can always decide to use this model for predicting x given y or y given x.

So, it does not matter how you cast this equation, how you build this model, it is more important that when you apply ordinary least squares you should actually ensure that the independent variable is an extremely accurate measurement or it represents the truth as closely as possible whereas,  $y$  could contain other factors or errors and so on and it is this method is tolerant to it.

So, this goes if on the other hand if you believe both  $x$  and  $y$  contain significant error, then perhaps you should consider other methods called total least squares or principal component regression that we will talk about later ok. If not in this lecture, but if we have the time we will do it later.

So, essentially what I am saying is that once you have decided based on purpose based on the kind of quality of the measurements what is the independent dependent variable, then you can go ahead and say given all the observations  $n$  observations what is the best estimate of  $\beta_0$  and  $\beta_1$  as I read that  $\beta_0$  is the intercept parameter and  $\beta_1$  is the slope to actually geometrically interpret  $\beta_0$ .  $\beta_0$  represents the value of  $y$  when  $x$  is 0.

So, when you put  $x=0$  and you look at where this line intercepts the  $y$  axis, this vertical distance is  $\beta_0$  and the slope which represents the slope of this regression line that is  $\beta_1$ . So, you are estimating the intercept and slope. So, now what is the methodology for estimating this  $\beta_0$ , ok?  $\beta_0$  and  $\beta_1$ . So, what we will do is we will do a kind of a thought experiment. You give values of  $\beta_0$  and  $\beta_1$  and then you can draw this line.

So, we will ask different people let us say values of  $\beta_0$  and  $\beta_1$  and draw appropriate lines. The line shape that the slope and the intercept will be different depending on what value you proposed for  $\beta_0$  and  $\beta_1$ , then once you have done this you will actually go back and find out how much deviation is there between the observed value and the line ok.

In this particular case, we will say the observed value let us take this observed value is  $y_i$  corresponding to this  $x_i$  which is 8. Now, the line if this particular equation is correct, then this is the predicted value of  $y$  which means for this given value of  $x_i$  according to this equation you believe  $y$  predicted should be here and then this deviation between the observed value and the predicted value which is on this line.

The vertical distance is what we call the estimated error, ok. So, you do not know what the actual error is, but if you propose values for  $\beta_0$   $\beta_1$  immediately I am able to derive an

estimate for this error which is the vertical distance of the point from that line, we estimate this error for all data points, ok. So, we compute  $e_i$  for every data point  $y_i$  using the proposed parameters  $\beta_0$   $\beta_1$  and the value of the independent variables we have for all the observations. Now, what we do is we can say as a metric what is the best line we propose that the best line is one which minimizes the deviations sum squared deviations or the distances.

So, overall the data points we will compute this distance which is geometric distance is nothing, but square of this value we will compute this and sum over all the data points n data points and we try to find  $\beta_0$  and  $\beta_1$  which minimizes this sum squared value or minimizes the sum of the vertical distances or the point from that line.

So, the notion of a best fit line in the least square sense or the ordinary least square sense is one that minimizes the vertical distance of the points from the proposed line. Now, you can once you set up this formulation, then we can say then whoever gives the best  $\beta_0$  and  $\beta_1$  will have the minimum vertical distance of the points from that line and this can be done now analytically.

Instead of asking you now for this  $\beta_0$  and  $\beta_1$ , I try to solve this optimization problem which means minimize this, find out  $\beta_0$   $\beta_1$  which minimizes this and this what is called the unconstrained optimization problem with two parameters. You differentiate this with respect to  $\beta_0$ , set it equal to 0 for those called the first order conditions.

Those of you have done a little bit of optimization will know that or calculus will know all I have to do is differentiate this function with respect to  $\beta_0$ , set it equal to 0, differentiate this function with respect to  $\beta_1$ , set it equal to 0 and solve the resulting set of equations. And finally, I will get the solution for  $\beta_0$  and  $\beta_1$  which minimizes this sum square error. So, the least squares technique uses this as a criterion in order to derive the best values of  $\beta_0$  and  $\beta_1$ .

Of course we can counter by saying I will use some other metric, maybe I should have used absolute value that will make the problem difficult. This method was proposed in the late 1700s by Gauss, our and another person called Legendre and they it has become popular as a methodology although in recent years other methods have taken over.

So, the method of least squares is a very popular technique and it gives you parameters analytically for the simplest cases. So, you get  $\beta_0$  estimated. So, the estimate that you derive is not that you should treat this estimate as actually the truth, it is an estimate from data. Had you given me a different sample maybe I would have got a different estimate. Remember that the estimate is always a function of the sample that you had given me.

So, we denote such estimates by this hat always implies. It is an estimated quantity and the estimated value of  $\beta_1$  turns out to be the cross covariance between x and y divided by the variance of x. You can prove this. So, remember you this cross covariance is essentially like a Pearson's coefficient. So, the Pearson's coefficient said if the coefficient Pearson's correlation coefficient was close to 1 or -1, you said that there is a you could interpret that there may exist a linear relationship.

Similarly you can see  $\widehat{\beta}_0$  is function of that coefficient it depends on the cross covariance between x and y and  $\beta_0$  the intercept turns out to be nothing, but the mean of y-the estimated value of  $\beta_1$  slope parameter multiplied by the mean of x. This is your intercept parameter.

Of course one could also ask suppose I know that if x is 0, y is 0. I know that apriori in this particular case for example if you do not service any units which means you have not travelled you are not let us say you are on holiday, then clearly you would have taken zero time for servicing. So, I know in this particular case perhaps that that if you process zero units, you should not have taken any time. and so, therefore the intercept should pass through 0.

If you know it and you want you want to force this line to pass through 0 0 the origin, then you should not estimate  $\beta_0$ . You should simply remove this parameter and simply write  $y = \beta_1 x$  and in which case the solution for  $\beta_1$  will turn out to be again  $S_{xy}$  by  $S_{xx}$ . Except that this  $S_{xy}$  is a cross covariance not about the mean, but about 0 which means you set  $\bar{xy}$  equals 0 in this expression and you will get sigma x y in the numerator over all data points divided by sigma  $(x_i - \bar{x})^2$ .

So, essentially you are taking the variance around 0 and the cross covariance around 0,0 0 and then you will get the estimated value of  $\beta_1$ . Of course,  $\beta_0$  in that case is assumed to be 0. So, the line will pass through 0,0 and you will get another slope you are forcing the line

to pass through 0,0. Remember you have to be careful when you do this because it will unless you are sure that should pass through the origin, you should not force this thing. You will get a bad fit, ok.

If you know it and you want demand it makes physical sense, then you are well within your rights to ah force  $\beta_0$  equal to 0. Do not estimate it that can be done by simply taking the cross covariance and variance around 0 instead of around their respective means.

(Refer Slide Time: 32:26)

**OLS: Testing Goodness of Fit**

- ❑ Prediction using the regression equation:  $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$
- ❑ Coefficient of determination -  $R^2$  is a measure of variability in output variable explained by input variable
 
$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Variability explained by  $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$

Total variability in y
- ❑  $R^2$  values: Between 0 and 1
  - Values close to 0 indicates poor fit
  - Values close to 1 indicates a good fit (However, should not be used as sole criterion to judge that a linear model is adequate)
- ❑ Adjusted  $\bar{R}^2$ 

$$\bar{R}^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2 / (n-p-1)}{\sum(y_i - \bar{y})^2 / (n-1)}$$

Data Analytics

27

So, this is as far as getting the solution is concerned. So, now once you get the solution you can ask for every given  $x_i$  what is the corresponding predicted value of  $y_i$  using the model. So, you plug in your value of  $x_i$  in the estimated model which is using the estimated parameter  $\hat{\beta}_0$  and  $\hat{\beta}_1$  and I will call this prediction  $\hat{y}_1$ , it is also an estimated quantity.

For any given  $x_i$ , I can estimate the corresponding  $y_i$  using the model and geometrically if you actually try to estimate  $y_i$  given this point, it will fall on this line ok. You draw the vertical line which intersects this particular regression line and that particular point on that line will represent  $\hat{y}_i$  for every point. So, for this point it is actually the corresponding predicted value will lie on this line here. If this is the best fit line is the blue line represents the best fit line in the least square sense.

So, you can do this for any new point which you have not seen before in the test set. Also let us look at some couple of other measures which you can derive from this. We can talk

about what is called the coefficient of determination R squared which is defined in this manner,

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2 / (n - p - 1)}{\sum(y_i - \bar{y})^2 / (n - 1)}$$

. So, essentially this particular quantity called R squared will be between 0 and 1 we can show, ok.

So, how to interpret this? The denominator is the variability in  $y_i$ . What do you mean by that? That is if you have given just  $y_i$  and try to find out how much variance there it is there in the data, this particular thing divided by n of course gives you the variance of  $y$ . So, you can say this much variability exists in the data.

Suppose I build the model and try to predict  $y_i$ . If  $x_i$  had a influence on  $y$ , then I should be able to reduce its variability, I should be able to do a better prediction and the difference between  $y_i$  and  $\hat{y}_i$  should be lower if  $x_i$  had a strong influence in determining  $y$ , ok. So, the numerator represents the variability which is explained by the explanatory variable  $x_i$  or the independent variable  $x_i$ , ok.

So, if the numerator is approximately equal to the denominator, then you basically get 1 and R squared will be close to 0. The implication of this is  $x_i$  has a very little impact on explaining  $y$  and probably there is no relationship between  $y$  and  $x$ . On the other hand if the numerator is close to 0 and then you get r square close to 1, it implies that the  $x_i$  can explain the variation in  $y_i$  which means there is a strong relationship between  $x_i$  and  $y_i$ . So, values close to 0 indicates a poor fit, values close to 1 indicates a good fit, but the problem does not end there

If you get R squared close to 1, you should not conclude your job is done and the linear relationship is good and so on and the Anscombe data for example, when we saw last year last class if you try to find the Anscombe data for the four datasets, you will get all R squared close to 1 and that does not mean that the linear model is good. You should look at other measures before you conclude conclusively determine that a linear model is good.

Value close to 1 is a good starting point. Yes you can now be a little bit assurance you get that the linear model perhaps can explain relationship between  $y_i$  and  $x_i$  ok. There is also something called the adjusted R square which we will come across which is

essentially this. If you look at the denominator, you can say that if you try to estimate a constant value suppose you say  $x_i$  has no influence and drop that  $i$ ,  $\beta_i$  and try to estimate  $\beta_0$  in the least square sense, you will find that the best value of  $\beta_0$  is actually best estimate is just  $\bar{y}$ .

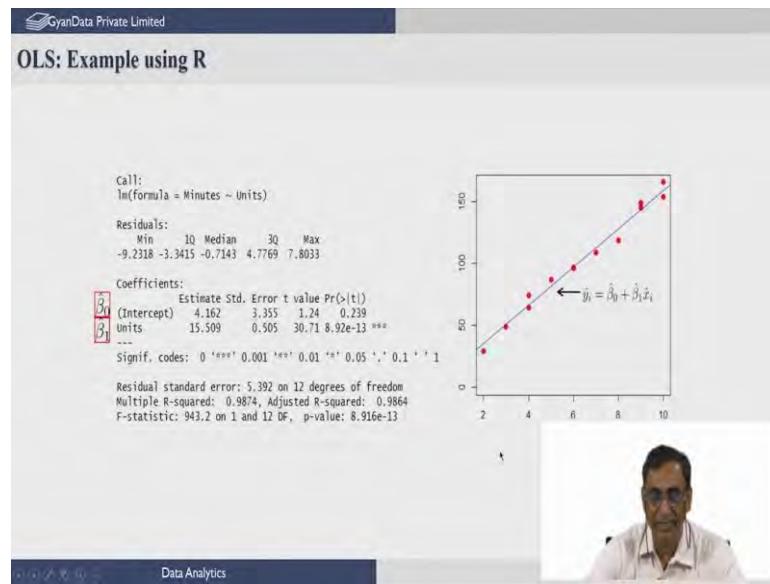
So, you can regard the denominator as fitting a model with just the parameter  $\beta_0$ . On the other hand the numerator you have used two parameters to fit the model. Whenever you use more parameters typically you should get a better fit. So, generally the numerator value is obtained because you have used two parameters whereas, the denominator you used only one parameter.

So, you have to account for the fact that you have used more parameters to obtain a better fit and not because there is a linear relationship between  $y_i$  and  $x_i$ . So, you should go back and account for this what we call the number of parameters you have used and or the number of degrees of freedom that is used in estimating the numerator. For example, you have  $n$  data points and in this case the  $p=2$  parameters.

So,  $n-1$  am sorry  $p$  equals 1 which happens to be only  $\beta_1$ . So,  $n-2$  would represent the number of degrees of freedom used to estimate this numerator variability whereas,  $n-1$  is used to estimate the denominator variability because you have used only the parameter  $\beta_0$  for denominator whereas, you used 2 parameters to estimate the numerator.

So, you should adjust this by dividing the number of degrees of freedom and the adjusted  $R^2$  essentially makes this adjustment and give the it is different from  $R^2$ , but it is a more accurate way of what we call judging whether there is a good linear good model between the dependent and the independent variable. And in this case  $p$  equals 1 because I have only one explanatory variable, but this it can extend into a many independent variable case where  $p$  is the number of independent variables you have chosen for fitting the model.

(Refer Slide Time: 38:32)



Ok finally we will end with the R command. The R command for fitting a linear model is just called lm if you have loaded the data set and then you say that what kind of, what you call ah variable is the independent variable and what is the dependent variable you indicate. In this case we have indicated minutes has the independent variable and units has the dependent variable and these are variables that forms part of the data set. They are defined the as these variables and therefore, you are using them.

So, loading of the data set you would have already seen, lm is the one that you used to build the model, you indicate what is the dependent and the independent variable and then you will get an output that is given here. First you will get the range of residuals which I said is the estimated value of  $\epsilon_i$  for all the data points.

In this case all the fourteen residuals are not given the max value min value. The first quartile, third quartile in the median are given here and I will only now look at two parameters; the  $\beta_0$  which is the first. The intercept is called the  $\beta_0$ . Estimated value is here and slope parameter the estimated value is 15.5 for this particular data set.

Now, I will also now only focus on this particular line which talks about the R squared value which we explain the to judge the quality of the model. It is a very high R squared you get or the adjusted R squared. So, from this we can conclude may be a linear model s explains their relationship between x and y very precisely, but we are not done yet. We have to do residual analysis, we have to do further ah what you call plots in order to judge

and conclude that a linear model is adequate. We will do this and the other things that outputs that are given as in the subsequent lectures I will explain them and we will see you in the next lecture.s

Thank you.

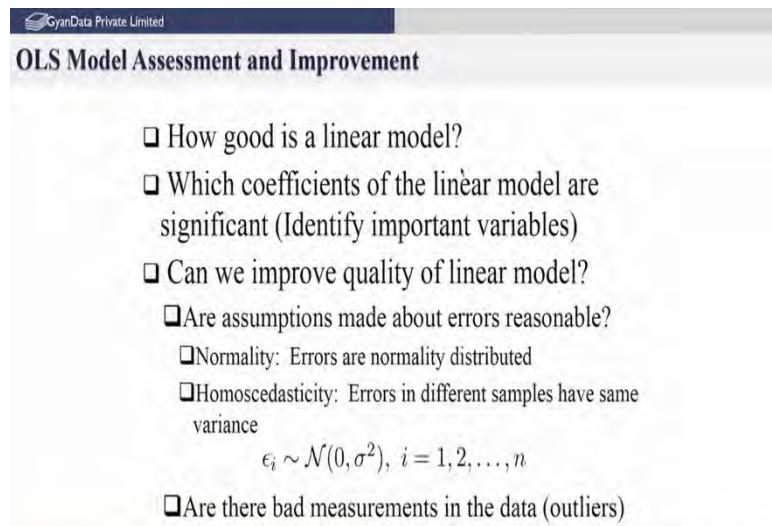
**Python for Data Science**  
**Prof. Raghunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 31**  
**Model Assessment**

Good morning everyone. In the previous lecture we saw how to fit a linear model between two variables  $x$  which is the independent variable and  $y$  which is the dependent variable using techniques called regression and in this particular lecture, we are going to assess whether the model we have actually fitted is reasonably good or not. There are many methods for making this assessment; we will look at some of these.

So, what are the useful questions to ask when we fit a model? The first question to ask is whether the linear model that we have fitted is adequate or not is good or not? If it is not good then perhaps we may have to go and fit a non-linear model. So, this is the first step that we will actually test whether the model is good or not.

(Refer Slide Time: 01:07)



The slide has a dark blue header with the GyanData Private Limited logo. The main title 'OLS Model Assessment and Improvement' is centered in a white box. Below the title is a bulleted list of questions:

- ❑ How good is a linear model?
- ❑ Which coefficients of the linear model are significant (Identify important variables)
- ❑ Can we improve quality of linear model?
  - ❑ Are assumptions made about errors reasonable?
    - ❑ Normality: Errors are normally distributed
    - ❑ Homoscedasticity: Errors in different samples have same variance
$$\epsilon_i \sim \mathcal{N}(0, \sigma^2), i = 1, 2, \dots, n$$
  - ❑ Are there bad measurements in the data (outliers)

Then even if you fit a model you may want to find out which coefficients of the linear model are relevant. For example, in the one variable case that we saw one independent variable the only two parameters that we are fitting are the intercept term  $\beta_0$  and the slope term  $\beta_1$ .

So, we want to know whether we should have fitted the intercept or not, whether we should have taken it as 0. When we have several independent variables in multi linear regression; we will see that it is also important to find out which variables are significant, whether we should use all the independent variables or whether we should discard some of them.

So, this particular test for finding which coefficients of the linear model are significant is useful not only in the uni varied case, but more useful in multi linear regression where we want to identify important variables. Suppose the linear model that we fit is acceptable, then we want to actually see whether we can improve the quality of the linear model. When fitting linear model using the method of least squares, we make several assumptions about the errors that corrupt the dependent variable measurements.

So, are these assumptions really valid? So, what are some of the assumptions that we make about the errors that corrupt the measurements of the dependent variable? We assume that the errors are normally distributed only this assumption can actually justify the choice of the method of least squares. We also assume that the errors in different samples have the same variance; now this is called the homoscedasticity assumption.

So, we are assuming that the errors in different samples are also having the same variance. In general the these two statements assumptions about the errors that they are normally distributed with identical variance can be compactly represented by saying that  $\epsilon_i$ ; the error corrupting measurement  $i$  is normally distributed with 0 mean and sigma square variance. Notice that sigma squared is same and does not depend on  $i$  which means it is the same for all samples  $i$  equals 1 to  $n$  that is the assumption we are made when we use the standard method of least squares.

Now, we also assume that all the measurements that we have made are reasonably good and there are no bad data points or what we call outliers in the data. We saw that even when we are estimating a sample mean one bad data can result in a very bad estimate of the mean.

So, similarly in the method of least squares; if we have one bad data point it can result in a very poor estimate of the coefficients. So, we want to remove such bad data from our data set and improve may be fit the linear model only using the remaining measurements and that will improve the quality of the linear model. So, these are some of the things we

need to actually verify. These assumptions what we have made about the errors, whether they are reasonable or not if there are bad data can we remove them or not.

And so, we will look at the first two questions in this lecture which is to assess whether the linear model that we have fitted is good and how do we decide whether the coefficients of the linear model are significant.

(Refer Slide Time: 04:35)

- ❑ Both  $\hat{\beta}_0$  and  $\hat{\beta}_1$  estimates are unbiased  

$$E[\hat{\beta}_0] = \beta_0, \quad E[\hat{\beta}_1] = \beta_1$$
- ❑ Variance of the estimates  

$$\text{var}[\hat{\beta}_1] = \frac{\sigma^2}{S_{xx}}, \quad \text{var}[\hat{\beta}_0] = \sigma^2 \frac{\sum x_i^2}{n S_{xx}}$$
- ❑ Estimate of  $\sigma^2$   

$$\hat{\sigma}^2 = \frac{\sum (y_i - \hat{y}_i)^2}{n-2} = \frac{SSE}{n-2}$$
- ❑ Distribution of slope estimate     $\hat{\beta}_1 \sim \mathcal{N}(\beta_1, \frac{\sigma^2}{S_{xx}})$

So, before we start we need to derive some properties of these estimates that we have derived. Remember that the coefficients of the linear model that we have fitted which is the intercept term  $\beta_0$  and the slope term  $\beta_1$ ; these are obtained from data from the sample of data that you have given.

We have indicated that these are estimates and not the true values by putting this karat term on top of each of these symbols; which means that this is an estimate  $\widehat{\beta}_0$  is an estimate of the true  $\beta_0$  and  $\widehat{\beta}_1$  is an estimate of the true  $\beta_1$  which we do not know. However, we can prove based on the assumptions we have made regarding the errors that the expected value of  $\widehat{\beta}_0$  will be  $=\beta_0$ . What does it mean? If we were to repeat these experiment collect another sample of n measurements and apply the method of least squares; we will get another estimate of  $\beta_0$ .

Suppose we do this experiment several times and we will get several estimates of  $\widehat{\beta}_0$ ; let me average all of them and the average value of that will tend towards the true value; that

is what this expression means, that if we were to repeat these experiment several times the average of the estimates that we derive will actually be a very good representation of the truth. Similarly, we can show that the expected value of  $\widehat{\beta}_1$ =the true value  $\beta_1$ ; notice that  $\beta_0$  and  $\beta_1$  are unknown values.

We are only saying that the expected value of  $\beta_1$  hat will be the true value and the expected value of  $\beta_0$  hat will be the true value and such statements are also known as if the estimates satisfies such properties, we also call these estimates are unbiased there is no bias in the estimate of  $\widehat{\beta}_0$  or  $\widehat{\beta}_1$ . The second important property that we need to derive about the estimates is the variability of the estimates.

Notice, we get different estimates of  $\beta_0$  hat depending on the sample that we have derived. And therefore, we want to ask what is the spread of these estimates of  $\widehat{\beta}_0$  and  $\widehat{\beta}_1$ ; if we were to repeat this experiment; we can show again through based on the assumptions we are made that the variance of  $\beta_1$  hat will be=sigma squared by  $S_{xx}$ ;  $S_{xx}$  represents the variance of x or  $(x - \bar{x})^2$  summed over all the samples; where as  $\sigma^2$  represents the variance of the error that corrupts the dependent variable y.

So, sigma squared is the error variance,  $S_{xx}$  is the variance of the independent variable. So, this ratio we can show will be=the variance of  $\widehat{\beta}_1$ . Similarly, we can show that the variance of  $\widehat{\beta}_0$  is  $\sigma^2$  which is the variance of the error, multiplied by this ratio the numerator is the sum squared values of all the independent variables; while the denominator represents the variance of the independent variable. In this the  $S_{xx}$  can be computed from data sigma, of  $x_i^2$  can be computed from data.

But we may or may not have knowledge about the variance of the error which corrupts the dependent variable; that depends on the instrument that was used to measure the dependent variable. If we have some knowledge of this instrument accuracy; we can take the sigma squared from that but in most cases data analysis cases we may not have been told what is the accuracy of the instrument used to measure the dependent variable.

So, sigma squared also have to somehow be estimated from the data; we can show that we can derive a very good estimate of sigma squared by this quantity that is described here which is nothing but the difference between the measured value  $y_i$  and the estimated value  $y_i$  hat which is obtained from the linear equation we have fitted the linear model.

So, for every  $x_i$  we can predict from the linear model what is the estimate of  $\hat{y}_i$  for every sample, then we can take the difference between the measured and their predicted value of the dependent variable; sum squared divided by  $n-2$  that is a good estimate of sigma hat squared which is the error in the dependent variable. Now, why do we divided by  $n-2$  instead of  $n-n$  or  $n-1$ ?

Very simple  $\hat{y}_i$  was estimated using the linear model it has two parameters  $\beta_0$  and  $\beta_1$  which means the two of the data points have been used to estimate  $\beta_0$  and  $\beta_1$ . And therefore, only the remaining  $n-2$  samples are available for estimating this sigma square ok. Suppose, you had only two samples then your numerator would be exactly 0 because you have more than two samples you have variability and that variability is caused by the error in the dependent variable; that is one of the reasons that you are dividing by  $n-2$  because two data points have been used to estimate the parameters  $\beta_0$  and  $\beta_1$ .

Now, this particular numerator term is also called the sum squared errors or SSE for short and so  $\widehat{\sigma^2} = \text{SSE } / (n-2)$ . So, from the data after we have fitted the model we can compute this value and compute this SSE and obtain an estimate for sigma hat square. So, you do not need to be told the information about the; accuracy of the instrument used to measure the dependent variable, you can get it from the data itself ok.

So, now finally not only we have got the first moment properties of  $\widehat{\beta_0}$ ,  $\widehat{\beta_1}$  as well as the second moment properties which is variance of  $\widehat{\beta_1}$  and the variance of  $\widehat{\beta_0}$ ; we can also derive the distribution of the parameter in particular  $\widehat{\beta_1}$  can be shown to be normally distributed. Of course, with because the expected value of  $\widehat{\beta_1}$  is  $\beta_1$ ; it is normally distributed with  $\beta_1$  the true unknown value of  $\beta_1$  as the mean and the variance given by sigma; if you substitute the sigma hat squared here, you can finally, show that this is nothing, but I am sorry.

So, this is the unknown  $\sigma^2$  divided by  $S_{xx}$  ok; sigma squared is essentially here we have derived this  $\sigma^2$  by  $S_{xx}$  is the variance of  $\beta_1$  hat. Now, if you do not know sigma squared you can replace the sigma squared with this sigma hat squared SSE by  $n-2$  ok.

So, once you have derived the distribution of the parameters; we can perform hypothesis testing on the parameters to decide whether these are significantly different from 0 and

that is what we are going to do. We can also derive what we call confidence intervals for these estimates based on their distribution characteristics that is the mean and the variance.

(Refer Slide Time: 11:49)

GyanData Private Limited  
OLS: Confidence Intervals on regression coefficients

□ 95% two-sided confidence intervals (CI) for  $\hat{\beta}_0$  and  $\hat{\beta}_1$

$$\hat{\beta}_1 \in [\hat{\beta}_1 - 2.18 s_{\hat{\beta}_1}, \hat{\beta}_1 + 2.18 s_{\hat{\beta}_1}], \quad s_{\hat{\beta}_1} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{(n-2)S_{xx}}} \quad t_{0.025, 12}$$

$$\hat{\beta}_0 \in [\hat{\beta}_0 - 2.18 s_{\hat{\beta}_0}, \hat{\beta}_0 + 2.18 s_{\hat{\beta}_0}], \quad s_{\hat{\beta}_0} = s_e \sqrt{\frac{\sum x_i^2}{n S_{xx}}} \\ s_e = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{(n-2)}}$$

Data Analytics 32

Now, the first thing we will do is to develop confidence intervals; confidence interval simply says what is the interval within which the true value unknown value is likely to be with 95 percent confidence or 90 percent confidence.

You can decide what size confidence interval size you need to have and correspondingly you can obtain the interval from the distribution. So, if you want 95 percent confidence interval also known as CI and its two sided because it could be either to the left of this estimated value or to the right of the estimated value.

So, we are obtaining the 95 percent confidence interval for  $\beta_1$  from its distribution knowing its normally distributed with some unknown variance. So, that we can actually derive from the from this particular range which is the estimated value of  $\beta_1$ ; which is  $\widehat{\beta}_1 \pm 2.18$  times, the standard deviation of  $\widehat{\beta}_1$  estimated from the data.

Notice, this is very similar to the normal thing which says that the true value will lie between estimate  $\pm 2$  times the standard deviation. The reason why we have 2.18 instead of 2 is because we are no longer obtaining the critical value from the normal distribution, but from the t distribution because  $\sigma^2$  is estimated from the data not known apriori.

So, the distribution slightly changes it is not the normal distribution, but the t distribution and that is what we have pointed out here this 2.18 is nothing but the critical value 2.5 percent critical value upper critical value with 12 degrees of freedom. Why 12 degrees of freedom? Because you have in this particular example, we had 14 points and we used two of the points for estimating the two parameters.

So,  $n-2$  is the degrees of freedom; which represents 12 in general depending on the number of data points this value 2.18 will change ok. So, that changes the degrees of freedom of the t distribution from which you should pick the upper and lower critical value. So, lower critical value is -2.18; the upper critical value is 2.18; 2.5 percent. So, the overall is 5 percent this confidence interval represents the 95 percent confidence interval for  $\beta_1$ .

So, all we are going to state is that the  $\beta_1$  true unknown  $\beta_1$  lies within this interval with 95 percent confidence; that is what we are saying ok.  $\widehat{\beta}_1$  can be estimated from data so you can construct this confidence interval. Similarly, you can construct the 95 percent confidence interval for  $\beta_0$  from its variance.

So, we are doing the same thing  $\widehat{\beta}_0 \pm 2.18$  times; standard deviation of  $\widehat{\beta}_0$  estimated from data which is what we call  $s_{\widehat{\beta}_0}$  remember

$$s_{\widehat{\beta}_0} = \sigma^2 \sqrt{\frac{\sum x_i^2}{n S_{xx}}}$$

which is nothing, but the square root of what we have derived in the earlier thing with sigma squared replaced by the estimated quantity; that is all this these two terms represents  $s_{\widehat{\beta}_0}$  and  $s_{\widehat{\beta}_1}$ .

So, having constructed this 95 percent confidence interval; you can also use it for testing whether  $\beta_0$  is the unknown  $\beta_0=0$  or not or the unknown  $\beta_1$  is 0 or not which is what we will do.

(Refer Slide Time: 15:29)

GyanData Private Limited

### OLS: Hypotheses test on regression coefficients

- ❑ In order to check if linear model fit is good or not we can test whether estimate  $\hat{\beta}_1$  is significant (different from zero) or not
- ❑ Null hypothesis  $H_0 : \beta_1 = 0$
- ❑ Alternative hypothesis  $H_1 : \beta_1 \neq 0$
- ❑ Null hypothesis implies  $\hat{y}_i = \hat{\beta}_0 + \epsilon_i$  ← Reduced Model
- ❑ Alternative hypothesis implies  $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \epsilon_i$  ← Full Model
- ❑ Do not Reject null hypothesis if CI for  $\beta_1$  includes 0
- ❑ Similarly if CI for  $\hat{\beta}_0$  includes 0, then intercept term is insignificant

So, let us look at why would we want to actually do this hypothesis tests. We have fitted a linear model assuming that you know that there is a linear dependency between x and y and we have obtained an estimate of  $\hat{\beta}_1$ .

Also we have also fitted an intercept term we may want to ask is the intercept term significant maybe the line should pass through 0,0 the origin. Maybe the y variable does not depend on  $x_1$  in a significant manner which means  $\hat{\beta}_1$  is approximately=0 that unknown  $\beta_1$  is exactly=0. Although we have got some estimate for  $\hat{\beta}_1$  non 0 estimate for  $\hat{\beta}_1$ .

So, the null hypothesis what we want to test is  $\beta_1=0$  versus the alternative that  $\beta_1\neq 0$ . If  $\beta_1=0$  it implies that the independent variable x has no effect on the dependent variable, but on the other if we reject this null hypothesis; we are concluding that the independent variable does have some effect on the dependent variable ok.

So, this particular hypothesis test can be also re interpreted as the null hypothesis implies  $\beta_1=0$ ; which means what we are doing is only a fit of  $y_i=a$  constant, whereas if we accept the or reject the null hypothesis; then we are actually fitting a linear model with  $\beta_0$  and  $\beta_1$  present ok. So, the null hypothesis represents the fit of a reduced model which involves only the constant whereas, the rejection of the null hypothesis or the alternative hypothesis implies that we believe there is a linear model that relates y to x.

So, between these two models we want to pick whether the reduced model is acceptable or maybe the full model is to be accepted and the reduced model should be rejected; that is what we are doing when we test this hypothesis  $\beta_1=0$  versus  $\beta_1 \neq 0$ . Remember  $\beta_1$  can be either positive or negative and that is why we are doing a two sided test.

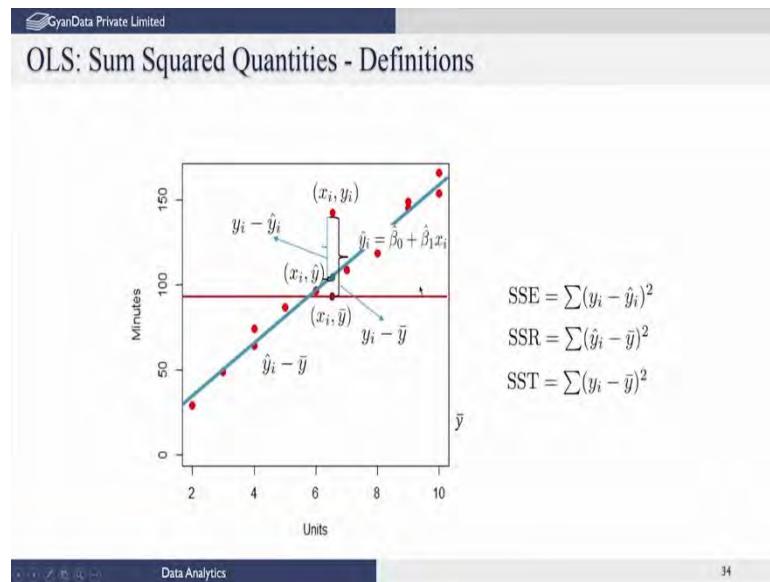
So, we can do it two ways we can actually reject the null hypothesis if the confidence interval for  $\beta_1$  include 0. So, notice that we have constructed the confidence interval for  $\beta_1$ . So, this term  $\widehat{\beta}_1 - 2.18$  may be negative and this maybe positive in which case this interval include 0 and then we have to definitely; we might make a decision that that  $\beta_1$  is insignificant and actually the true  $\beta_1=0$ .

On the other hand, if both these quantities if the interval is to the left of 0 which is completely negative or to the right of 0 which means both these quantities are positive; then this interval will not contain 0 and then we make the conclusion reject the null hypothesis that  $\beta_1$  equal 0 which means  $\beta_1$  is significant.

So, from the confidence interval itself; it is possible make the reject or not reject the null hypothesis. So, we can extend this kind of analysis to even test whether  $\beta_0$  is 0 or 0. So, if the confidence interval for  $\beta_0$ ; this particular interval includes 0, then we say that the intercept term is insignificant otherwise we will say that the intercept term should be is significant and should be retained in the model ok.

So, let us actually when we do a final example we; we will see this. There are other ways of performing this test and we will continue the we will do that also because that is very useful when we come to multi linear regression. In the uni variant regression we have only these two parameters but multi linear regression there are several parameters we will have one corresponding to each independent variable and therefore, there will be lot more hypothesis test you will do therefore, we will extend this kind of an argument to test for  $\beta_1=0$  or  $\beta_1 \neq 0$  using what is called a F test which we will go through.

(Refer Slide Time: 19:39)



So, before performing this F test to check whether a reduced model is adequate or we should accept a full model; we will use some definitions for some squared quantities. Notice that let us say that we have the set of data; in this case, we have the example of the number of units that were repaired and the time taken in minutes to repair the units by different sales person and we had 14 such data points 14 such salesmen, who have actually reported the data.

So, the red points actually represents the data and the best the linear fit using the method of least squares using all the data points; we got something that is indicated by the blue line. Now, suppose we believe that a constant model is good then we would actually fitted this particular horizontal line would be the best fit representing  $\bar{y}$ . The best estimate of the constant model is the mean of  $y$  for all values of  $x$ ; our prediction best prediction for  $y_i$  is the mean value of  $y_i$  which means  $x$  has no relevance;  $\beta_1$  is 0, so we will estimate the best constant fit for  $y_i$  is this mean value ok.

So, the red line represents the best fit when we ignore  $\beta_1$  the slope the blue line represents the best fit of the data when we include the slope parameter  $\beta_1$ . Now let us look at certain sum squared deviation the deviation between  $y_i$  and  $\bar{y}$  which is the red line; best fit of the constant this distance is  $y_i - \bar{y}$  and sum squared of all these vertical distances from the point to the red horizontal line; constant line that is what we call the SS total or sum squared

total which also represents the variance of  $y$ ;  $(y_i - \bar{y})^2$  all that we have not done is divided by  $n$ .

If we had divided by  $n$  or  $n-1$  we would have got the variance of  $y$ , but this represents the sum squared errors in  $y_i$ ; when we ignore the slope parameter that is another way of looking at it. The distance between  $y_i$  and  $\hat{y}_i$ ; so now suppose we assume that the slope parameter is relevant, then we would have fitted this blue line and for every  $x_i$  let us take this  $x_i$ ;  $y_i$  corresponding to this independent variable, the predicted value of  $y_i$  using this linear model would be the intersection point of this vertical line with the blue line which is represented by the blue dot which is what we call  $\hat{y}_i$ .

And therefore, this vertical distance between the measured and the predicted value is the sum squared errors is called  $SSE = (y_i - \hat{y}_i)^2$ . And this is the total error if we include the slope parameter in the fit ok. So, the difference between these two quantities SS Total-SS Error will be equal to what is also called the sum squared residuals which is nothing, but the predicted value-the mean value  $\bar{y}$ ; sum squared over all the data points.

Now, we can show that SST will always be greater than SSE because SSE was obtained by fitting two parameters. Therefore, you should be able to reduce the error may be marginally, but you will be always able to be able to reduce the error. So, SS total is the we will always be greater than SSE and therefore, this difference SSR will also be positive; all of these are positive quantities.

Now, one can interpret SS total as the goodness of fit if we assume a constant model; we can interpret SSE as the goodness of fit of the linear model. And therefore, we can now use this to perform a test, literally intuitively we can say that if the reduction by including the slope parameter that is SST-SSE is significant; then we conclude it is worthwhile including this extra parameter otherwise not. This can be converted into hypothesis test formal hypothesis test and that is what is called the F test.

(Refer Slide Time: 24:03)

GyanData Private Limited

### OLS: F-Test for choosing between models

- ❑ F-test for rejecting reduced model
- ❑ SST is goodness of fit for reduced model (null hypothesis)
- ❑ SSE is goodness of fit for full model (alternative hypothesis)
- ❑ F-statistic  $F_o = \frac{SST - SSE}{SSE/(n-2)} = \frac{SSR}{SSE/(n-2)}$
- ❑ At 5% level of significance reject null hypothesis if  $F_o \geq F_{(1,n-2;0.05)}$  (upper critical value of F distribution with 1 and n-2 dfs)
- ❑ Note that the numerator has 1 df

Data Analytics

35

So, what we are doing is as I said that SS total is a measure of how good the reduced model is which is reduced model here implies a constant model whereas, SSE represents how good the linear model; if we include the slope parameter.

So, we are asking whether the reduced model should be accepted which is the null hypothesis or should be rejected in favor of this alternative which is to include the slope parameter. So, as I said the F statistic for doing this hypothesis test is to compute the difference in the goodness of fit for the reduced model which is always higher-the goodness of fit SSE for the alternative hypothesis.

So, this represents the sum squared errors for the reduced sort of model fit; SSE represents the goodness of fit for the alternative hypothesis fit. This difference if it is large enough as I said, then we can actually say may be it is worthwhile going with the alternative hypothesis rather than the null hypothesis. So, SSR which is the difference between this should be large enough.

So, normalization what; what the denominator represents in some sense a percentage SSE is the error obtained for the alternative hypothesis. Remember because of the difference in the number of parameters used in the model; we have to take that into account. The numerator SST has n-1 degrees of freedom because we are fitting only one parameter. This has n-2 degrees of freedom because we are fitting two parameters; so the difference actually means its only one extra parameter.

So, there is numerator which is SSR has only 1 degree of freedom which is  $n-1-(n-2)$  whereas, the denominator SSE has  $n-2$  degrees of freedom because it has two parameters which is fitted. So, we are dividing the SSE by  $n-2$ ; the number of degrees of freedom. So, average sum squared errors per degree of freedom that is what we are saying and that is your normalization SSR divided by this normalizes the quantity.

And we can show formally that it is an F statistic because it is a ratio of two squared quantities and each squared quantity is itself a chi squared variable because it is a square of a normal variable. Therefore, this is the ratio of 2 chi squared and we have seen in the hypothesis testing the ratio of 2 chi squared variable is an F distribution with appropriate degrees of freedom the numerator degrees of freedom is 1, the denominator degrees of freedom is  $n-2$ .

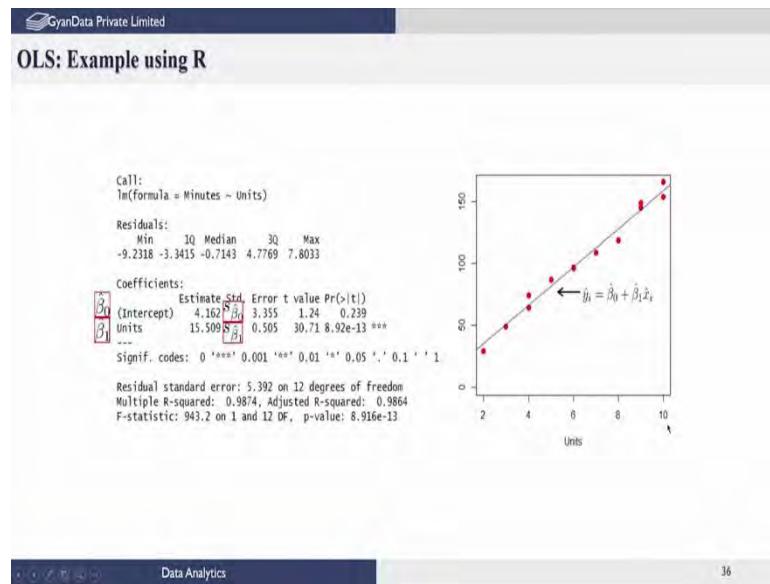
So, if we want to now do a hypothesis test using this statistic  $F_0$  we compare  $F_0$  with the critical value from the F distribution. Notice  $F$  is actually a positive quantity, so we do one sided test; if we choose the level of significance as 5; 5 percent; then we choose the upper critical value from the F distribution with 1 and  $n-2$  degrees of freedom and 5 percent level of significance or what we call the upper critical values probability is 5 percent 0.05.

So, once we get this from F distribution; we got this threshold and if the statistic exceeds and threshold, then we will reject the null hypothesis and say the full model is better than the reduced mode. We will accept the full mode or we say the; we reject the reduced model in favor of the full model that. The slope parameter is worth including in the model we will get a better fit that is how we actually conclude.

So, now there are several ways for deciding whether the linear model we have fitted is good or not. We could have used the r squared value we said that if it is close to plus one then we should that is one indicator that the linear model may be good it is not sufficient what I call sufficient to conclude but it is good indicator. We can also do the test for  $\beta$  significance of  $\beta_1$ ; if we conclude that  $\beta_1$  is not significant then maybe, then a linear model is not good enough we have to find something else or we can do an F test and conclude whether the including the slope parameter is significant.

So, these are various ways by which we can decide that the linear model is acceptable or not or the fit is good. We cannot stop at this we have to do further tests, but at least these are good initial indicators that we are on the right track.

(Refer Slide Time: 28:39)



So, let us apply this to the example of repair of or the servicing problem, where we have 14 data points and the time taken and the number of units repaired by different salesmen are given.

So, in this case we have these 14 points which we have showed, we have fitted the data using r remember that lm is the function which we should call for fitting a linear model and here we are predicting the dependent variable is minutes and the independent variable is units and once we have fitted this using the R function; it gives out all of this output and it gives you the coefficient, the intercept term turns out to be 4.162; the slope parameter turns out to be 15.501 ok.

But also it also tells you what is the standard deviation; estimated standard deviation of this parameter which is  $s_{\hat{\beta}_0}$  of the intercept. We also tells you what is the standard deviation of this estimate for  $\hat{\beta}_1$  which turns out to be 0.505 all of this calculated from the data using the formulas we have described. Now, once it has given out we can actually now perhaps construct confidence intervals and find out whether these are significant or not or itself actually tells you something whether these if you run a hypothesis test; whether you can we will conclude whether  $\hat{\beta}_0$  is significant or  $\hat{\beta}_1$  is significant and that is indicated by what is called this P value that it has reported.

So, if you get a very high value t value is represents the statistic which you have again described earlier. So, it has computed the statistic for you for  $\hat{\beta}_0$  and the statistic for

testing whether  $\beta_1=0$  or not and it has computed this statistic value and it has compared with the critical value the distribution t distribution with the appropriate degrees of freedom and concluded that the upper critical or the probabilities 0.239; which means if you get very high value for this anything greater than 0.01 or 0.05; it means you should reject the we should not reject the null hypothesis. On the other hand, if you get a very low value it means you should reject the null hypothesis with greater confidence you can reject the null hypothesis.

So, in this case all its saying is if you choose a level of significance 0.001; you would not reject the null hypothesis, if you choose 0.05; you will not reject the null hypothesis, if you choose 0.01 as your level of significance, you will not reject the null hypothesis. So, that is what the star indicates at what level of significance will you reject it. Whereas, in the case of  $\beta_1$ ; you will reject the null hypothesis which means you will conclude that  $\beta_1$  is significant even if you choose very low significance value 0.05, 0.01, 0.001 or even lower value. In fact, upto  $10^{-13}$  you will end up rejecting the null hypothesis. Very low type one error probability if you choose also, you will reject the null hypothesis.

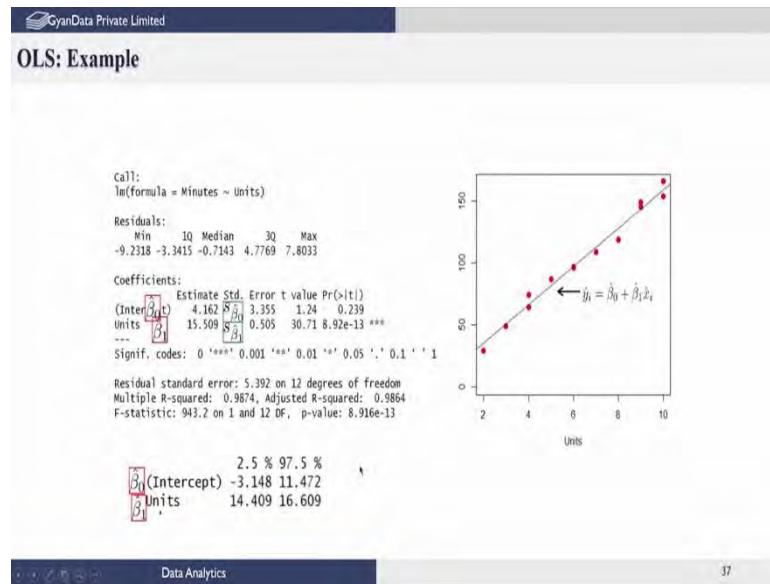
So, therefore you can concluding from these values that  $\beta_0$  hat is insignificant which means  $\widehat{\beta}_0 = 0$  is a reasonable hypothesis,  $\widehat{\beta}_1 \neq 0$  is a reasonable hypothesis. Let us go and see whether this makes sense for this data. We know that if there no units are repaired then clearly no time should be taken by the sales repair person; which means because you have not taken any time for servicing because he has not repaired any units.

So, this line technically should pass through 0,0 and that is what he has said, but; however, we went ahead merrily and fitted an intercept term but the test for hypothesis says you can safely assume  $\beta_0$  the intercept is 0; it makes physical sense also and we could have only fitted  $\beta_1$  that is good enough for this data ok.

So, perhaps you should redo this linear fit with  $\beta_0$  0 and only using  $\beta_1$  and the; you will get a slight different solution and you can test again. So, another way of deciding whether the significant whether the slope parameter is significant or not is to look at the F statistic. Notice the F statistic is very high and this p value is very low which means you will reject the null hypothesis that the reduced model is adequate; implying that you should use  $\beta_1$ , including  $\beta_1$  is very good you will get a better fit using  $\beta_1$  in your modeling.

So, the high value of test statistic indicates that you reject the null hypothesis or a low value of p value for this F statistic indicates that you reject the null hypothesis even at a very low significance level.

(Refer Slide Time: 33:41)



You can also construct the confidence interval for  $\beta_0$  and  $\beta_1$  and from the earlier thing you say approximately it is estimate  $\pm 2.18$  times the standard error and that is what is seen  $4.1 \pm 2.18$  times  $3.35$  and that turns out to give that gives the interval confidence interval  $-3.148$  to  $11.472$ ; that means, with 95 percent confidence, we can claim that the true  $\beta_0$  lies in this interval.

Similarly, we can construct the interval confidence interval for  $\widehat{\beta}_1$  95 percent confidence interval and it turns out it is  $15 \pm$  approximately two times  $0.5$  which is  $14$  and  $16.6$ . Now, clearly the interval confidence interval for  $\beta_0$  includes  $0$  and therefore, we should not reject the null hypothesis  $\beta_0=0$  ok.

We should simply accept that  $\beta_0$  perhaps  $=0$ , whereas the interval for confidence interval for  $\beta_1$  does not include  $0$ ; so, we can reject the null hypothesis that  $\beta_1=0$  and the slope is an important parameter to retain in the model. Now, all this we have done only for single thing; we will be extending it to the multi linear case and we will also look at other assumptions; the influence of bad data and so on in the following lectures.

So, see you in the next lecture.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

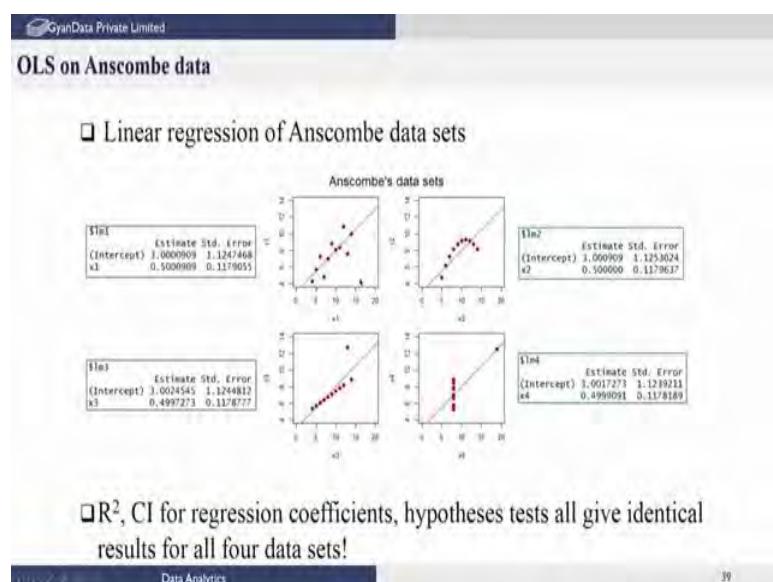
**Lecture – 32**  
**Diagnostics to Improve Linear Model Fit**

Good morning. In the previous lecture, we saw different measures that we can use to assess whether the linear model that we have fitted is good or not. For example, we can use the R squared value and if we find that the R squared value is close to plus one we can maybe accept the fact that the linear model is good.

We can also check based on the F statistic whether the reduced model is better than the model with the slope parameter. So, if we reject the null hypothesis there again we may conclude that the linear model is acceptable. We can also do this by testing the significance of the slope parameter we can look at the confidence interval for the slope parameter and if that does not include 0, then maybe we can accept a linear model.

But, all of these measures are not sufficient they only provide an initial indicator I will show you some data set which shows that that these measures are not completely sufficient to accept a linear model. We will use other diagnostic measure to conclusively accept or reject a linear model fit. So, let us look at a data set provided by Anscombe.

(Refer Slide Time: 01:38)



We have seen this before in the when we analyzed statistical measures in one of the lectures. The Anscombe data set is consists of four data sets each one of them having 11 data points,  $x$  versus  $y$ . They are synthetically constructed to illustrate the point. For example, these four data sets are plotted  $x$  versus  $y$ , the scatter plot is given; first data set here, second, third and fourth.

And, in all of these if you actually look at it look at the scatter plot we may say that look a linear model is adequate for the first data set and perhaps for the third data set, but the second data set indicates that the linear model may not be a good choice, a long linear model or a quadratic model may be a better fit. The last dataset is a very poorly designed data set, you can see that the experiment is conducted only at two distinct values of  $x$ ; you have one value of  $x$  here for which you have 10 experiments conducted you have got 10 different  $y$  values for the same  $x$ . And, then you have one more experimental observation at a different value of  $x$ .

So, you should in this case you should not attempt to fit a linear model with the data, instead you should ask the experimenter to go and collect data at different values of  $x$  then come back and try to check whether that is valid. Unfortunately, when we actually apply linear regression to these data sets and then find the slope and the intercept parameter we find that in all four cases we get the same intercept value of 3 you can see that all four data sets you get a value of 3 and you also get the same slope parameter which is 0.5 in all four cases.

So, the regression model if you fit to any of these data four data sets you will get the same estimate of the intercept and slope. Furthermore, you get the same standard error of it which is 1.12 for intercept and point one for the slope and if you run a confidence interval for the slope parameter, you may end up accepting that this slope is acceptable for all four cases and you may conclude incorrectly conclude that the linear model is adequate.

You can actually run the  $R^2$  squared value, it will be the same for all four data sets; you can run the a hypothesis test whether a reduced model is acceptable compared to a model with the slope parameter again you will reject a null hypothesis using the  $F$  statistic and you may conclude for all four cases you get the same identical result that a linear model is a good fit. Clearly, it is not so. One can of course, do scatter plots and try to judge it in this particular case because it is a univariate example, but when you have many independent

variables then you have to examine several such scatter plots and that may not be very easy.

So, if you assume there are 100 independent variables you have to examine 100 such plots of  $y$  versus  $x$  and it may not be possible for you to make a visual conclusion from that. So, we will use other kinds of plots called residual plots which will enable us to do this whether it is a univariate regression problem or a multivariate regression problem. We will see what these are.

(Refer Slide Time: 04:57)

GyanData Private Limited

### OLS: Residual Analysis

□ Questions:

- Do the underlying data satisfy the assumptions on errors (normality, same variance)?
- Is data free of outliers?
- Do some observations exert more influence than others?
- Can the regression equation be improved by using a nonlinear model?

Anscombe's data sets

So, the main questions that we are trying to ask now whether a linear model is adequate? We have had some measures we have seen, but they are not adequate. We will use additional things and when we needed linear regression we did make additional assumptions although they have not been stated explicitly. We assumed that the errors that corrupt the independent variables are normally distributed and they have identical variance. Only under this these assumptions can you use a least squares method to perform linear regression that way you can at least prove that the least squares method has some nice properties.

So, we do not know whether this is true and we have to verify whether the errors are normally distributed and have equal variance. We also may have a problem of data containing outliers which we may have to remove and that also we have to solve. Additional questions may be that some observations may have unduly high influence than

others and we want to identify such points and perhaps remove them or at least be aware of this. And, lastly of course, a linear model may be inadequate so, we have to try and fit a non-linear model.

So, I am going to only address the first two questions; whether the errors are normally distributed? Whether they have equal variance and whether there are outliers in the data? These two things we will address using residual plots. So, let us do this illustrate with the Anscombe data set and also other data set.

(Refer Slide Time: 06:27)

A straightforward method for assessment of a model is by analysing residuals using *Residual plots*

Residual definition for OLS

$$e_i = y_i - \hat{y}_i, \quad i = 1, 2, \dots, n$$

➤ Variance of  $e_i$  is not same for all data points and also correlated

$$\text{Var}(e_i) = \sigma^2(1 - p_{ii}), \quad p_{ii} = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum(x_i - \bar{x})^2}$$

$$\text{Cov}(e_i e_j) = -\sigma^2(p_{ij}), \quad p_{ij} = \frac{1}{n} + \frac{(x_i - \bar{x})(x_j - \bar{x})}{\sum(x_i - \bar{x})^2}$$

So, one way of assessing whether there are outliers or whether linear model is adequate or not is using what we call the residual plots and let us see what these residuals are. By definition, a residual is the difference between the measured value of the dependent variable-the predicted value of the dependent variable for each sample. So,  $y_i$  represents the measured value,  $\hat{y}_i$  is the predicted value using the linear regression model that we have fitted. So, that difference is designated as  $e_i$  and it is called the residual and that is nothing, but the vertical distance between the fitted line and the observation point.

Now, we can try to compute the statistical properties of these residuals and we will be able to show that the variance of these residuals are not all identical even though we started with the assumption that all errors corrupting the measured values of the same variance, but if residual which is a result of the fit will not have the same variance for all data points. In fact, you can show that the variance of the  $i$ -th sample is  $\sigma^2$  which represents

the error in the measured value of the dependent variable multiplied by one-p ii, where p ii is divide by this.

Notice that  $p_{ii}$  depends on the i-th sample, numerator depends on the i-th sample therefore,  $p_{ii}$  depends on the i-th sampled and varies with sample to sample. So, the variance of the residual will not be identical for all samples, it is given by this quantity. We also can show that the residuals are not independent even though we assume that the errors corrupting the measurements are all independent. The residuals in the samples are not independent and they have a correlation covariance and that covariance can be shown to be given by this quantity.

The reason for the variance not being identical of the residuals or their them being correlated is because you notice that this  $\hat{y}_i$  we have actually have here is a result of the regression. It depends on all variables, all measurements; it is not depend only on the i-th measurement. This predicted value is a function of all the observations and that because of that it introduces a correlations between the different a residuals and also imparts different variance to different residuals.

And so, having derived this notice that even if we do not have a priori knowledge of sigma square which is the variance of error in the measurements we have we can estimate this quantity. We have already seen this in the previous lecture. We can replace this by SSE by n-2 which is an estimate of the sigma square and substitute this to get an estimated variance of each residual.

(Refer Slide Time: 09:34)

GyanData Private Limited

### OLS: Residual plots

- Standardized residual
- If residual variance is estimated from data then standardized residual has a t distribution with n-2 df

$$z_i = \frac{e_i}{s_e \sqrt{(1 - p_{ii})}}$$

Data Analytics

We will standardize these residuals, where what we mean by standardization is to divide the residual by its standard deviation estimated standard deviation ok. All of this can be computed from the data and therefore, you get for each sample a standardized residual after performing the linear fit which is given by this quantity.

Now, you can also show that this particular quantity the standardized residual will have a t distribution with n-2 degrees of freedom. Now, what these statistical properties allow you to know perform test on the residuals which what we will use to identify outliers and also test whether there is set the variances in the different measurements are identical or not.

(Refer Slide Time: 10:20)

The slide has a dark blue header bar with the GyanData Private Limited logo. The main title 'OLS: Residual plots' is in bold black font. Below it is a bulleted list:

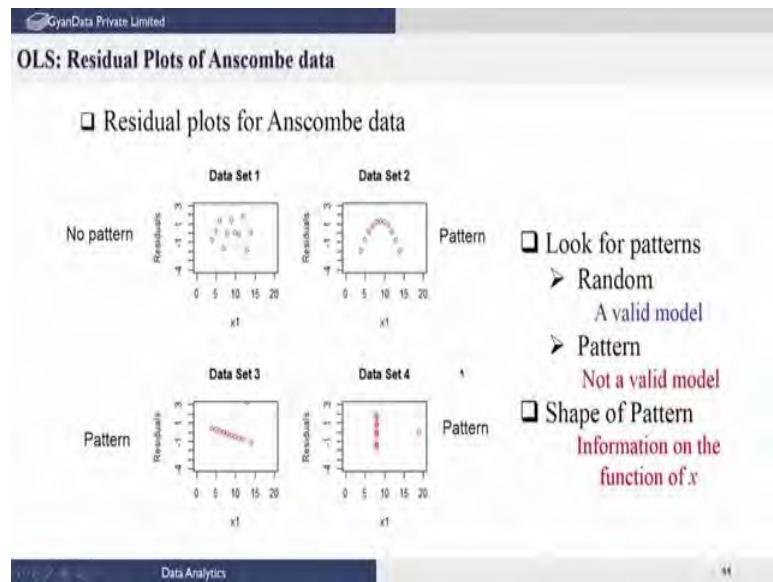
- ❑ Residual plot
- ❑ Plot of residuals vs predicted (fitted) value of dependent variable
- ❑ Residual plots are used for assessing
  - ❑ Validity of the linear model
  - ❑ Normality of the errors
  - ❑ Homoscedastic vs heteroscedastic error

At the bottom left of the slide is a small video player interface with the text 'Data Analytics' and a play button icon. A video frame shows a man with glasses and a blue shirt speaking.

So, we will plot the residual what we call residual plots we will plot the residuals with respect to the predicted or fitted value of the dependent variable. Remember, there is only one dependent variable, even if there are multiple independent variables we have only one dependent variable, we can plug the residuals with respect to the predicted value of the dependent variable and the predicted values will obtain after the regression, remember. So, this is called the residual plot.

What is called the residual versus the fitted or predicted value and this plot is very useful in testing the validity of the linear model in determining whether the errors are normally distributed, assumptions on errors are and whether the variances of all errors are identical or not which is called a homoscedasticity case which means the errors and all measured values are identical or the variance of the error in different measured values are non-identical which is called the heteroscedastic case or heteroscedastic error. So, let us see how each of these how the plot looks for each of these cases.

(Refer Slide Time: 11:28)



Now, let us plot the residual plots for the four data sets provided by Anscombe. Notice that we have done the regression model. Regression model we computed all these parameters R squared confidence, interval they all turned out to be identical. They gave us no clues whether the linear model is good for all four data sets or not. Basically, they say they would say that the linear model is adequate, but when we do the residual plot here we are plotted the residual versus the I have plotted with respect to the independent variable.

But, because it is a univariate case we are plotted with respect to the independent variable, but technically you should plot the residual with respect to the predicted value of the dependent variable. Remember, because we presume that the predicted variable is linearly dependent on x. In this case it may not matter, if the pattern will look the same you can try it out for yourself; if you plot the residual with respect to the predicted value of the dependent variable then you will get this kind of pattern of the residuals for the four data sets.

The first dataset if you look at it exhibits no pattern. The residuals seem to be randomly distributed between this case between -3 and plus 3 and whereas, for the second dataset there is a distinct pattern, the residuals look like a quadratic like a parabola and so, therefore, there exists a pattern in the data set 2. For the third data set basically you can say that there is no pattern except that our constant, more or less linear or constant. There seems to be a small bias because of the slope left in the residuals.

Data set 4 as we saw before, is a poorly designed experimental data set. All the y values are obtained at a single x value and that is what the residuals are also showing the 10 of the data points obtained that the same x x value or showing different – different residuals and the one single residual at a different x value showing something. So, from this you cannot judge anything, all you can say is that the experimental data set is very poorly designed and we need to get back to the experimenter and ask him to provide a different data set.

Now, based on this we can safely conclude that data set 1 clearly linear model is adequate all the measures previous measures also were satisfied and now the residual plot also shows a random pattern which means or random or what we call no pattern, then linear model is adequate whereas, for data set 2 by looking at the residual plot we can conclude that a linear model is inadequate should not be used for this data set.

For the third data set, however, we know there is one data point that is lying far away and perhaps that is the one that is causing all of this slightly linear pattern here. And, if we remove this outlier and retry it maybe this will resolve this problem will get resolved in linear model may be adequate for data set 3. For data set 4 again there is a distinct constant pattern and therefore, we can conclude that a linear model should not be used. In fact, no model should be used between x and y because y it does not seem to be dependent on x here.

So, the residual plot clearly gives the game away and it should be used along with other measures in order to finally, conclude that the linear model that were fitted for the data is acceptable or not. So, in this case data set 1 certainly will accept, data set 3 we will have to do further analysis, but for 2 and 4 we will completely reject the linear model.

(Refer Slide Time: 14:59)

The slide has a dark blue header with the text 'GyanData Private Limited'. The main title is 'Normal Q-Q Plot' in bold black font. Below the title is a bulleted list of points:

- Plot of sample quantiles against quantiles from normal distribution
- Quantile (or percentile) is the data value below which a certain percentage of data lies
- Quantiles for a standard normal distribution  
[10% 20% 30% 40% 50% 60% 70% 80% 90%]  
[-1.28 -0.84 -0.52 -0.25 0.0 0.25 0.52 0.84 1.28]
- Sample quantiles
  - Arrange standardized samples in increasing order
  - Choose number of quantiles
  - Find corresponding quantiles of standard normal distribution
  - If data points fall on 45 deg line, then data is from normal distribution
- Function `qqnorm(x)` in R

At the bottom left, there is a small bar with the text 'Data Analytics'.

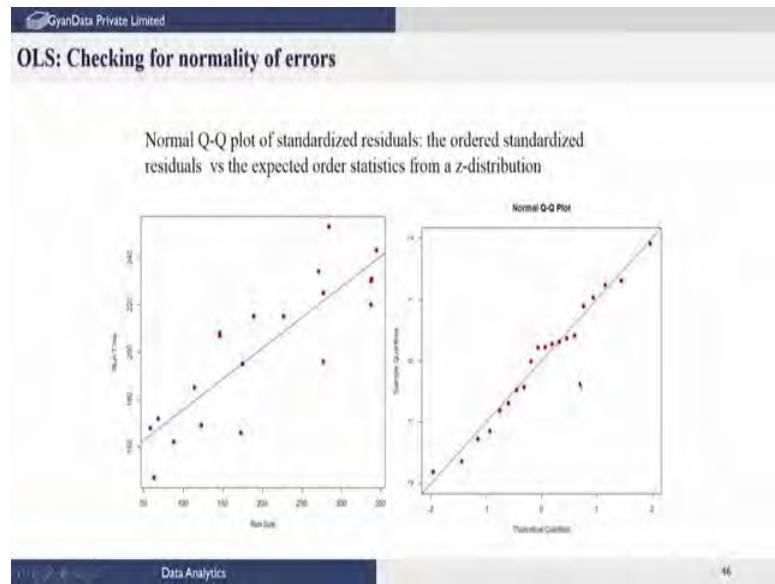
Now, the test for normality can also be done using the residuals. We have already seen the when we did statistical analysis the notion of a probability plot where we plot the sample quantiles against the quantiles from the distribution with which we want to compare. So, if we want to compare whether a set of given a given set of data follows a certain distribution then we plot the sample quantiles from the quantiles drawn for that particular distribution against which we want to test this sample.

So, in this case we want to test whether the residuals that we have the standardized residuals come from a standard normal distribution and therefore, we will take the quantiles from the standard normal distribution and plot it. Just to recap; what do we mean by a quantile? It is a percentile data value below which a certain percentage of data lies.

For example, if you want to find given a data set what is the value below which 10 percent of the data lies maybe-1.28, here we are given which means 10 percent of the samples lie below-1.28. 20 percent of the samples lie below-0.84 and so on and so forth we have computed this. This we can plot against the standard normal values 10 percent value probably where the probability between-infinity and the value is 10 percent and the value between-infinity and that value should be 20 percent and so on so forth. Those represents the x values corresponding to these probabilities and, we can use that plotted and then of course, before computing these contexts we have arranged the data.

So, we have seen this before I have just only recapped this and we can use what is called a qqnorm function in R to actually do it if you give the data set  $x$  and ask it to do a probability plot qqnorm will do this for you directly in R.

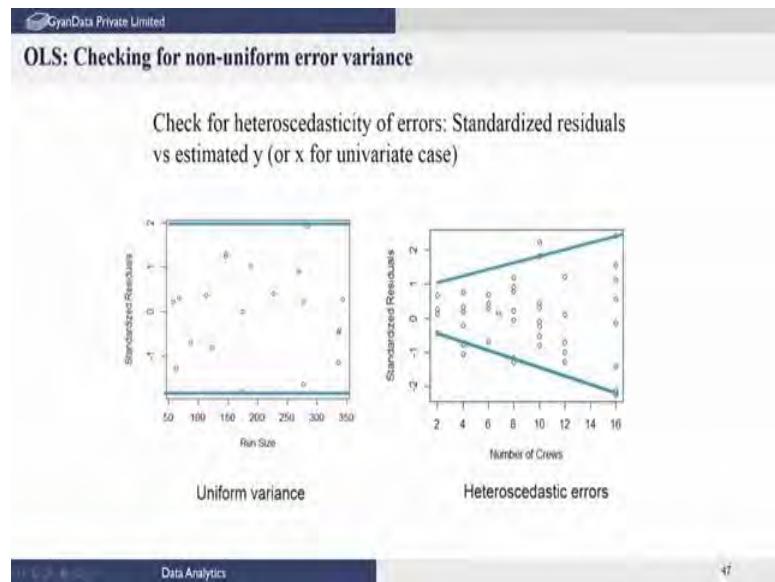
(Refer Slide Time: 17:02)



So, this is a sample Q-Q plot I have taken for some arbitrary random data set samples drawn from the standard normal distribution and you can see that if you do the normal Q-Q plot for the residuals after fitting the regression line it seems to closely follow the 45 degree line. So, the theoretical quantiles computed from the standard normal distribution and the quantiles computed from the sample residuals standardized residuals follow on the follow fall on the 45 degree line and therefore, in this case we can safely conclude that the errors in the data come from a standard normal distribution. So, a Q-Q plot.

If this thing is in does not happen, if we find the significant deviation of this quantiles from the 45 degree line then a normal distribution assumption is incorrect which means we have to modify the least squares objective function to actually accommodate this. It may not have may or may not have a significant effect on the regression coefficient, but there are ways of dealing with it which I will not go into.

(Refer Slide Time: 18:12)



A third thing that we need to test is whether the residual variances, I am sorry, the error variances in the data are having a uniform variance or I have different variances for different samples and again here what we do is look at the residual plot and standardized residuals versus the predicted values what you have to plot and if you do and look at this thing it seems to be that there is no particular trend in the residuals.

For example, in the right hand side we find that the residuals close to when the number of values is 2 and 2 is spread is very small whereas, when the number of crews is 16 the spread is very high. So, the spread increases or looks like a funnel when we actually look at the residuals whereas, such a effect is not found on the data set corresponding to the left hand side thing.

So, here I have plotted the standardized residual for two different data sets just to illustrate the type of figures you might get. If you get a figure such as in the left then we can safely conclude, that the errors in different measurements have the same variance whereas, if you have a funnel type of effect then you will know that the errors where a variances increases as the value increases. So, it depends on the value itself which implies that you cannot use a standard least squares method; you should use a weighted least squares method.

So, data points which are corresponding to this these four should be given more weight and data points corresponding to this will be given less weight and we call that a weighted least squares that is the way we have to deal with what we call heteroskedastic errors of

this guy. Again, I have I am not going to go into the whole thing I have just to illustrate that first the residual plots are used in order to verify the assumptions and if the assumptions are not valid, then we have correction mechanisms to modify our regression procedure.

But, linear this does not indicate a linear model is not adequate the linear model adequacy test is basically based on the pattern. If there is no pattern in the residuals you can go ahead and assume that the linear model fit is adequate as long as other measures are also satisfactory, but here it is related to the error variances and in this case we only modify the linear regression method and we still go with a linear model for these cases for this cases such as the one shown on the right.

(Refer Slide Time: 21:01)

GyanData Private Limited

### OLS: Checking for outliers in data

- ❑ Outliers: Points which do not conform to the pattern in bulk of the data
- ❑ Outliers can be identified using hypotheses test of residual of each sample
  - For a 5% level of significance a sample is considered an outlier if the corresponding standardized residuals of lie outside [-2, 2]
  - Even if several residuals lie outside confidence region, identify only one outlier at every iteration (corresponding to the sample with largest standard residual magnitude) – An outlier in a sample can ‘smear’ to other samples due to the regression
  - Apply regression to reduced sample set and iterate until no outliers are detected

The last thing that we need to do is also clean out the data. We do not want to use data that have got large errors what we call outliers minus points which do not conform to the pattern that is found in the bulk of the data. And, the outliers can be easily identified using hypothesis test of the residual for each sample, we have actually found the residual for each sample we have actually finally, found a standardized residual. So, the standardized residual roughly follow we know it is follows a T distribution, but we can for large enough number of samples we can assume that it follows a normal distribution.

So, if we I use a 5 percent level of significance we can run a test hypothesis test for each sample residual and if the residual lies outside -2 to plus 2, we can conclude that the sample

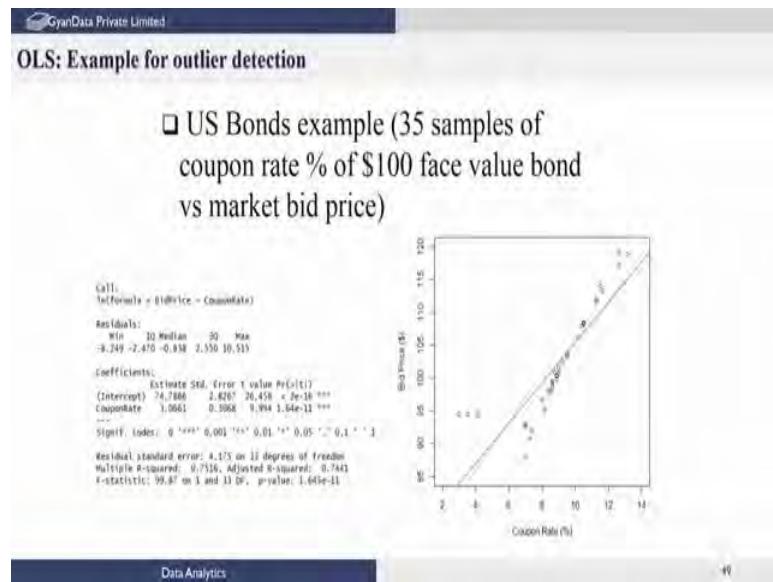
is an outlier. So, for each sample we test whether the sample standardized residual lies outside of this interval and if it is lies outside this interval we can conclude that that particular sample may be an outlier and remove it from a data set.

The only thing when we do outlier detection is this it may turn out that we do that first time we fit a regression and do an outlier detection we may find several residuals lying outside the confidence interval-2 to plus 2, 95 percent confidence interval in which case we do not throw all the samples out at the same time. We only throw out the one that is most offending which is we identify the outlier that corresponds to the sample with the largest standard standardized residual magnitude which of which is farthest away from-2 or plus 2 that is the one we take and remove it.

Once we remove that we again run a regression on the remaining samples and again run this outlier detection test. So, we remove only one outlier at a time the reason for this is when we perform an outlier detection we should be aware that a single outlier can smear affect the residuals of other samples because of our regression parameters are obtained from all the data points. Therefore, even a single outlier can cause other outliers to fall outside the confidence interval.

Therefore, we do not want to hastily conclude that all residuals following outside the confidence interval outliers, only the one that has the maximum magnitude we actually take it out and then we redo this so, that one at a time we do it will be a safe way of performing outlier detection.

(Refer Slide Time: 23:34)



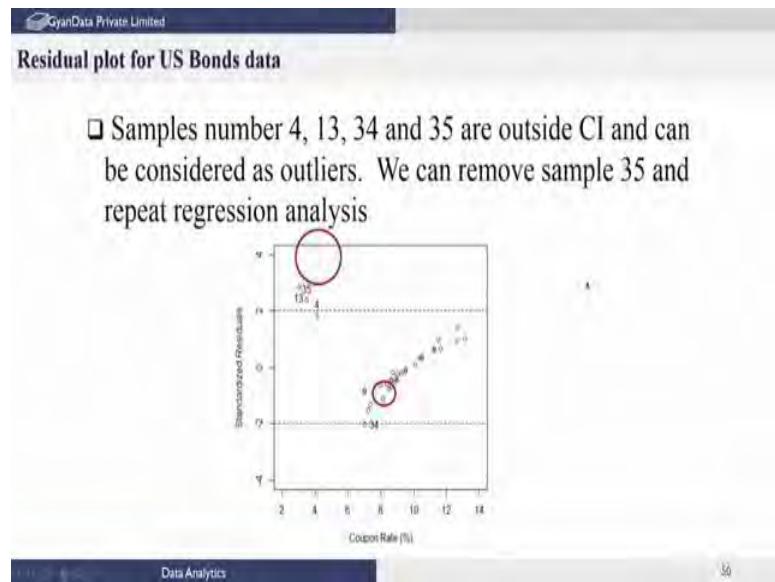
Again we will illustrate this with an example. Here is a US bonds example which consists of 35 samples, US bonds whose face value is 100 dollars and it is a guaranteed interest rate is provided for each of these bonds depending on when they were released and so on and there are different bonds with different interest rates. But, these are also traded in the market and their selling price in the market or bid price would be different depending on the kind of interest rate they attract.

So, you would presume that the bond which has a higher interest rate would have a higher market price. So, there might be a linear correlation on linear relation between the market price and the interest rate for that bond. So, here there are 35 samples that are obtained from a thing. These datasets are standard data sets that you can actually download from the net. If you just search for it, you will get it just like the Anscombe data set and what you called computer repair time data set that I have been using in the previous lectures.

If you perform a regression and you will get a fit of this kind ok. So, it shows that a linear fit seems to be adequate you can run the R command `lm` and you will find that the intercept is 74.8 and the slope is 3.6 and standard errors given and clearly the what you call the p-value is very very low which means that you will not reject the significance that is the intercept is significant and the slope is also significant, they are not close to 0. You can of course, compute confidence interval and come to the same judgment.

You can run an F-test. Here also it says the p-value of the F-test is 11 which means you will reject the null hypothesis and conclude that a full model is adequate which means the slope is important here ok. So, the R value seems to be reasonably good 0.75 and so, we can say the initial indicators are that a linear model is adequate. Now, let us go ahead and do the residual analysis for this.

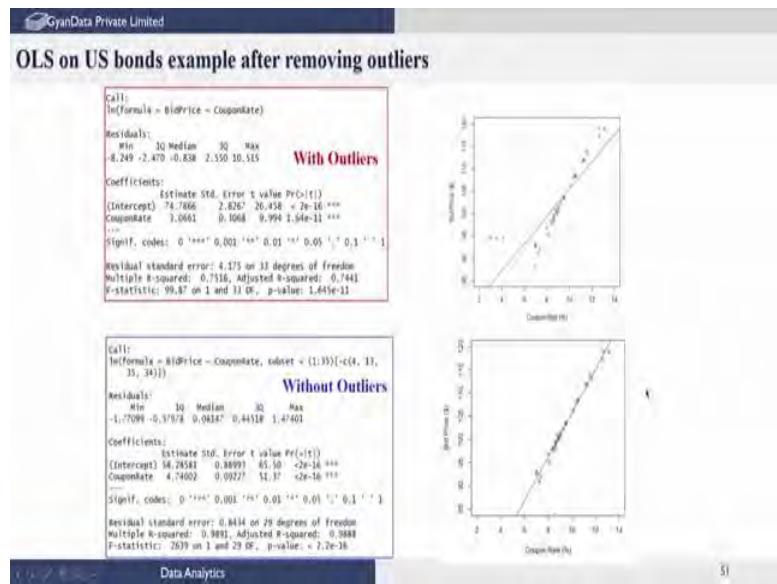
(Refer Slide Time: 24:44)



We perform a residual plot for standardized residual plot and we find that except for these four points 35 sample number 35, sample number 13, sample number 34, seems to be outside of the plus-2 confidence interval and they may be you may conclude that these are outliers while the others are within the bound and they are definitely not out outliers. They seem to be some kind of a pattern as the coupon rate increases the standardized residuals increase.

So, maybe as there is a certain amount of non-linearity in the model, but let us remove these outliers before making a final conclusion. So, we can remove all these four outliers at the same time if you want. As I said that is not a good idea perhaps we should remove only sample number 35 which has farthest away from the boundary with the residual with the largest magnitude and redo this because of lack of time I have just removed all four at the same time and then done the analysis. My suggestion is you do one at a time and then repeat this exercise for yourself.

(Refer Slide Time: 27:01)



Here we have removed these four samples 4, 13, 35 and thing and run the regression analysis again. You can see that the regression analysis maintain retaining all the samples is shown on the right hand side the plot the and their corresponding intercept coupon the slope as well as the F-test statistic and so on R squared values shown here.

And, once we remove these four samples which we outliers and then rerun it now the fit trims to be much much better. It is also seen on the left hand side that the fit is much better. You can see that the R squared value has gone up to 0.99 from 0.75 the again the test on the intercept and the coupon rate or slope shows that that they are significant and therefore, you should not assume that they are close to 0.

It also shows that the F-statistic has also a low p-value which means you take null hypothesis that a reduce model is adequate which means the linear model with the slope parameter is a much better fit. So, all of these indicators so seems to show that a linear model is adequate and the fit seems to be good, but we should do a residual plot again with this data and if that actually shows no pattern we can actually stop there. We can say there are no outliers and therefore, we can conclude that the regression model, that were fitted for this data is a reasonably good one.

Next class we will see how to actually extend all of these ideas to the multiple linear regression which consists of many independent variables and one dependent variable.

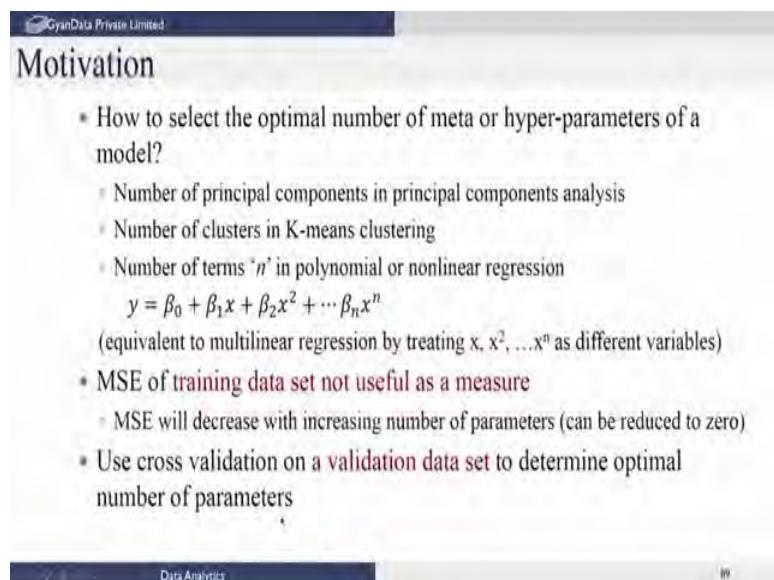
Thank you.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 33**  
**Cross Validation**

Welcome everybody to this last lecture on regression. In this lecture I am going to introduce a concept called Cross Validation which is a very useful thing in model building.

(Refer Slide Time: 00:27)



**Motivation**

- \* How to select the optimal number of meta or hyper-parameters of a model?
  - Number of principal components in principal components analysis
  - Number of clusters in K-means clustering
  - Number of terms ‘n’ in polynomial or nonlinear regression
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_n x^n$$
(equivalent to multilinear regression by treating  $x, x^2, \dots, x^n$  as different variables)
- \* MSE of **training data set** not useful as a measure
  - MSE will decrease with increasing number of parameters (can be reduced to zero)
- \* Use cross validation on a **validation data set** to determine optimal number of parameters

The main purpose of cross validation is to select what we call the number of meta parameters or hyper parameters of a model. Although we have not introduced principal component analysis in this series of lectures, nevertheless when you learn it in a higher advanced course on data analytics, you will come across this idea of principal components and one of the problems in principal component analysis is to select the number of principal components that are relevant.

We call this hyper parameter or a meta parameter of the model. Similarly later on in this course you will come across clustering in particular you will come across K means clustering and here again you have to choose the number of clusters required to group the data and the number of clusters is called the meta parameter of the clustering modeling. Similarly, if you’re building a non-linear regression model for example, let us take a

polynomial regression model, where the dependent variable  $y$  is written as a polynomial function of the independent variable  $x$ .

Let us assume there is only one variable  $x$ , you can write the regression model non-linear regression model as  $y = \beta_0 + \beta_1x + \beta_2x^2 + \dots + \beta_nx^n$ . where  $x^2, x^3$  and so on higher powers of  $x$ . This is known as a polynomial model notice this the polynomial model can also be the parameters of this model can be obtained using multi linear regression.

If you take  $x$  as a variable,  $x^2$  as a different variable and  $x^n$  as a different variable computed from data that you are given, treat them as different variables then you can use multi linear regression methods in order to estimate the parameters beta naught beta 1 and so on with the beta n. Here again we have to decide how many powers of  $x$  we have to choose.

So, if you choose higher powers of  $x$ , then corresponding to each power you got a extra parameter that you need to estimate. For example, in this case you have  $n + 1$  parameters if you have chosen,  $x$  power  $n$  as your highest degree of the polynomial. The choice of the degree of the polynomial to fit is again the meta parameter of the non-linear regression model. So, in all of this case is you have to find out the optimal number of meta parameter, optimal number of parameters of the model that you need to use in order to obtain the best model.

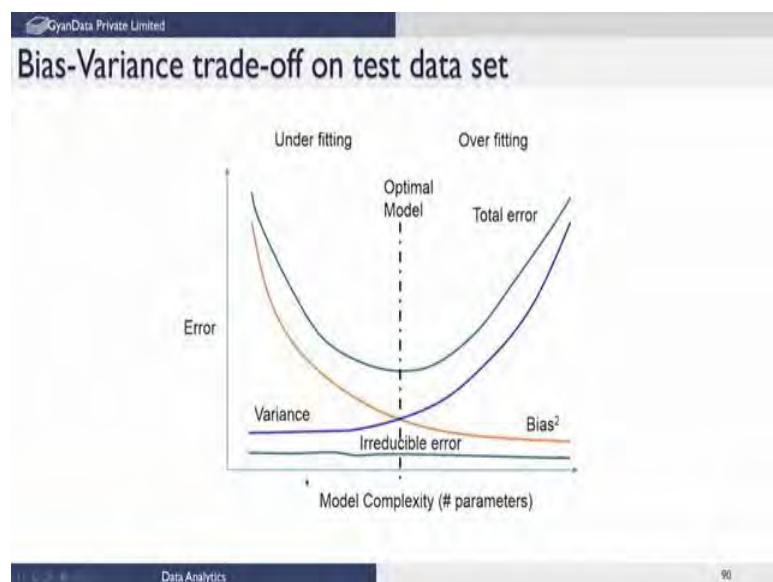
Now, you might think that it is easy to actually obtain this by looking at the mean squared error of the training data. So, for example, if you take this polynomial model fit it and compute the mean squared error in the training data, that is nothing, but the difference between  $y$  and  $y$  hat after you have found out the best fit parameters of beta naught beta 1 up to beta n, you can predict for every data by substituting the value of  $x$   $x$  squared and so on and you predict  $y$  hat and you compute the difference  $y - y$  hat and some over squared of all this is called the mean squared error ok.

So, the mean squared error of the training data you might think is useful as a measure for finding the optimal number of parameters, but that is not true. Because as you increase the number of parameters if you keep adding higher order terms, you will find that the mean squared error decreases. So, ultimately you can get the mean squared error to decrease to 0 as you increase the number of parameters, this is called over fitting.

So, you can always get the mean squared error of the training data to 0 by choosing sufficient number of parameters in the model. Therefore, you cannot use the training data set in order to find out the best number of optimal number of parameters to use in the model. So, we do something call cross validation, this has to be done on a different data set that is not used in the training.

We call this the validation data set and we use the validation data set in order to decide the optimal number of parameters of meta parameters of this model. So, we will use this polynomial regression model as an example throughout in order to illustrate this idea of cross validation.

(Refer Slide Time: 04:41)



So, schematically what happens when you actually increase the model complexity of the number of meta parameters of the model. So, you will find that the mean squared error on the test set continuously decreases ok. So, we will goes to a 0 as we said on the training set. However, on the validation set what will happen is, if you look at the mean squared error on the validation set that will initially decrease as you increase the number of parameters, but beyond the certain point, the mean squared error on the validation set will start increasing.

So, the optimal number of parameters you should choose of the model complexity is you choose corresponds to the minimum value of the mean squared error on the validation set and this is called the optimal model. If you choose less number of parameters than the

optimal model, we call this under fitting on the other hand if you use more parameters in your model, then the optimal model value is called over fitting.

So, over fitting basically means you are using unnecessarily more parameters than necessary to explain the data on the other hand if you use less parameters, you actually are not sufficiently your model is not going to be that accurate. Typically there are 2 measures for determine the quality of the model. One is called the bias in your prediction error and if you know if you increase the number of parameters of the model this bias squared of the bias term will start decreasing; however, the variability in your model predictions that will start increasing as you increase the model complexity of number of parameters.

So, it is basically the tradeoff between these two that gives rise to this minimum value of the MSE on the validation set that is what you are looking for. So, you want this optimal tradeoff between the bias which keeps reducing as you increase the number of parameters and the variance which keeps increasing as the number of parameters of the model increases. So, this is what we are going to find out by cross validation.

(Refer Slide Time: 06:45)

GyanData Private Limited

## Training and Validation data sets

- For large data sets divide data set into training data set (~ 70% of the samples) and remaining validation/test data
  - Training set:  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
  - Test set:  $(x_{0,i}, y_{0,i}) : i = 1 \dots n_t$  observations
- Training error rate
$$MSE_{Training} = \frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{x}_i^T \hat{\beta})^2$$
 ← Not of our interest for predictive ability of the model
- Test error rates
$$MSE_{Test} = \frac{1}{n_t} \sum_{i=1}^{n_t} (y_{0,i} - \mathbf{x}_{0,i}^T \hat{\beta})^2$$
 ← Of our interest

Data scarcity: Test data are not available

So, if you have a large data set then you can always divide this data set into 2 parts, 1 use for training typically 70 percent of the data samples you will use for training and the remaining you will set a part for the validation. So, let us call the samples that you used for training as  $(x_1, y_1), (x_2, y_2)$  and so on where x represents the independent variable y the

dependent variable and the validation set we will denote it by the symbols  $x_{0,i}$  and  $y_{0,i}$ , where there are  $n_t$  observations in the validation set.

So, typically as I said if you have a large number of samples, you can set up are the 70 percent of the samples for training and the remaining 30 percent you can use for validation. Now you can always of course, define the mean squared error in the training set after building the model.

So, this is nothing, but the difference between the measured observed value of the dependent variable - the predicted value after you estimated the parameters let say using least squares regression. So, this is the prediction error on the training data, square overall the samples and taken the average. That is the mean square error that we have seen before.

You can do a similar thing for the validation set also, you can take the difference between the measured value or observed value in the validation sample set - the predicted value for the validation samples and again you can take the sum squared difference between the observation - the predicted value for the validation set, squared over all samples and one the averaged average value. So, that is called the mean squared error on the test or validation.

So, this particular term as I said the MSE on training is not useful for the purpose of deciding on the optimal number of parameters of the model; however, this test MSE test or the mean squared error or the validation data set is the one that we are going to used for finding the optimal number of parameters of the model.

(Refer Slide Time: 08:49)

Validation Set Approach

- \* Enough data: (1) Training set, (2) Validation set, and (3) Test set
- \* Not enough data: Generate validation sets from a training set
- \* Validation set approach: Divides (often randomly) the training set into two parts

|                                    |         |       |
|------------------------------------|---------|-------|
| 1 2 3 4                            | n       |       |
| A training set                     | 1 2 3 4 | $n_t$ |
| A validation set (or hold-out set) | 1 2 3 4 | $n_v$ |

- \* Use training set, to fit the model
- \* Use validation set, to predict validation set errors
- Provides an estimate of test error rates

So, as I said if you have large number of amount of samples then you can actually divide it into a training set a validation set for finding the optimal number of parameters of the model, but and finally, if you want to assess how good your optimal model is you can run it on a test set.

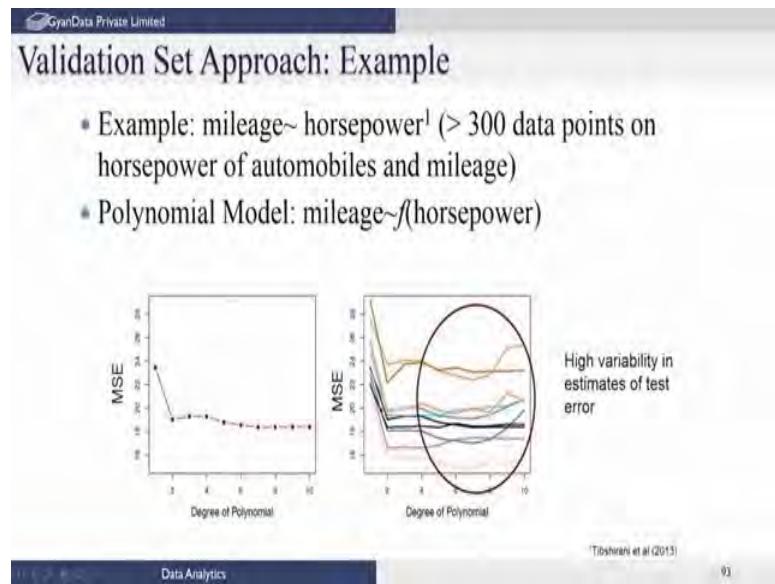
So, typically you take the data set and you divide into three parts; one the training set the validation set where you are trying to use for finding the optimal number of parameters and finally, the test set for to see whether them optimal model you have built is good enough. We will not worry about the test set in this particular lecture we will only worry about this validation set.

Unfortunately, if you do not have large number of samples. So, you would actually generate validation set from the training set itself and we will see how to do this using what is called k- fold cross validation and bootstrapping and so on. So, this is what we will do if you do not have enough data.

We will first look at the case of where you have sufficient number of samples. So, essentially you have  $n$  samples and you can divide into a training set consisting of  $n_t$  samples and the remaining samples you actually used for validation this is a hold out sample as we call it. So, you build a model using only the training set and after your build the model you test it, find the mean squared error on the validation set do this for every choice of the parameter.

So, if you have for example, a polynomial model, you will first see whether a linear model is good, then you will check as quadratic model and a cubic model and so, on you keep increasing the degree of the polynomial and for each case build the model using the training set and see how the MSE of that particular model is on the validation set, and plot this MSE on the validation set as a function of the degree of the polynomial. So, this is what we are going to do for one of the examples and then see how we pick the optimal number of parameters.

(Refer Slide Time: 10:49)



So, here is the case example of a mileage of some automobiles and the horse power of the engine. So, essentially this particular data set contains 300 data points, actually this is sufficiently large, but we are going to assume this is not large enough and we use the validation and cross validation approach on this data set. So, we have 300 data points or more of automobiles different types of automobiles, for which the horse power and the mileage is given.

We are going to fit a polynomial or a non-linear model between mileage and horse power we of course, we can also dry a linear model, but a polynomial model means you can also try quadratic and cubic models and so on so forth. So, this what we are going to illustrate. So, as we increase the degree of the polynomial, here what we have shown is the mean squared error on the validation set. So, suppose we take 70 percent of this is for training

and the remaining 30 percent 100 data points are so, for validation and we look at the mean squared error on the validation set for different choices of the polynomial order.

In this case the first polynomial degree is one implies we are taking a linear model and for two implies were to fitting a quadratic model, for three implies a cubic model and a quadratic model and so on so forth and we are tried polynomial up to degree 10 and we are shown how the mean squared error on the validation data set is as you increase the polynomial order.

You can see very clearly that the value reaches more or less a minimum at to and after which it does not significantly changed. Typically this should actually start increasing, but in most experimental data sets you will find that the mean squared error on the validation set flat ins out and does not significantly decrease beyond the point.

So, you can choose the optimal order degree of the polynomial to fit in this particular case as to, that is a quadratic model fixed this data very well that is what you actually conclude from this particular cross validation mechanism. Of course, on the right hand side we have shown plots for different choices of the training set. For example, if you choose 200 data points out of this randomly and perform regression polynomial regression for different polynomial degree and plot the MSE you will get let us say one curve in this case let us say the yellow curve you get here.

Similarly, if you take another random set and do it you will get another curve. So, these different curves correspond to different random samples taken from this 300 thing as training and the remaining is used as testing. You can see that as you increase the degree of the polynomial the variability or the estimates or the range of the estimates is very very large.

So, it indicates that if you over fit you will get a very high variability whereas, on the other hand if you choose order of the polynomial 1 or 2 you find that the variability is not that significant comparatively. So, typically if you over fit the model you will find high variability in your estimates that you obtain or the mean squared error values that you obtain ok.

(Refer Slide Time: 14:08)

The slide has a dark blue header bar with the text 'GyanData Private Limited' and a small logo. The main title 'Sampling for small data sets' is in bold black font. Below the title is a bulleted list:

- Validation of models by repeatedly drawing random samples from a training set
  - Validation set (random sampling)
  - K-fold cross validation
  - Bootstrap
- Objective: Predict the performance of model(s) on the validation/test sets (drawn from training data)
- Resampling methods useful for data scarce situations

All this is good if you have a large data set what happens when you have extremely small data set and you cannot divided into training and a validation set? You do not have sufficient samples for training. Typically you need a reasonable number of samples in the training set to build the model therefore, in this case we cannot set a part or divided into a 7030 what I call division and therefore, you have to do some of the strategies.

So, these strategies what are called cross validation using k- fold cross validation or a bootstrap that we will see. Here again we will predict the performance of the model on the validation set, but the validation set is not separated from the training set precisely, but on the other hand it is drawn from the training set and we will see how we do this. So, these methods K- fold cross validation is useful when we have very few samples for a training.

(Refer Slide Time: 15:05)

The diagram illustrates the process of Leave-one-out-cross-validation (LOOCV). It shows four samples labeled 1, 2, 3, 4. In each of the first three rows, one sample is highlighted in orange (1, 2, or 3 respectively), while the others are in green. This represents leaving one sample out for testing. The fourth row shows all four samples in green again, indicating the final result after all samples have been used as test samples.

\* Build model using  $(n-1)$  samples and predict the response ( $y_i$ ) for the remaining sample

$$CV_1 = \frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{x}_i^T \hat{\boldsymbol{\beta}}^{(1)})^2$$

Data Analytics 95

So, I will first start with leave one out cross validation what is called LOOCV. In this case we have n samples as I said n is not very large may be 20 or 30 samples we have for training. So, what we will do is, you first leave out the first sample and use the remaining samples for building your model; that means, you will use samples 2, 3, 4 up to n, it to build your model and once you have build the model you test the performance of the model or predict for this sample that you have left out and you will get an MSE for this sample.

And similarly in the next round what you do is leave out the second sample and choose all the remaining for training and then use that model for predicting on the sample that is left out. So, in every time you build the model by leaving out one sample from this list of n samples and predict the performance of the model on the left out sample. This you will do for every choice of the model parameter.

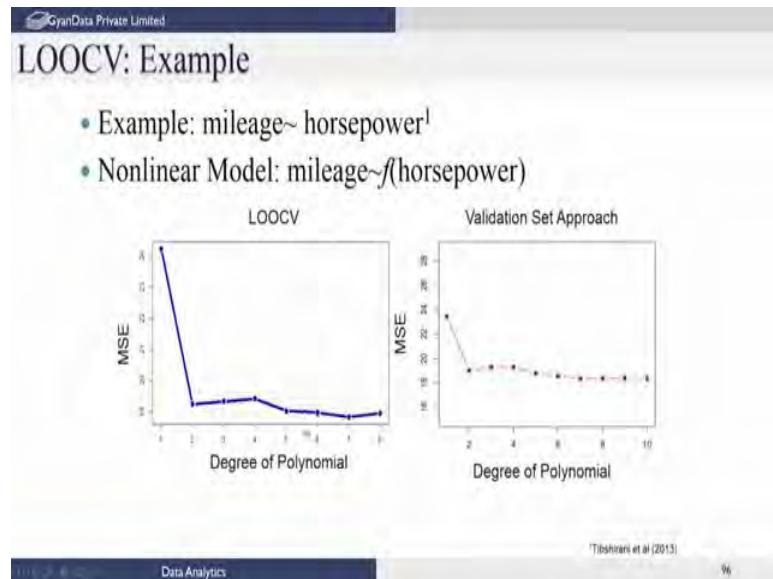
For example, if you are building a non-linear regression model, you will build regression model using let say only beta naught and beta 1 the linear term and predict a MSE on this left out sample. You will also build a model using the same training set quadratic model and predicted on this and so on so forth so, that you will get the MSE for the left out sample for all choices of the parameters you want to try out. Do this with the second sample being left out and the third sample being left out in term ok.

So, that every sample has a chance of being in the validation set and also be being part of the training set in the other cases. So, once you have done this for a particular choice of

the model parameters, let say you have building a linear model. You find out the sum squared value of the prediction errors on the left out sample over all the samples. For example, you would have got a MSE for this, MSE for this, MSE for this when the first sample second sample and third sample was left out that you are cumulating it here and taking the average of all that. This you do repeatedly for every choice of the parameters in the model.

For example, the linear, the quadratic, the cubic and so on so forth and you get the mean squared error or cross validation error for different values of the parameters which you can plot.

(Refer Slide Time: 17:33)



Here again we have shown the mean squared error for different choice of the degree of the polynomial for the same data set, in this case we have use to left leave one out cross validation strategy; that means, if you have 300 samples, we would have left out one sample build the model using 299 samples predicted on the sample that is left out, do this in term average over all of this for every choice of these parameter degree of the polynomial and mean squared error we have plotted.

Again we see that the MSE on the cross validation leave one out cross validation reaches more or less a minimum for a degree of polynomial equal 2 after which it just keeps remains more or less flat. So, the optimal in this case is also indicated as a second order a polynomial is best for this particular example.

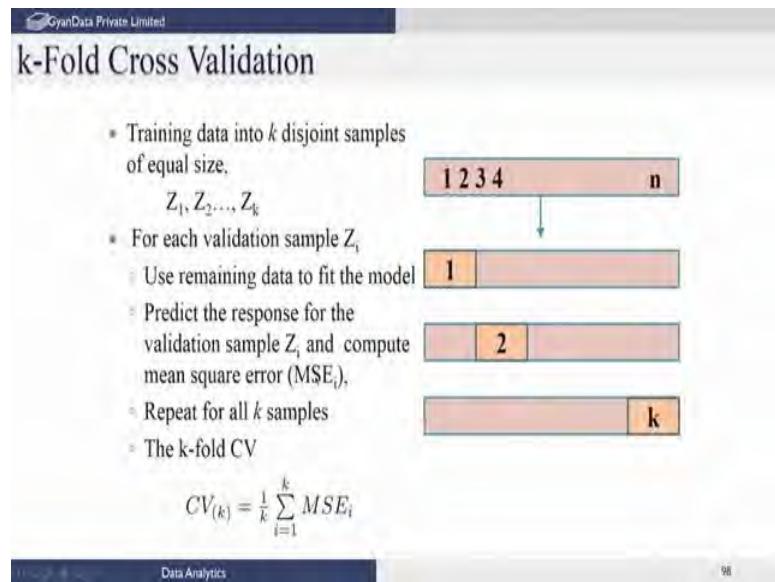
(Refer Slide Time: 18:20)

• Leave-one-out-cross-validation (LOOCV)  
• Advantages  
    Far less bias comparison to the validation set approach  
    Training set contains  $(n-1)$  observations each iteration  
    Yield the same results  
        No randomness in the training/validation set splits  
        Does not overestimate the test error rate as much as the validation set approach  
• Disadvantages  
    Expensive to implement due to fitting happens  $n$  times  
    It may select a model of excessive size (more variables) than the optimal model

So, leave one out cross validation as an advantage as compare to the validation set approach when 2 are we can show that it does not over estimate the test error rate as much as the validation set approach ok. It is comparatively expensive to implement because you are building the model n times one for each sample being left out and you have to repeat this for all choices of the hyperparameter model parameter.

For example you have to do this for the linear model, the quadratic model, the cubic model and so and so forth. So, you have not only have to do this n times, but you have to do this n times for every choice of the number of parameters of the model. So, it is quite a lot of computation that it takes. In general it may actually sometimes fit the model which is slightly more than the optimal model by not always, but sometimes it is also possible that the leave one out cross validation procedure over fits the model.

(Refer Slide Time: 19:21)



We can also do what is called the  $k$ - fold cross validation, where here instead of leaving one out we first divide the entire training set into  $k$  folds or  $k$  groups. So, let us say the first group contains let say the first four data samples, the second group contains the next four and so and so forth and you have divided this entire  $n$  samples into  $k$  groups. Now instead of leaving one out we will leave one group out.

So, for example, in this first case we will leave the first four samples they belonging to group 1 and use the remaining samples and build a model for whatever choice of the parameters where we have used let say we have building a linear model, we will use the remaining groups build the linear model and then predict for the set of samples in group 1 that was left out and compute the MSE for this group.

Similarly, in the next round what we will do is leave group 2 out build the model let say the linear model that we are building with the remaining groups and then find the prediction error for group 2 and so on so forth until we find the prediction error for group  $k$  and then we average over all these groups. So, the MSE in this case for all groups whether there are  $k$  groups and  $1$  by  $k$  that will be the cross validation error for leave this  $k$ - fold cross validation.

Now, you can you have to repeat this for every choice of the parameter you have done this for the linear model you have to do this for the quadratic model, cubic model and so on so

forth and then you can plot this cross validation error for leave for this k- fold cross validation.

(Refer Slide Time: 20:59)

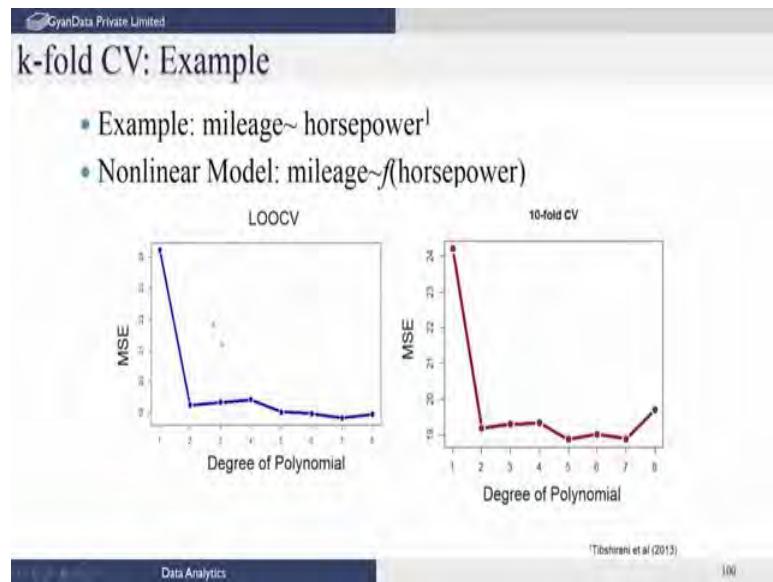
The slide has a header 'GyanData Private Limited' and a footer 'Data Analytics' and '99'.

**k-fold Validation**

- \* For  $k=n$ , Leave-one-out-cross-validation (LOOCV)
- \* In practice,  $k=5$  or  $10$  is taken,
- \* Less computation cost
- \* For computationally intensive learning methods
  - LOOCV fits the model  $n$  times
  - $k$ -fold CV fits the model  $k$  times

Notice that if  $k$  equals  $n$  you are essentially going back to leave one out cross validation in practice you can choose the number of groups equal to either 5 or 10 and do a 10 fold cross validation or 5 fold cross validation. This is obviously, less expensive computationally as compared to leave one out cross validation and as you see that leave one out cross validation we will do a model fitting  $n$  times for every choice of the parameter whereas,  $k$ -fold cross validation will do the model building  $k$  times for every choice of the parameter.

(Refer Slide Time: 21:35)



Again we have illustrated this k- fold cross validation for this mileage auto data, again we plot the MSE for different degrees of the polynomial. We have used to 10 fold cross validation and we are plotting this error and we will see that here also the minimal error occurs at 2 showing that the quadratic model is probably best for this particular data after which the error actually essentially flat and so.

So, cross validation is a important method or a approach for finding the optimal number of parameters of a model, this happens in clustering, this will happen in non-linear model fitting and principle component analysis and so on and its useful later on you will see in the clustering lectures, the use of cross validation for determining the optimal number of clusters.

Thank you.

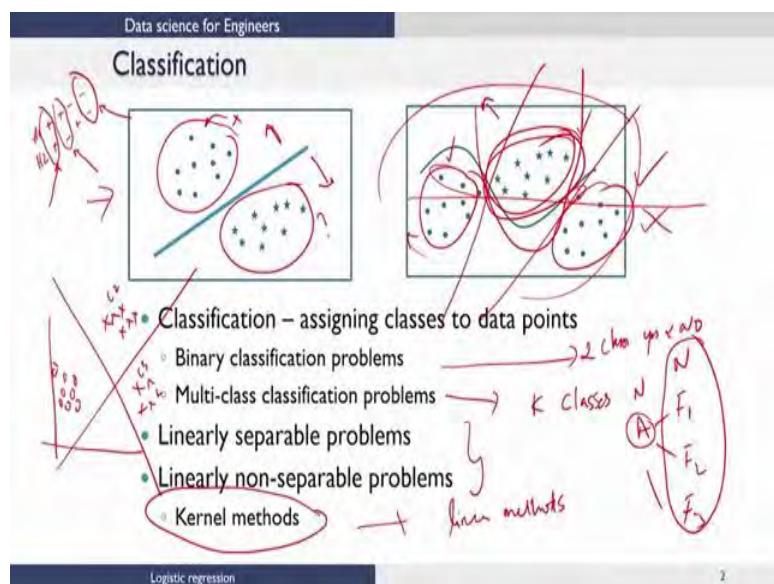
**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 34**  
**Classification**

Let us continue our lectures on Algorithms for Data Science. We will now see some algorithms that are used in what we call as the Classification problems; I will first start by discussing classification problems again. We have done this before, but I thought I will come back to this for this part of the lectures and then I will tell you the types of classification problems quickly. And then describe some characteristic that we look for in these classification problems and, then I will teach three techniques which could be used in solving classification problems.

In general many of the techniques that are used in classification can also be modified or used for function approximation problems. So, this is something that you should keep in mind. Similarly, techniques used for function approximation problems can also be modified and used for classification problems. Nonetheless in this series of lectures we will look at these algorithms and then give them a classification flavor and that would come to both the way we describe the algorithm and also the case studies that are used with the algorithms.

(Refer Slide Time: 01:49)



So, let us look at what classification means. So, we had described this before. If I am given data that is label two different classes and that is what I start with, then if I am able to develop an algorithm which will be able to distinguish these classes and how do you know that this algorithm distinguishes these classes. Whenever a new data point comes if I send it through my algorithm and if I originally had  $K$  different labels the algorithm should label the new point into one of the  $K$  groups. So, that is typically what is called a classification problem.

So, pictorially we represent here. Here we say here is a group of data, here is a group of data. Now, if I were to derive a classifier, then line like this could be a classifier and remember we have seen this in linear algebra and before. I have two half spaces and I could say this half space is class 1 and this half space is class 2.

Now, you notice that while I have data points only in a small region and this classifier has derived an unbounded a classification region. You could come up with classifiers which are bounded also we will discuss that later. But, here now if I get a new point if I get a point like this then I would like the classifier to say that this point most likely belongs to the class which is denoted by star points here. And, that is what would happen and similarly, if I have a point here I would like that to be classified to this class and so on.

Now, we can think of two types of problems the simpler type of classification problem is what is called the binary classification problem. Basically binary classification problems are where there are two classes, yes and no. So, examples are for example, if you have data for a process or an equipment and then you might want to simply classify this data as belonging to normal behavior of the equipment or an abnormal behavior of the equipment. So, that is just binary. So, if I get a new data, I want to say from this data if the equipment is working properly or it is not working properly.

Another example would be if let us say you get some tissue sample and you characterize that sample through certain means and then using those characteristics you want to classify this as a cancerous or non-cancerous sample. So, that is another binary classification example.

Now, more complicated version of this problem is what we call as a multi-class classification problem, where I have labels from several different classes. So here I have

just two, but in a general case in a multi-class problem I might have K classes. Classic example again going back to the equipment example that we just described instead of saying if the equipment is just normal or abnormal, if we could actually further resolve this abnormality into several different fault classes – let us say fault 1, fault 2, fault 3 then if you put all of this together normal fault 1, fault 2, fault 3 now you have four class problem.

So, if I have annotated data where the data is labeled as being normal or as being collected when fault F 1 occurred or as having been collected when fault F 2 occurs, fault F 3 occurs and so on, then whenever a new data point comes in we could label it as normal in which case we do not have to do anything or if we could label it as one of these fault classes, then we could take appropriate action based on what fault class it belongs to.

Now, from a classification viewpoint the complexity of the problem typically depends on how the data is organized. Now, if the data is organized let us talk about just binary classification problem and many of these ideas translate to multi-class classification problems. In a binary classification problem if the data is organized like it is shown in this picture here we call this data as linearly separable where I could use hyper plane to separate this data into two sides of the hyper plane or two half spaces and that gives me perfect classification. So, these are types of problems which are called linearly separable problems.

So, in cases where you are looking at linearly separable problems the classification problem then becomes one of identifying the hyper plane that would classify the data. So, these I would call as simpler problems in binary classification. However, I also have a picture on the right hand side; this also turns out to be a binary classification problem. However, if you look at this, this data and this data both belong to class 1 and this data belongs to class 2.

Now, however, you try to draw a hyper plane so, if I were to draw a hyper plane here and then say this is all class 1 and this is class 2; then these points are classified correctly, these points are classified correctly and these points are poorly classified or misclassified. Now, if I were to come up similarly with the hyper plane like this you will see similar arguments where these are points that will be poorly classified.

So, whatever you do if you try to come up with something like this, then these are points that would be misclassified. So, there is no way in which I can simply generate a hyper plane that would classify this data into two regions. However, this does not mean this is not a solvable problem, it only means that this problem is not linearly separable or what I have called here as linearly non-separable problems.

So, you need to come up with not a hyper plane, but very simply in Layman terms curved surfaces and here for example, if you were able to generate a curve like this and then you could use that as a decision function. And then say on one side of the curve I have data point belonging to class 2 and on the other side I have data points belonging to class 1.

So, in general when we look at classification problems we look at whether they are linearly separable or not separable. And, from data science viewpoint this problem right here becomes lot harder because when we look at a decision function in a linearly separable problem we know the functional form it is a hyper plane and we are simply going to look for that in the binary classification case.

However, when you look at non-linear decision boundaries there are many many functional forms that you can look at and then see which one works. For example, one could maybe come up with something like this or one could maybe come up with things where I just do something like this and so on. So, there are many many possibilities. Now, which of these possibilities would you use is something that the algorithm by itself has to figure out. So, since there are many possibilities this become harder problems to solve.

So, all of this we described for binary classification problems. Many of these ideas also translate to multi class problems; for example, if you take let us say I have data from three classes like this here. So, these are three classes; now, if I want to separate these three classes and then ask myself if I can separate these two through linear methods.

Now, it is slightly different from the binary classification problem because we needed only one decision function and based on one decision function we could say whether a point belongs to class 1 or class 2. In multi class problems you could come up with more decision functions and more decision functions would mean more hyper planes and then

you can use some logic after that to be able to identify a data point as belonging to a particular class.

So, when I have something like this here let us say this is class 1, this is class 2 and this is class 3. What I could possibly do is the following. I could do hyper plane like this and a hyper plane like this. Now, when I have these two hyper planes, then I have basically four combinations that are possible. So, for example, if I take hyper plane 1, hyper plane 2 as the two decision functions then I could generate four regions. For example, I could generate plus plus, plus minus, minus plus, minus minus.

So, you know that for a particular hyper plane you have two half spaces a positive half space and a negative half space. So, when I have something like this here then basically what it says is the point is in the positive half space of both hyper plane 1 and hyper plane 2. And, when I have a point like this says the point is in the positive half space of hyper plane 1 and the negative half space of hyper plane 2. And, in this case you would say it is in the negative half space of both the hyper planes.

So, now, you see that when you go to multi-class problems if we were to use more than one hyper plane and then depending on the hyper planes you get a certain number of possibilities. So, in this case when I use these two hyper planes I got basically four spaces as I show here. So, in this multi-class problem which is I have three classes; if I could have data belonging to one class falling here, data belonging to another class falling here and let us say the data belonging to the third class falling here. For example, then I could use these two hyper planes and the corresponding decision functions to be able to classify this three class problem.

So, when we describe multi-class problems, we look at more hyper planes and then depending on how we derive these hyper planes we could have these classes being separated in terms of half spaces or a combination of half spaces. So, this is another important idea that one should remember when we go to a multi-class problems. So, when you solve multi-class problems we can treat them directly as multi-class problems or you could solve many binary classification problems and come up with logic. On the other side of these classification results to label the result into one of the multiple classes that you have.

So, these give you some basic conceptual ideas on how classification problems are solved particularly binary classification problems and multi-class classification problems. The key ideas that I said that we want to remember here or whether these problems are linearly separable or linearly non-separable and in the linearly non-separable problems there are multiple options.

One way to address the multiple options is through a beautiful idea called Kernel methods where this notion of checking several non-linear surfaces can still be solved under certain conditions on the non-linear functions that we want to check using simple I would I am going to call it linear methods and I will explain this later in the course.

So, the idea here is that if you choose certain forms of non-linear functions which obey certain rules and those rules typically are called you know Kernel tricks then you could use whatever we use in terms of hyper planes, those ideas for solving those class of problems. So, Kernel methods are important when we have linearly non-separable problems, ok.

So, with this I just wanted to give you a brief idea on the conceptual underpinnings of classification algorithms. The math behind all of this is what we will try to teach at least some of it is what we will try to teach in this course. And, in more advanced machine learning courses you will see the math behind all of this in much more detail.

So, as far as classification is concerned we are going to start with an algorithm called logistic regression, we will follow that up with kNN classifier, then we will teach something called k-means, clustering k-means come under what are called clustering techniques. Now, typically you can use these clustering techniques in a function approximation or classification problems and I am going to teach these techniques and use case studies that give distinct classification flavor to the results that we are going to see using these techniques.

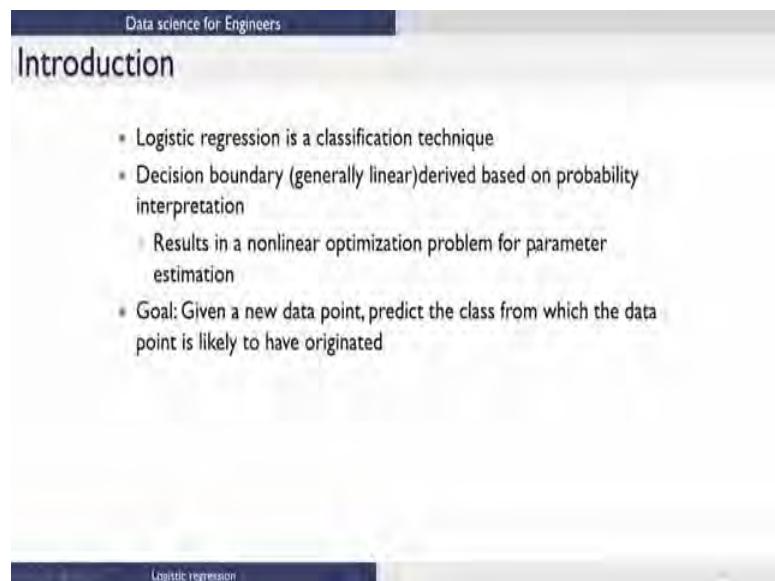
So, I will start logistic regression in the next lecture and I hope to see you then.

Thank you.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 35**  
**Logistic regression**

(Refer Slide Time: 00:23)



The slide has a dark blue header bar with the text "Data science for Engineers". The main title "Introduction" is centered above a bulleted list. The list contains the following points:

- Logistic regression is a classification technique
- Decision boundary (generally linear) derived based on probability interpretation
- Results in a nonlinear optimization problem for parameter estimation
- Goal: Given a new data point, predict the class from which the data point is likely to have originated

In this lecture I will describe a technique called Logistic regression. Logistic regression is a classification technique which basically develops linear boundary regions based on certain probability interpretations. While in general we develop linear decision boundaries this technique can also be extended to develop non-linear boundaries using what is called polynomial logistic regression. For problems where we are going to develop linear boundaries the solution still results in a non-linear optimization problem for parameter estimation as we will see in this lecture.

So, the goal of this technique is given a new data point I would like to predict the class from which this data point could have originated. So, in that sense this is a classification technique that is used in a wide variety of problems and it is surprisingly effective for a large class of problems.

(Refer Slide Time: 01:24)

Data science for Engineers

## Binary classification problem

- Classification is the task of identifying a category that a new observation belongs to based on the data with known categories
- When the number of categories is 2, it becomes a binary classification problem
- Binary classification is a simple “Yes” or “No” problem

Logistic regression



Just to recap things that we have seen before we have talked about binary classification problem before just to make sure that we recall some of the things that we have talked about before. We said classification is the task of identifying what category a new data point or an observation belongs to. There could be many categories to which the data could belong, but when the number of categories is 2 it is what we call as the binary classification problem.

We can also think of binary classification problems as simple yes or no problems where you either say something belongs to a particular category or no it does not belong to that category.

(Refer Slide Time: 02:13)

Data science for Engineers

## Input features

- Input features can be both qualitative and quantitative
- If the inputs are qualitative, then there has to be a systematic way of converting them to quantities
  - For example: A binary input like a "Yes" or "No" can be encoded as "1" and "0"
- Some data analytics approach can handle qualitative variables directly

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Handwritten notes:  
Qualitative data conversion:  
- Yes → 1  
- No → 0  
- Other values → 0.1, 0.3, 0.05, -2

Logistic regression

Now, whenever we talk about classification problems we have described this before we say it data is represented by many attributes  $x_1$  to  $x_n$ . We can also call this as input features as shown in the slide and these input features could be quantitative or qualitative. Now, quantitative features can be used as they are.

However, if we are going to use a quantitative technique, but we want to use input features which are qualitative, then we should have some way of converting this qualitative features into quantitative values. One simple example is if I have a binary input like a yes or no for a feature. So, what do we mean by this? So, I could have yes let us say a 0.1, 0.3 and then another data point could be no 0.05 - 2 and so on.

So, you notice that these are quantitative numbers while these are qualitative features. Now, you could convert this all into quantitative features by a coding yes as 1 and no as 0. So, then those also become number. There is a very crude way of doing this there might be much better ways of coding qualitative features into quantitative features and so on. You also have to remember that there are some data analytics approach that can directly handle these qualitative features without a need to convert them into numbers and so on. So, you should keep in mind that that is also possible.

(Refer Slide Time: 04:01)

Data science for Engineers

## Linear classifier

- Decision function is linear
- Binary classification can be performed depending on the side of the half-plane that the data falls in
- We saw this before in the linear algebra module
- However, simply guessing "yes" or "no" is pretty crude
- Can we do something better using probabilities ?

Logistic regression

Raghunatha

Now, that we have these features we go back to our pictorial understanding of these things just for the sake of illustration, let us take this example where we have this 2 dimensional data. So, here we would say  $x$  is  $x_1, x_2$  variables let us say  $x_1$  is here  $x_2$  is here. So, its organized into data like this now let us assume that all the circular data belong to 1 category and all the stored data belong to another category. Notice that circle data would have certain  $x_1$  and certain  $x_2$  and similarly started I would have certain  $x_1$  and certain  $x_2$ .

So, in other words all values of  $x_1$  and  $x_2$  such that the data is here belongs to 1 class and such that the data is here belongs to another class. Now, if we were able to come up with hyper plane such as the one that is shown here we learn from our linear algebra module that to one side of this hyper plane is a half space this side is half space and depending on the way the normal is defined you would have a positive value and a negative value to each side of the hyper plane. So, this is something that we are dealt with in detail in 1 of the linear algebra classes.

So, if you were to do this classification problem then what you could say is if I get a data point somewhere here I could say it belongs to whatever this class is here. So, let us call this for example, we could call this class 0 we could call this class 1 and we would say whenever a data point falls to this side of a line then it is class 0 and if a data point falls to this side of the line we will say its class 1 and so on.

However, notice that any data point, so whether it falls here or it falls here we are going to say class 0, but intuitively you know that if this is really a true separation of these classes, then this is for sure going to belong to class 0. But as I go closer and closer to this line there is this uncertainty about whether it belongs to this class or this class because data is inherently noisy. So, I could have a data point which is true value here; however, because of noise it could slip to the other side and so on.

So, as I come closer and closer to this then you know the probability or the confidence with which I can say it belongs to a particular class can intuitively come down. So, simply saying yes this data point and this data point belongs to class 0 is 1 answer, but that is pretty crude. So, the question that this logistics regression addresses can we do something better using probabilities.

So, I would like to say that the probability that this belongs to class 1 is much higher than this because its far away from the decision boundary. So, how do we do this? Is a question that logistics regression addresses.

(Refer Slide Time: 07:41)

Data science for Engineers

## Output

- Why model probabilities ?
  - The probability of a “Yes” or “No” gives a better understanding of the sample’s membership to a particular category
  - Estimating the binary outputs from the probabilities is straight forward through simple thresholding
  - How does one model this probability ?

Logistic regression

So, as I mentioned before the probability of something being from a class if we can answer that question that is better than just saying yes or no answers right. So, one could say yes this belongs to a class a better nuanced answer could be that yes it belongs to a class, but with a certain probability as the probability is higher, then you feel more confident about assigning that class to the data.

On the other hand if we model through probabilities we do not want to lose binary answer like yes or no also. So, if I have probabilities for something I can easily convert them to yes or no answers through some thresholding which we will see in the logistics regression methodology when we describe that. So, while we do not lose the ability to categorically say if a data belongs to a particular class or not by modeling this probability. On the other hand we get a benefit of getting a nuanced answer instead of just saying yes or no.

(Refer Slide Time: 08:50)

So, the question then is how does one model these probabilities? So, let us go back and look at the picture that we had before let us say this is  $x_1$  and  $x_2$ . Remember that this hyper plane would typically have this form here the solution is written in the vector form, if I want to expand it in terms of  $x_1$  and  $x_2$  what I could do is I could write this as

$$p(x) = \beta_0 + \beta_{11}x_1 + \beta_{12}x_2.$$

So, this could be the equation of this line in this 2 dimensional space. Now, one idea might be just to say this itself is a probability and then let us see what happens. The difficulty with this is this  $p$  of  $x$  is not bounded because its just a linear function whereas, you know that the probability has to be bounded between 0 and 1. So, we have to find some function which is bounded between 0 and 1.

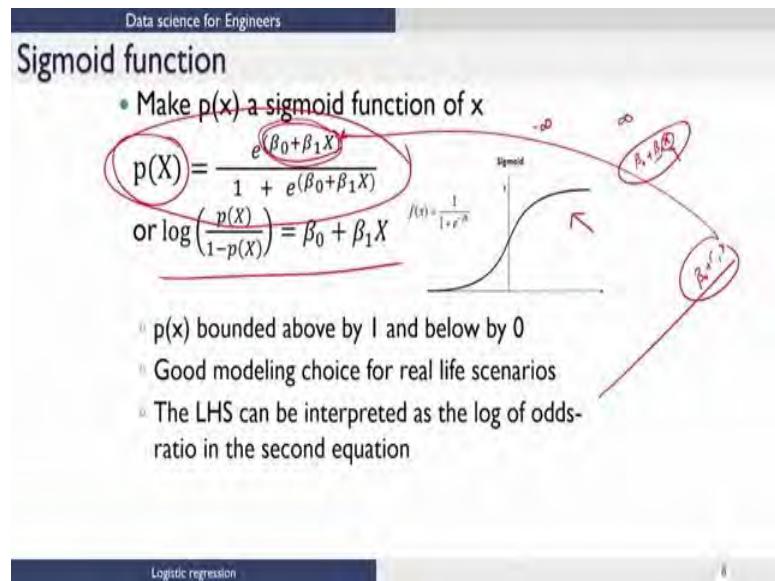
The reason why we are still talking about this linear function is because this is the decision boundary. So, what we are trying to do here is really instead of just looking at this decision

boundary and then saying yes and no + and -. What we are trying to do is we are trying to use this equation itself to come up with some probabilistic interpretation that is a reason why we are still sticking to this equation and trying to see if we can model probabilities as a function of this equation which is the hyper plane.

So, you could think of something slightly different and then say look instead of saying  $p(x)$  is this let me say  $\log(p) = \beta_0 + \beta_1 X_1$  in this case you will notice that it is bounded only on one side in other words if I write  $\log(p) = \beta_0 + \beta_1 X_1$  will ensure that  $p(x)$  never becomes negative.

However, on the positive side  $p(x)$  can go to  $\infty$  that again is a problem because we need to bound  $p(x)$  between 0 and 1. So, this is an important thing to remember. So, it only bounds this one side. So, is that something more sophisticated we can do.

(Refer Slide Time: 11:13)



The next idea is to write  $p(x)$  as what is called a sigmoidal function; the sigmoidal function as relevance in many areas. So, this is a function that is used in neural networks and other very interesting applications. So, the sigmoid has an interesting form which is shown here.

Now, let us look at this form right here. I want you to notice two things; number 1 is still we are trying to stick this hyper plane equation into the probability expression because that is a decision surface. Remember intuitively somehow we are trying to convert that hyper

plane into a probability interpretation. So, that is the reason why we are still sticking to this  $\beta_0 + \beta_1 X$ .

Now, let us look at this equation and then see what happens. So, if you take this argument  $\beta_0 + \beta_1 X$ . So, that argument depending on the value of  $x$  you take could go all the way from  $-\infty$  to  $\infty$  right. So, just take a single variable case if I write let us say  $\beta_0 + \beta_1$  just 1 variable  $X$  not a vector.

Now, if  $\beta_1$  is positive if you take  $x$  to be a very very large value this number will become very large if  $\beta_1$  is negative if you take  $x$  to very very large value in the positive side this number will become  $-\infty$ . And similarly if  $\beta_1$  takes the other values you can correspondingly choose  $x$  to be positive or negative and then make this unbounded between  $-\infty$  to  $\infty$ .

So, we will see what happens to this function when  $\beta_0 + \beta_1 X$  is  $-\infty$  you would get this to be  $e^{-\infty}$  divided by  $1 + e^{-\infty}$  or you can just think of this as very large number. So, in that case the numerator will become 0 and the denominator will become  $1 + 0$ . So, on the lower side this expression will be bounded by 0.

Now, if you take  $\beta_0 + \beta_1 X$  to be a very large positive number, then the numerator will be a very very large positive number and the denominator will be  $1 +$  that very large positive number. So, this will be bounded by 1 on the upper side. So, now from the equation for the hyper plane we have been able to come up with a definition of a probability which makes sense which is bounded between 0 and 1. So, its an important idea to remember.

By doing this what we are doing is the following, if we were not using this probability all that we will do is we look at this equation and whenever a new point comes in we will evaluate this  $\beta_0 + \beta_1 X$ . And then based on whether its positive or negative we are going to say yes or no right.

Now, what has happened is instead of that this number is put back into this expression and depending on what value you get you get a probabilistic interpretation that is a beauty of this idea here. You can rearrange this in this form and then say

$$\log \left( \frac{p(x)}{1 - p(x)} \right) = \beta_0 + \beta_1 X$$

the reason why I show you this form is because the left hand side could be interpreted as log of odds ratio which is an idea that is used in several places. So, that is the connection here.

(Refer Slide Time: 14:56)

Data science for Engineers

## Estimation of the parameters

- We find parameters in such a way that plugging these in the model equation should give the best possible classification for the inputs from both the classes
- This can be formalized by maximizing the following likelihood function

$$L(\beta_0, \beta_1) = \prod_{i=1}^n (p(x_i))^{y_i} (1 - p(x_i))^{(1-y_i)}$$

when  $x_i$  belongs to class 0,  $y_i = 0$   
when  $x_i$  belongs to class 1,  $y_i = 1$

$p(x)$

Logistic regression

Now, we have these probabilities and remember if you were to write this hyper plane equation as the way we wrote in the last few slides  $\beta_0 + \beta_{11}x_1 + \beta_{12}x_2$ . The job of identifying a classifier as far as we are concerned is done when we identify values for the parameters  $\beta_0$ ,  $\beta_{11}$  and  $\beta_{12}$ . So, we still have to figure out what are the values for this.

Once we have a value for this any time I get a new point I simply put it into the  $p$  of  $x$  equation that we saw in the last slide and then get a probability right. So, this still needs to be identified. And; obviously, if we are looking at a classification problem where I have this on this side and starts on this side, I want to identify these  $\beta_0$ ,  $\beta_1$ ,  $\beta_{11}$  and  $\beta_{12}$  such a way that this classification problem is solved.

So, I need to have some objective for identifying these values right. Remember in the optimization lectures I told you that all machine learning techniques can be interpreted in some sense as an optimization problem. So, here again we come back to the same thing and then we say look we want to identify this hyper plane, but I need to have some objective function that I can use to identify these values.

So, these  $\beta_0$ ,  $\beta_{11}$  and  $\beta_{12}$  will become the decision variables, but I still need an objective function. And as we discussed before when we are talking about the optimization techniques the objective function has to reflect what we want to do with this problem. So, here is an objective function looks little complicated, but I will explain this as we go along.

So, I said in the optimization lectures we could look at maximizing or minimizing. In this case what we are going to say is I want to find values for  $\beta_0$ ,  $\beta_{11}$  and  $\beta_{12}$  such that this objective is maximized. So, take a minute to look at this objective and then see why someone might want to do something like this. So, when I look at this objective function let us say I again draw this and then let us say I have these points on one side and the other points on the other side.

So, let us call this class 0 and let us call this class 1. So, what I would like to do is I want to convert this decision function into probabilities. So, the way I am going to think about this is when I am on this line I should have the probability being equal to 0.5 which basically says that if I am on the line I cannot make a choice between class 1 and class 2 because the probability is exactly 0.5. So, I cannot say anything about it.

Now, what I would like to do is you can interpret it in many ways; one thing would be to say as I go away from this line in this direction I want the probability of the data belonging to class 1 to keep decreasing. The minute that the probability that the data belongs to class 1 keeps decreasing that automatically means since there are only 2 classes and this is a binary classification problem, the probability that the data belongs to class 0 keeps increasing.

So, if you think of this interpretation where as I go from here. So, here the probability that the data point belongs to class 1 let us say is 0.5, then basically it could either belong to class 1 or class 0. And if it is such that the probability keeps decreasing here of the data point belonging to class 1 then it has to belong to class 0. So, that is a basic idea.

So, in other words we could say the probability function that we define before should be such that whenever a data point belongs to class 0 and I put that into that probability expression I want a small probability. So, you might interpret that probability as the probability that the data belongs to class 1 for example. And whenever I take a data point from this side and put it into that probability function then I want the probability to be very

high because I want that is the probability that the data belongs to class 1. So, that is a basic idea.

So, in other words we can paraphrase this and then say for any data point on this side belonging to class 0, we want to minimize  $p(x)$  when  $x$  is substituted into that probability function and for any point on this side when we substitute these data points into the probability function we want to maximize that probability. So, if you look at this here what this says is if this data point belongs to class 0 then  $y_i$  is 0.

So, whenever a data point belongs to class 0 anything to the power 0 is 1; so this will vanish. So, in the product there will be functions of this form which will be  $1 - p(x_i)$  and because  $y_i$  is 0 this will become 1. So, this will become something to the power 0 1. So, this term will vanish and the only thing that will remain is  $1 - p(x_i)$ . So, if we try to maximize  $1 - p(x_i)$ , then that is equivalent to minimizing  $p(x_i)$ .

So, for all the points that belong to class 0 we are minimizing  $p(x_i)$ . Now, let us look at the other case of a data point belonging to class 1 in which case  $y_i$  is 1. So,  $1 - 1 = 0$ ; so this term will be something to the power 0 which will become 1. So, it kind of drops out. So, the only thing that will remain is  $p(x_i)$  now  $y_i$  is 1, so power 1 it will be just left with  $p(x_i)$ . And since this data belongs to class 1 I want this probability to be very large. So, when I maximize this it will be a large number.

So, you have to think carefully about this equation there are many things going on here; number 1 that this is a multiplication of the probabilities for each of the data points. So, this includes data points from class 0 and class 1. The other thing that you should remember is let us say I have a product of several numbers, if I am guaranteed that every number is positive right then the product will be maximized when each of these individual numbers are maximized. So, that is a principle that is also operating here that is why we do this product of all the probabilities.

However, if a data point belongs to class 1 I want probability to be high. So, the individual term is just written as  $p(x_i)$ , so this is high for class 1. When a data point belongs to class 0 I still want this number to be high; that means, this number will be small. So, it automatically takes care of this as far as class 0 and class 1 are concerned.

So, while this looks little complicated this is written in this way because its easier to write this as 1 expression. Now let us take a simple example to see how this will look. Let us say I have class 0. I have 2 data points  $x_1$  and  $x_2$  and class 1. I have 2 data points  $x_3$  and  $x_4$ . So, this objective function when its written out would look something like this.

So, when we take let us say these points belonging to class 0, then I said the only thing that will be remaining is here. So, this will be  $1 - p(x_1)$ , for the second data point it will be  $1 - p(x_2)$ , then for the data third data point will be  $p(x_3)$  and for the fourth data point it will be  $p(x_4)$ . So, this would be the expression from here.

So, now, when we maximize this then since  $p$  of  $xs$  are bounded between 0 and 1 this is a positive number; this is a positive number; positive number positive number and if the product has to be maximized, then each number has to be individually maximized; that means, this has to be maximized. So, it will go closer and closer to 1 the closer to 1 it is better.

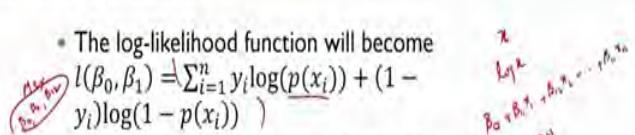
So, you notice that to  $x_4$  would be optimized to belong in class 1 similarly  $x_3$  would be optimized to belong in class 1 and when you come to these 2 numbers you would see that this would be a large number if  $p(x_1)$  is a small number. So,  $p(x_1)$  basically means that  $x_1$  is optimized to be in class 0 and similarly  $x_2$  is optimized to be in class 0. So, this is an important idea that you have to understand in terms of how this objective function is generated.

(Refer Slide Time: 24:44)

**Data science for Engineers**

## Log-likelihood function

- The log-likelihood function will become  

$$l(\beta_0, \beta_1) = \sum_{i=1}^n y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))$$

- Simplifying this expression and using the definition for  $p(x)$  will result in an expression with the parameters of the linear decision boundary  

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x)}}$$
- Now the parameters can be estimated by maximizing the above expression using any nonlinear optimization solver

**Logistic regression**

Now, one simple trick you can do is take that objective function and take a log of that and then maximize it. So, if I am maximizing a positive number  $x$  then that is equivalent to maximizing  $\log(x)$  also. So, whenever this is maximized that will also be maximized. The reason why you do this it makes the product into some makes it looks simple. So, remember from our optimization lectures we said we are going to maximize this objective. So, we always write this objective in terms of decision variables and the decision variables in this case are  $\beta_0$ ,  $\beta_{11}$  and  $\beta_{12}$  as we described before.

So, what happens is each of these probability expressions if you recall from your previous slides will have these 3 variables and  $x_i$  are the points that are already given, so you simply substitute them into this expression. So, this whole expression would become a function of  $\beta_0$ ,  $\beta_{11}$  and  $\beta_{12}$  right.

Now, we have come back to our familiar optimization territory where we have this function which is a function of these decision variables this needs to be maximized and this is an unconstrained maximization problem because we are not putting any constraints or  $\beta_0$ ,  $\beta_1$  and  $\beta_2$ . So, they can take any value that we want. And also the fact that the way the probability is defined this would also become a non-linear function. So, basically we have a non-linear optimization problem in several decision variables and you could use any non-linear optimization technique to solve this problem.

And when you solve this problem what you get is basically the hyper plane. So, in this case its a 2 dimensional problem. So, we have 3 parameters now if there is an  $n$  dimensional problem if you have let us say  $n$  variables. So, I will have something like  $\beta_0 + \beta_{11}x_1 + \beta_{12}x_2 + \dots + \beta_{1n}x_n$  this will be an  $n + 1$  variable problem there are  $n + 1$  decision variables. These  $n + 1$  decision variables will be identified through this optimization solution. And for any new data point once we put that data point into the  $p(x)$  function that sigmoidal function that we have described, then you would get the probability that it belongs to class 0 or class 1.

So, this is the basic idea of logistic regression. In the next lecture I will take a very simple example with several data points to show you how this works in practice and I will also introduce a notion of regularization which would help in avoiding over fitting when we do logistics regression. I will explain what over fitting means in the next lecture also. With that you will have a theoretical understanding of how logistics regression works and in a

subsequent lecture Doctor Hemant Kumar would illustrate how to use this technique in our on a case study problem.

So, that will give you the practical experience of how to use logistics regression and how to make sense out of the results that you get from using logistics regression on an example problem.

Thank you and I will see you in the next lecture.

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 36**  
**k-Nearest Neighbors (kNN)**

In this lecture, we will look at a very very simple yet powerful classification algorithm called the k-Nearest Neighbors. So, let me introduce a k-Nearest Neighbor classification algorithm.

(Refer Slide Time: 00:31)

Data science for Engineers

## Introduction

- k Nearest Neighbors(kNN) is a non-parametric method used for classification
- It is a lazy learning algorithm where all computation is deferred until classification
- It is also an instance based learning algorithm where the function is approximated locally

Handwritten notes on a 2D plot showing data points (circles and crosses) and a decision boundary (a line with a shaded region). A point 'e' is shown with arrows pointing to its neighbors '1' and '2'. Below the plot is a small video player interface with a play button and the text 'k-Nearest Neighbors'.

It is what is called a non-parametric algorithm for classification. So, let me explain what this non-parametric means. Remember, when we looked at logistics regression for example, we said let us say there is data like this and we are going to use the train data to develop a hyper plane of this form  $\beta_0 + \beta_{11}x_1 + \beta_{12}x_2$  in the 2D case and any time a new data point comes what we do is we use the parameters that have been estimated from the train data to make predictions about test data.

So, remember we had this  $e^{\text{power this term}} / (1 + e^{\text{power this term}})$  right here. So, this function is actually a function of  $\beta_0 + \beta_{11}x_1 + \beta_{12}x_2$ . So, these are all parameters that I have already been derived out of this data and any time a new test data comes in, it is sent through this function and then you make a prediction. So, this is a parametric method because parameters have been derived from the data.

k-Nearest Neighbor is a different idea where I do not get parameters like this out of the data. I am going to use the data itself to make classification. So, that is an interesting different idea that one uses in k-Nearest Neighbors. I just want to make sure that we get the terminology right. We will later see that the k-Nearest Neighbor there is one parameter that we use for classifying which is the number of neighbors that I am going to look at. So, I do not want you to wonder since we are using anyway a parameter in the k-Nearest Neighbor why am I calling it non-parametric.

So, the distinction here is subtle, but I want you to remember this the number of neighbors that we use in the k-Nearest Neighbor algorithm that you will see later is actually a tuning parameter for the algorithm that is not a parameter that I have derived from the data. Whereas, in logistic regression these are parameters I can derive only from the data. I cannot say what these values will be a priori whereas, I could say I will use a k-Nearest Neighbor with 2 neighbors, 3 neighbors and so on. So, that is a tuning parameter.

So, I want you to remember the distinction between a tuning parameter and parameters that are derived from the data and the fact that k-Nearest Neighbor is a non-parametric method speaks to the fact that we are not deriving any parameters from the data itself; however, we are free to use tuning parameters for k-Nearest Neighbors. So, that is an important thing to remember.

It is also called a lazy learning algorithm where all the computation is deferred until classification. So, what we mean by this is the following: if I give train data for example, for logistic regression I have to do this work to get these parameters before I can do any classification for a test data point right. So, without these parameters I can never classify test data points. However, in k-Nearest Neighbor you just give me data and a test data point I will classify. So, we will see how that is done, but no work needs to be done before I am able to classify a test data point. So, that is another important difference between k-Nearest Neighbor and logistic regression for example.

It is also called as an instant based learning where the function is approximated locally. So, we will come back to this notion of local as I describe this algorithm.

(Refer Slide Time: 04:35)

Data science for Engineers

## Why kNN and when does one use it?

- Why kNN ?
  - Simplest of all classification algorithms and easy to implement
  - There is no explicit training phase and the algorithm does not perform any generalization of the training data
- When does one use this algorithm?
  - When there are nonlinear decision boundaries between classes
  - When the amount of data is large

k-Nearest Neighbors

Now, we might ask when do we use this as I started this lecture I mentioned it simplest of classification algorithms, very easy to implement and you will see when I explain the algorithm how simple it is.

There is no explicit training phase and so on and there is no generalization for the training data and all that. It just that I give the data and then I just wait till they give me a new data point to say what class it should belong to. Of course, based on the algorithm itself I can also predict for the train data points itself what class they should belong to and then maybe compare it with the label that the data point has and so on. Nonetheless, I am not going to explicitly get some parameters out.

And, when does one use this algorithm? This is a simple algorithm when there are complicated non-linear decision boundaries this algorithm actually works surprisingly well and when you have large amount of data and the train phase can be bogged down by large number of data in terms of an optimization algorithm and so on then you can use this. However, a caveat is you will see as we describe this algorithm when you have more and more data the classification of nearest neighbor itself will become complicated.

So, there are ways to address this but when we say when the amount of data is log all that we are saying is since there is no explicit training phase. There is no optimization with a large number of data points to be able to identify parameters that are useless at later in classification. So, in other words in other algorithms you will do all the effort a priori and

once you have the parameters then it classification becomes on the test data point becomes easier.

However, since k and then is a lazy algorithm all the calculations are deferred till you actually have to do something at that point there might be a lot more classic lot more calculations if the data is large.

(Refer Slide Time: 06:53)

Data science for Engineers

## k Nearest Neighbors

- Input features
  - Input features can be both quantitative and qualitative
- Outputs
  - Outputs are categorical values, which typically are the classes of the data
- kNN explains a categorical value using the majority votes of nearest neighbors

k-Nearest Neighbors

So, the input features for k-Nearest Neighbors could both quantitative and qualitative and output are typically categorical values which are what type of class does this data belong. Now, it is not necessary that we use k-Nearest Neighbor only for classification though that is where it is used the most you could also use it with very very simple extensions or simple definitions for function approximation problems also and you will see as I describe this algorithm how it could be adapted for function approximation problems quite easily.

Nonetheless, as far as this lecture is concerned we are going to say the outputs or categorical values which basically says different classes and what class does this data point belong to. In one word if you were to explain k-Nearest Neighbor algorithm you would simply say k-Nearest Neighbor explains a categorical value using the majority vote of nearest neighbors. So, what basically we are saying is if there is a particular data point and I want to find out which class this data point belongs to all I need to do is look at all the neighboring data points, and then find which class they belong to and then take a majority vote and that is what is the class that is assigned to this data point.

So, it is something like if you want to know a person you know his neighbors something like that is what is used in k-Nearest Neighbors.

(Refer Slide Time: 08:35)

Data science for Engineers

## Assumptions

- Being nonparametric, the algorithm does not make any assumptions about the underlying data distribution
- Select the parameter  $k$  based on the data
- Requires a distance metric to define proximity between any two data points
  - Example: Euclidean distance, Mahalanobis distance or Hamming distance

k-Nearest Neighbors

Now, remember at the beginning of this portion of data science algorithms I talked about the assumptions that are made by different algorithms. Here for example, because this is a non-parametric algorithm we really do not make any assumptions about the underlying data distribution, we are just going to look at the nearest neighbors and then come up with an answer. So, we are not going to assume probability distribution or any other a separable ad assumptions and so on.

As I mentioned before this  $k$  the number of neighbors we are going to look at is a tuning parameter and this is something that you select right. So, you use a tuning parameter, run your algorithm and you get good results then keep that parameter if not you kind of play around with it and then find the best  $k$  for your data.

The key thing is that because we keep talking about neighbors and from a data science viewpoint whenever we talk about neighbors we have to talk about a distance between a data point and its neighbor. We really need a distance metric for this algorithm to work and this distance metric would basically say how what is the proximity between any two data points. The distance metric could be Euclidean distance, Mahalanobis distance, Hamming distance and so on.

So, there are several distance metrics that you could use to basically use k-Nearest Neighbor.

(Refer Slide Time: 10:11)

Data science for Engineers

## Algorithm

- The kNN classification is performed using the following four steps
- Compute the distance metric between the test data point and all the labeled data points
- Order the labeled data points in the increasing order of this distance metric
- Select the top  $k$  labeled data points and look at the class labels
- Find the class label that the majority of these  $k$  labeled data points have and assign it to the test data point

$k=3$     $k=1$

X<sub>1</sub>, X<sub>2</sub>, X<sub>3</sub>, X<sub>4</sub>, X<sub>5</sub>, X<sub>N</sub>   X<sub>new</sub>

c<sub>1</sub>, c<sub>2</sub>

$d_1, d_2, \dots, d_N$

$N \rightarrow d_1 \rightarrow c_1$   
 $X_1 \rightarrow d_2 \rightarrow c_1$   
 $X_2 \rightarrow d_3 \rightarrow c_1$   
 $X_3 \rightarrow d_4 \rightarrow c_1$   
 $X_4 \rightarrow d_5 \rightarrow c_1$   
 $X_5 \rightarrow d_6 \rightarrow c_1$   
 $\vdots$   
 $X_N \rightarrow d_N \rightarrow c_1$

X<sub>new</sub> →  $d_{N+1}$  → c<sub>1</sub>

k-Nearest Neighbors

So, in terms of the algorithm itself it is performed using the following four steps. Nothing is done till the algorithm gets a data point to be classified. Once you get a data point to be classified let us say I have  $N$  data points in my database and each has a class label. So, for example,  $X_1$  belongs to class 1,  $X_2$  belongs to class 1,  $X_3$  belongs to class 2 and so on  $X_N$  belongs to let us say class 1. So, this is you know a binary situation, binary classification problem. This need not be a binary classification problem.

For example,  $X_2$  could belong to class 2 and so on. So, there may be many classes. So, multi class problems are also very very easy to solve using kNN algorithm. So, let us anyway stick to binary problem. Then what you are going to do is let us say I have a new test point which I call it  $X_{\text{new}}$  and I want to find out how I classify this. So, the very first which is what we talk about here is we find a distance between this new test point and each of the label data points in the data set.

So, for example, there could be a distance  $d_1$  between  $X_{\text{new}}$  and  $X_1$ ,  $d_2$  between  $X_{\text{new}}$  and  $X_2$ ,  $d_3$  and so on and  $d_N$ . So, once you calculate this distance then what you do is you have  $n$  distances and this is the reason why we said you need a distance metric right in last slide for a kNN to work. Once we have this distance then what we do is basically we look at all of these distances, and then say I order that the distances from the smallest to the largest.

So, let us say if  $d_N$  is the smallest distance so,  $d_N$  may be  $d_5, d_3, d_{10}$  whatever it is. So, I order them this is the smallest to the largest you can also think of this as closest to the farthest right because the distances are all from  $X_{\text{new}}$ . So, the distance is 0 then the point is  $X_{\text{new}}$  itself. So, any small distance is the closest to  $X_{\text{new}}$  and as you go down it is farther and farther.

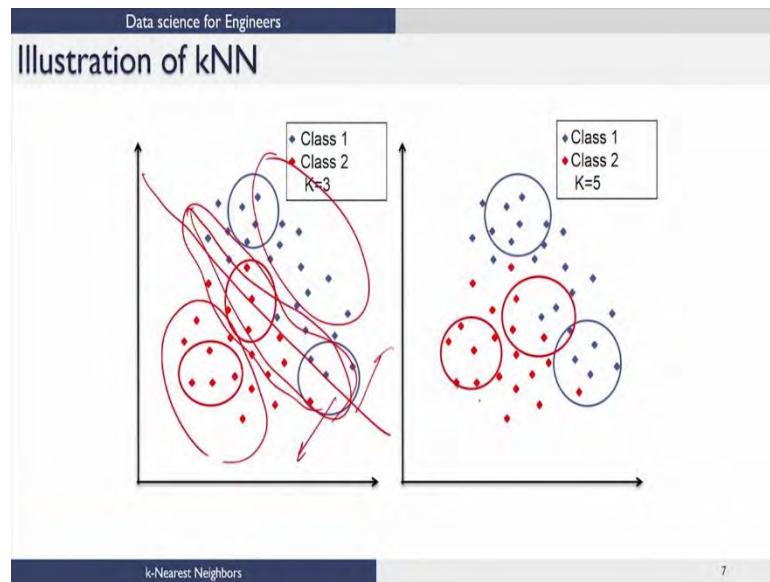
Now, the next step is very simple if let us say you are looking at k-Nearest Neighbors with K equal to 3, then what you are going to do is you are going to find the first three distances in this and this distance is from  $X_N$ , this distance is from  $X_5$  and this distance is from  $X_3$ . So, once we order this according to distance and go from the smallest to largest once we started in this fashion then we also know what the corresponding data points are. So, this belongs this is the data point  $X_N$ . So, this is the distance between  $X_N$  and  $X_{\text{new}}$ , distance between  $X_5$  and this. So, now, I have these three data points that I picked out from the data set.

Now, if I want to classify this all that I do is the following. I find out what class these data points belong to. So, if all of them belong to class 1 then I say  $X_{\text{new}}$  is class 1; if all of them belong to class 2 I say  $X_{\text{new}}$  is class 2; if two of them belong to class one and the third one belongs to class 2 I do a majority vote and still say it is class 1. If two of them belong to class 2 and one belongs to class 1, I say it is class 2, that is it that is all the algorithm is. So, it says to find the class label that the majority of this K label data points have and I assign it to the test data point, very simple right.

Now, I also said this algorithm with minor modifications can be used for function approximation. So, for example, if you so choose to you could take this and then let us say if you want to predict what an output will be for a new point, you could find the output value for these three points and take an average for example, very trivial and then say that is the output corresponding to this. So, that becomes adaptation of this for function approximation problems and so on. Nonetheless for classification this is the basic idea.

Now, if you said K is equal to 5 then what you do is you go down to 5 numbers and then do the majority voting. So, that is all we do.

(Refer Slide Time: 15:03)



So, let us look at this very simple idea here let us say this is actually the training data itself and then I want to look at K equal to 3 and then see what labels will be for the training data itself and the blue are actually labeled. So, this is supervised right. So, the blue are all belonging to class 1 and the red is all belonging to class 2 and then let us say for example, I want to figure out this point here blue point.

Though I know the label is blue what class would k-Nearest Neighbor algorithm say this point belongs to? Say, if I want to take K equal to 3 then basically I have to find three nearest points which are these three. So, this is what is represented and since the majority is blue this will be blue. So, basically if you think about it this point would be classified correctly and so on.

Now, even in the training set for example, if you take this red point I know the label is red; however, if I were to run k-Nearest Neighbor with three data points when you find the three closest point they all belong to blue. So, this would be misclassified as blue even in the training data set. So, you will notice one general principle is there is a possibility of data points getting misclassified only in kind of this region where there is a mix of both of these data points. However, as you go farther away the chance of miss classification it keeps coming down.

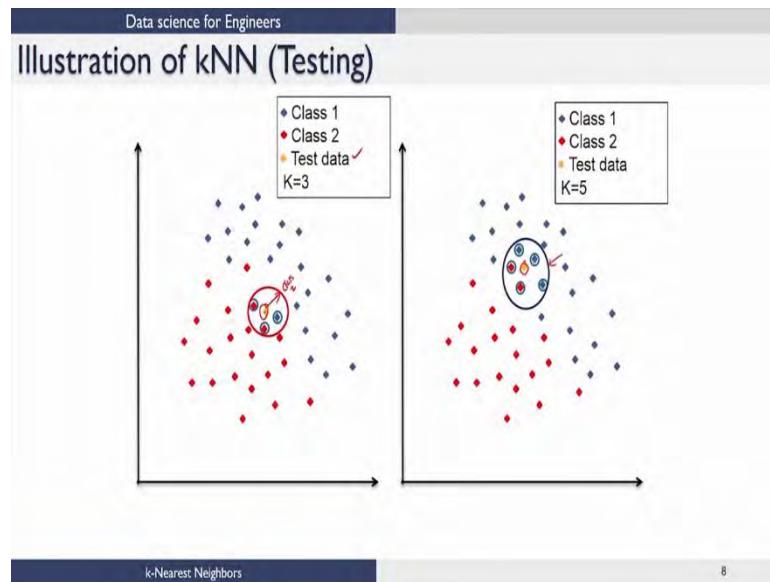
So, again in some sense you see a parallel saying if I were to draw a line here and then say this is all red class this is all blue class then points around this line is where the problem is

right as they go farther away the problems are less. Nonetheless, notice how we have never defined a boundary line or a curve here at all right. The data points themselves tell you how this boundary is not.

So, in that sense it is while it is simple it is also in some sense sophisticated because we never have to guess a boundary, the data points themselves define a boundary in some sense. So, I can actually effectively use this algorithm for complicated non-linear boundaries which you would have to guess a priori if we were using a parametric approach. So, that is the key idea here.

A similar illustration for K equal to 5, now if I want to let us say a check this data point from the training set itself then I look at it is five neighbors closest 1 2 3 4 5, all of them are red. So, this is classified as red and so on. So, this is the basic idea.

(Refer Slide Time: 18:03)



Now, you do not have to do anything till you get a data point. So, you could verify how well the algorithm will do on the training set itself. However, if you give me a new test data here so, which is what is shown by this data point then if you want to do a classification there is no label for this. Remember, the other red and blue data points already have a label from prior knowledge; this does not have a label. So, I want to find out a label for it.

So, if I were to use K equal to 3, then for this data point I will find the three closest neighbors, they happen to be these three data points. Then I will notice that two out of these are red. So, this point will get a label class 2. If on the other hand the test data point is here and you were using K equal to 5, then you will look at the five closest neighbor to this point and then you see that two of them are class 2 and three are class 1. So, majority voting this will be put into class 1. So, you will get a label of class 1 for this data point.

So, this is the basic idea of k-Nearest Neighbor. So, very very simple.

(Refer Slide Time: 19:17)

Data science for Engineers

## Things to consider

- Following are some things one should consider before applying kNN algorithm
  - Parameter selection
  - Presence of noise
  - Feature selection and scaling
  - Curse of dimensionality

k-Nearest Neighbors

However, these are some of these the things to consider before applying kNN. So, one has to choose the number of neighbors that one is going to use the value of K whether 3 5 7 9 whatever that is and the results can quite significantly depend on the parameter that you choose particularly when you have noise in the data and that has to be taken into account.

The other thing to keep in mind when using kNN is that when you do a distance between two data points  $X_1$  and  $X_2$  let us say and, let us say there are n components in these. The distance metric will take all of these components into picture. So, since we are comparing distance then that basically means every attribute for this data in this data point we are comparing distances.

The problem with this is that if for example, there are a whole lot of attributes which actually are not at all important from a classification viewpoint, then what happens is

though they are not important from a classification viewpoint they contribute in the distance measure. So, there is there is a possibility of this features actually kind of spoiling the results in k-Nearest Neighbor. So, it is important to pick features which are which are of relevance in some sense. So, the distance metric actually uses only features which will give it the discriminating capacity. So, that is one thing to keep in mind.

The other the problem is these are these are handle able, these are rather easily handled, but these are things to keep in mind when you look at these kinds of algorithms and also particularly this being the first course on data science I am assuming for most of you these are kinds of things that you might not have thought about before. So, it is worthwhile to kind of think about this do some mental experiments to see why these kinds of things might be important and so on.

Now, the other aspect is scaling. So, for example, if there are two attributes, let us say in data temperature and concentration and temperatures are in values of 100 concentrations are in values of 0.1, 0.2 and so on when you take a distance measure and these numbers will dominate over this. So, it is always a good idea to scale your data in some format before doing this distance, otherwise while this might be an important variable from a classification viewpoint it will never show up because these numbers are bigger and they will simply dominate the small number. So, feature selection and scaling are things to keep in mind.

And, the last thing is curse of dimensionality. So, I told you that while this is a very nice algorithm to apply because there is not much computation that is done at the beginning itself. However, if you notice if I get a test data point and I have to find let us say the five closest neighbors there is no way in which I can do this it looks like, unless I calculate all the distances. So, that can become a serious problem if the number of data points in my database is very large.

Let us say I have 10000 data points and let us assume that I have an algorithm k-Nearest Neighbor algorithm with K equal to 5. So, really what I am looking for is finding 5 closest data points from this data base to this data point. However, it looks like I have to calculate all the 10000 distances and then sort them and then pick the top 5. So, in other words to get this top 5 I have to do. So, much work right. So, there must be smarter ways of doing

it but nonetheless one has to remember that the number of data points and number of features one has to think how to apply this algorithm carefully.

(Refer Slide Time: 24:01)

Data science for Engineers

## Parameter selection

- The best choice of  $k$  depends on the data
- Larger values of  $k$  reduce the effect of noise on classification but makes the decision boundaries between classes less distinct
- Smaller values of  $k$  tend to be affected by the noise with clear separation between classes

OOO

k-Nearest Neighbors

So, the best choice of  $k$  depends on the data and one general rule of thumb is if you use large values for  $k$  then clearly you can see you are taking lot more neighbors. So, you are getting lot more information. So, the effect of noise on classification can become less. However, if you take large number of neighbors, then your decision boundaries are likely to become less crisp and more diffuse right. So, because if let us say there are two classes like this then for this data point if you take a large number of neighbors then you might pick many neighbors from the other class also. So, that can make the boundaries less crisp and more diffuse.

On the fifth slide flipside if you use smaller values of  $k$  then your algorithm is likely to be affected by noise and outliers. However, your decision boundaries as a rule of thumb are likely to become crisper. So, these are some things to keep in mind.

(Refer Slide Time: 25:05)

Feature selection and scaling

- It is important to remove irrelevant features
- When the number of features is too large, and suspected to be highly redundant, feature extraction is required
- If the features are carefully chosen then it is expected that the classification will be better

And, as I mentioned before it is important to remove relevant features and scaling is also an important idea. So, if you choose your features carefully then you would get better classification with kNN.

So, with this we come to an end of this lecture on k-Nearest Neighbors and following this lecture there will be a case study which will use k-Nearest Neighbor that will be taught by one of the teaching assistants and after that I will teach a lecture on k-means clustering. Thank you and I look forward to seeing you again in a future lecture.

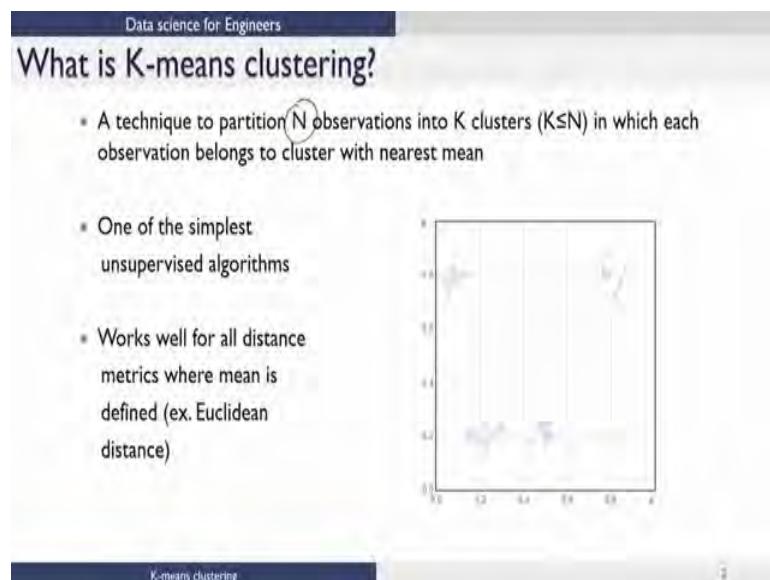
Thanks.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 37**  
**K – means Clustering**

So, we are into the last theory lecture of this course I am going to talk about K-means Clustering today following this lecture there will be a demonstration of this technique on a case study by the TA's of this course.

(Refer Slide Time: 00:53)



Data science for Engineers

## What is K-means clustering?

- \* A technique to partition  $N$  observations into  $K$  clusters ( $K \leq N$ ) in which each observation belongs to cluster with nearest mean
- \* One of the simplest unsupervised algorithms
- \* Works well for all distance metrics where mean is defined (ex. Euclidean distance)

K-means clustering

So, what is K-means clustering? So, K-means clustering is a technique that you can use to cluster or partition a certain number of observations, let us say  $N$  observations in to  $K$  clusters. The number of clusters which is  $K$  is something that you either choose or you can run the algorithm for several case and find what is an optimum number of clusters that this data should be partitioned into.

Just a little bit of semantics I am teaching in clustering here under the heading of classification. In general typical classification algorithms that we see are usually super waste algorithms; in the sense that the data is partitioned into different classes and these labels are generally given. So, these are labeled data points and the classification algorithms job is to actually find decision boundaries between these different classes. So, those are supervised algorithms.

So, an example of that would be the k-nearest neighbor that we saw before where we have label data and when you get a test data you kind of bin it into the most likely class that it belongs to. However, many of the clustering algorithms are unsupervised algorithms in the sense that you have N observations as we mentioned here, but they are not really labeled into different classes. So, you might think of clustering as slightly different from classification, where you are actually just finding out if there are data points which share some common characteristics or attributes.

However, as far as this course is concerned we would still think of this as some form of classification or categorization of data into groups just that we do not know how many groups are there a priori. However, for a clustering technique to be useful once you partition this data into different groups as a second step one would like to look at whether there are certain characteristics that kind of pop out from each of this group to understand and label these individual groups in some way or the other.

So, in that sense we would still think of this as some form of categorization which I could also call as classification into different groups without any supervision. So, having said all of that K-means is one of the simplest unsupervised algorithms, where if you give the number of clusters or number of categories that exist in the data then this algorithm will partition these N observations into these categories. Now, this algorithm works very well for all distance matrix you again need a distance metric as we will see where we can clearly define a mean for the samples.

(Refer Slide Time: 04:32)

Data science for Engineers

## Description of K-means clustering

Given  $N$  observations  $(x_1, x_2, \dots, x_N)$ , K-means clustering will partition  $n$  observations into  $K$  ( $K \leq N$ ) sets  $S = \{S_1, \dots, S_k\}$  so as to minimize the within cluster sum of squares (WCSS)

$$\arg \min_S \sum_{i=1}^k \sum_{x \in S_i} \|x - \mu_i\|^2$$

Where  $\mu_i$  is the mean of points in  $S_i$ .

K-means clustering

So, when we were talking about optimization for data science I told you that you know all kinds of algorithms that you come up with in machine learning there will be some optimization basis for these algorithms. So, let us start by describing what a K-means clustering optimizes or what is objective that is driving the K-means clustering algorithm. So, as we described in the previous slide there are  $N$  observations  $x_1$  to  $x_N$  and we are asking the algorithm to partition this into  $K$  clusters. So, what does it mean when we say we partition it into  $K$  clusters? So, we will generate  $k$  sets  $s_1$  to  $s_k$  and we will say this data belongs to this set and this data belongs to the other set and so on.

So, to give a very simple example let us say you have these observations  $x_1, x_2$  all the way up to  $x_N$  and just for the sake of argument let us take that we are going to partition this data into 2 clusters ok. So, there is going to be one set for one cluster, cluster 1 and there is going to be another set for the other cluster, cluster 2. Now, the job of K-means clustering algorithm would be to put these data points into these two bins ok. So, let us say this could go here this could go here  $x_3$  could go here  $x_N$  could go here maybe an  $x_4$  could go here and so on.

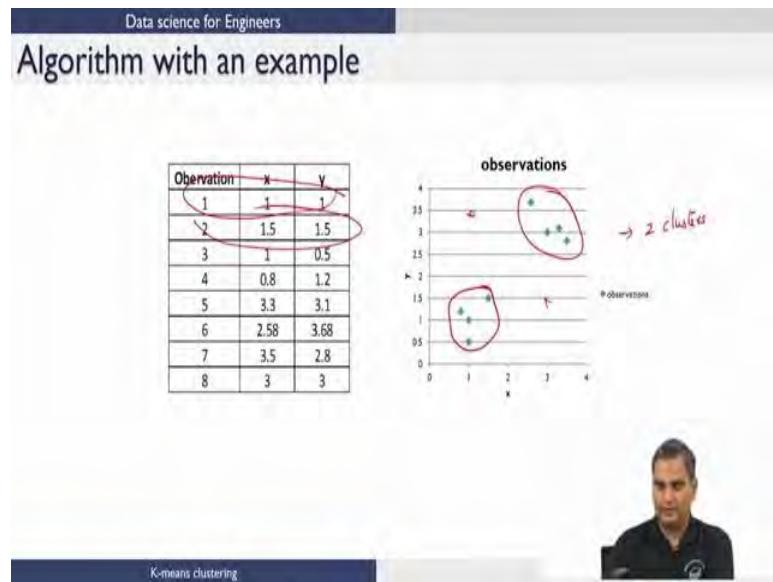
So, all you are doing is you are taking all of these data points and putting them into two bins and while you do it what you are looking for really in a clustering algorithm is to make sure that all the data points that are in this bin have certain characteristics which are like in the sense that if I take two data points here and two data points here, these two

data points will be more like and these two data points will be more like each other, but if I take a data point here and here they will be in some sense unlike each other. So, if you cluster the data this way then it is something that we can use where we could say look all of these data points share certain common characteristics and then we are going to make some judgments based on what those characteristics are.

So, what we really would like to do is the following: we would like to keep these as compact as possible so that like data points are grouped together which would translate to minimizing within cluster sum of square distances. So, I want this to be a compact group. So, a mathematical way of doing this would be the following. So, if I have  $k$  sets into which I am putting all of this in, you take set by set and then make sure that if you calculate a mean for all of this data the difference between the data and the mean squared in a nonsense is as minimum as possible for each cluster and if you have  $k$  clusters you kind of sum all of them together and then say I want a minimum for this whole function. So, this is a basic idea of K-means clustering.

So, there are there is a double summation the first summation is for all the data that has been identified to belong to a set, I will calculate the mean of that set and I will find out a solution for this these means in such a way that this within cluster distance  $x$  which is in the set minus I mean is minimized not only for one cluster, but for all clusters. So, this is how you define this objective. So, this is the objective function that is being optimized and as we have been mentioned mentioning before this  $\mu_i$  is mean of all the points in set  $S_i$ .

(Refer Slide Time: 09:09)



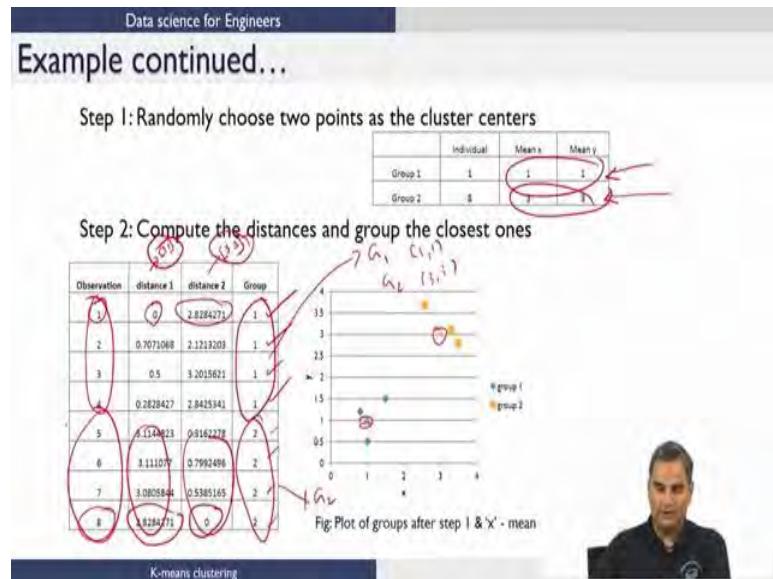
Now, let us look at how an algorithm such as K-means works it is a very simple idea. So, let us again illustrate this with a very simple example. Let us say there are two dimensions that we are interested in x and y and there are eight observations. So, this is one data point, this is another data point. So, there are eight data points and if you simply plot this you would see this and when we talk about cluster. So, notionally you would think that there is a very clear separation of this data into 2 clusters right.

Now, again as I mentioned before in one of the earlier lectures on data science very easy to see this in 2-dimensions, but the minute we increase the number of dimensions and the minute we increase the complexity of the organization of the data which you will see at the end of this lecture itself you will find that finding the number of clusters is really not that obvious. In any case let us say we want to identify or partition this data into different clusters and let us assume for the sake of illustration with this example that we are interested in partitioning this data into 2 clusters.

So, we are interested in partitioning this data into 2 clusters. So, right at the beginning we have no information right. So, we do not know that this all belong to one cluster all of these data points belong to another cluster, we are just saying that are going to be 2 clusters that is it. So, how do we find the 2 clusters because we have no information labels for this and this is an unsupervised algorithm. What we do is we know ultimately there are going to be 2 clusters. So, what we are going to do is, we are going to start off 2

cluster centers in some random location. So, that is the first thing that we are going to do. So, you could start off 2 clusters somewhere here and here or you could actually pick some points in the data itself to start these 2 clusters.

(Refer Slide Time: 11:42)



So, for example let us say we randomly choose two points as cluster centers. So, let us say one point that we have chosen is 1, 1. So, which would be this point here and let us assume this is what is group 1 and let us assume that for group 2 we choose 0.33 which is here. The way we have chosen this if we go back to the previous slide and look at this the two cluster centers have been picked from the data itself. So, the one group was observation 1 and the other group centre was observation 8.

Now, you could do this or like I mentioned before you could pick a point which is not there in the data also and we will see the impact of choosing this cluster centers later in this lecture. Now, that we have two cluster centers then what this algorithm does is the following it finds for every data point in our database this algorithm first finds out the distance of that data point from each one of this cluster centers. So, in this table we have distance 1 which is the distance from the point 1, 1 and we have distance 2 which is the distance from the point 3, 3.

Now, if you notice since the first point from the data itself, ie, the distance of 1, 1 from 1, 1 is 0. So, you see that distance 1 is 0 and distance 2 is the distance of the point 1, 1 from 3, 3. Similarly, since we chose 3, 3 as representative of group 2 if you look at point 8

which was a 3, 3 point the distance of 3, 3 from 3, 3 is 0 and this is the distance of 3, 3 from 1, 1 which will be the same as this. For every other point since they are not either 1, 1 or 3, 3 there will be distances you can calculate.

So, for example, this is a distance of the second point from 1, 1 and this is a distance of the second point from 3, 3 and if you go back to the previous slide the second point is the second point is actually 1.5, 1.5 and so on. So, you will go through here each of these points are different from 1, 1; 3, 3 you will generate these distances. So, for every point you will generate two distance one from 1, 1 other one from 3, 3.

Now, since we want all the points that are like 1, 1 to be in one group and all the points which are like 3, 3 to be in the other group, we use a distance as a metric for likeness. So, if a point is closer to 1, 1 then it is more like 1, 1 than 3, 3 and similarly if a point is close to 3, 3 it is more like 3, 3 than 1, 1. So, we are using a very very simple logic here.

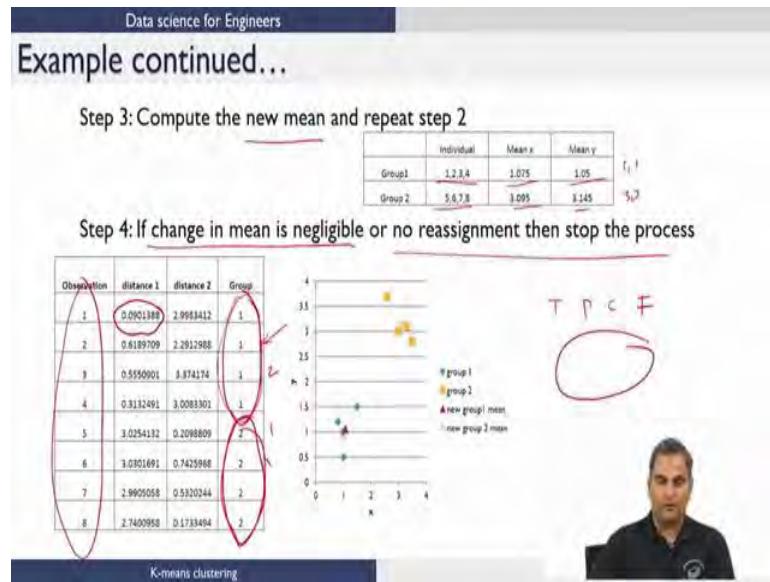
So, you compare these two distances and whichever is a smaller distance you assign that point to that group. So, here distance 1 is less than distance 2. So, this is assigned to group 1 this observation which is basically the point 1, 1 the second observation again if you look at it distance 1 is less than distance 2. So, the second observation is also assigned to group 1 and you will notice through this process a third and fourth will be assigned to group 1 and 5, 6, 7, 8 will be assigned to group 2 because these distances are less than these distances.

So, by starting at some random initial point we have been able to group these data points into two different groups one group which is characterized by all these data points the other group which is characterized by these data points. Now, just as a quick note whether this is in 2-dimensions or N dimensions it is really does not matter because the distance formula simply translates and you could have done that for two classes in N dimensions as easily. So, while visualizing data in N dimensions are hard this calculation whether it is 2 or N dimension it is actually just the same.

Now, what you do is you know that these group positions are the centers of these groups were randomly chosen now, but we have now more information to update the centers because I know all of these data points belong to group 1 and all of these data points belong to group 2. So, a better representation for this group; so, initially the representation for this group was 1, 1, a better representation for this group would be the

mean of all of these four samples and the initial representation for group 2 was 3, 3, but a better representation for group 2 would be the mean of all of these points. So, that is step 3.

(Refer Slide Time: 17:23)



So, we compute the new mean and because group 1 has points 1, 2, 3, 4, we do a mean of those points and the x is 1.075 and y is 1.05. Similarly, for group 2 we do the mean of the labels are the data points 5, 6, 7, 8 and you will see the mean here. So, notice how from 1, 1 and 3, 3 the mean has been updated in this case because we chose a very simple example the updation is only slight. So, these points move a little bit.

Now, you redo the same computation because I still have the same eight observations but now, group 1 is represented not by 1, 1 but by 1.075 and 1.05 and group 2 is represented by 3.095 and 3.145 and not 3, 3. So, for each of these points you can again calculate a distance 1 and distance 2. And, notice previously this distance was 0 because the representative point was 1, 1. Now that the representative point has become 1, 1.075 and 1.05 this distance is no more 0, but it is still a small number.

So, for each of these data points with these new, new means you calculate these distances and again use the same logic to see whether distance 1 is smaller or distance 2 is smaller and depending on that you assign these groups. Now, if we notice that by doing this there was no assignment reassignment that happened. So, whatever were the samples that were

originally in group 1 remain in group 1 and whatever are the samples which are in group 2 remain in group 2.

So, if you again compute a mean because the points have not changed the mean is not going to change. So, even after this the mean will be the same. So, if you keep repeating this process there is no never going to be any reassignment. So, this clustering procedure stops. Now, if it were the case that because of this new mean let us say for example, if this had gone into group 2 and let us say this and this had gone into group 1, then correspondingly you have to collect all the points in group 1 and calculate a new mean collect all the points in group 2 and calculate a new mean and then do this process again.

(Refer Slide Time: 19:50)

- Elbow method – looks at percentage of variance explained as a function of number of clusters
- The point where marginal decrease plateaus is an indicator of the optimal number of clusters
- We will see a demonstration of this in the example

And, you keep doing this process till the mean change is negligible or no reassignment. So, one of these two things happen, then you can stop the process. So, this is a very very simple technique. Now, you notice this technique is very very easily implementable it does not matter the dimensionality of the data you could have 20 variables, 30 variables the procedure remains the same. The only thing is we have to specify how many clusters that are there in the data and also remember that this is a unsupervised learning and so, we originally do not know any labels.

But, at the end of this process at least what we have done is we have categorized this data into classes or groups. Now, if you find something specific about this group by further analysis then you could basically sometimes maybe even be able to label these groups

and the broad categories. If that is not possible at least you know from a distance viewpoint that these two might be different behavior from the system viewpoint. So, from an engineering viewpoint why would something like this be important?

If I let us say give you data for several variables temperatures, pressures, concentrations and so on flow for several variables, then you could run a clustering algorithm purely on this data and then say I find actually that there are two clusters of this data. Then, a natural question and engineer might ask is if the process is stable I would expect all of the data points to be like, but since it seems like there are two distinct groups of data either both or normal, but there is some reason why there is this two distinct group or maybe one is normal and one is not really normal, then you can actually go on probe of these two groups which is normal which is not normal and so on.

So, in that sense an algorithm like this allows you to work with just raw data without any annotation; what I mean by annotation here is without any more information in terms of labeling of the data and then start making some judgments about how the data might be organized. And, you can look at this multi-dimensional data and then look at what is the optimum number of clusters maybe there are five groups in this multi-dimensional data which would be impossible to find out by just looking at an excel sheet.

But, once you do an algorithm like this then it maybe organizes this into 5 or 6 or 7 groups then that allows you to go and probe more into what these groups might mean in terms of process operations and so on. So, it is an important algorithm in that sense. Now, we kept talking about finding the number of clusters, till now I said you let the algorithm know how many clusters you want to look for, but there is something called an elbow method where you look at the results of the clustering for different number of clusters and use a graph to find out what is an optimum number of clusters. So, this is called an elbow method.

And, you will see a demonstration of this in an example. The case study that follows this lecture you will see how this plot looks and how you can make judgments about the optimal number of clusters that you should use. So, basically this approach uses what is called a percentage of variance explained as a function of number of clusters. But those are very typical looking plots and you can look at those and then be able to figure out what is the optimal number of clusters.

(Refer Slide Time: 23:55)

Data science for Engineers

## Disadvantages of K-means

- This algorithm could converge to a local minima, therefore role of initial position is very important
- If the clusters are not spherical, then K-means can fail to identify the correct number of clusters

K-means clustering

So, I explained how in an unsupervised fashion you can actually start making sense out of data and this is a very important idea for engineers because this kind of allows a first level categorization of data just purely based on data and then once that is done then one could bring in their domain knowledge to understand these classes and then see whether there are some judgments that one could make.

Couple of disadvantages of K-means that I would like to mention: the algorithm can be quite sensitive to the initial guess that you use. It is very easy to see why this might be. So, let me show you a very simple example. So, let us say I have data like this. So, if my if I start my cluster points initial cluster points here and here you can very clearly see by the algorithm all these data points will be closer to this.

So, all of this will be assigned to this group and all of these data points are closer to this. So, they will be assigned to this group, then you will calculate a mean and once a mean is calculated there will never be any reassignment possible afterwards then you have clearly these two clusters very well separated. However, if just to make this point supposing I start these two cluster centers for example, one here and one somewhere here then what is going to happen is that when you calculate the distance between this cluster and all of these data points and this cluster center.

And, all of these data points it is going to happen that all these data points are going to be closer to this and all of these data points are going to be closer to this than this point. So,

after the first round of the clustering calculations you will see that the center might not even move right because the mean of this and this might be somewhere in the center. So, this will never move, but the algorithm will say all of these data points belong to this center and this center will never have any data points for it. So, this is a very trivial case, but it just still makes the point that if you use the same algorithm depending on how you start your cluster centers you can get different results.

So, you would like to avoid situations like this and these are actually easy situations to avoid, but when you have multi-dimensional data and lots of data and which you cannot visualize like I showed you here it turns out it is not really that obvious to see how you should initialize. So, there are ways of solving this problem, but that is something to keep in mind. So, every time you run an algorithm if the initial guesses are different you are likely to get at least minor differences in your results.

And, the other important thing to notice look at how I have been plotting this data, I typically I have been plotting to this data, so that you know the clusters are in general spherical. But, let us say if I have data like this where you know all of this belongs to one class and all of this belongs to another class, now K-means clustering can have difficulty with this kind of data simply because if you look at data within this class this point and this point are quite far though they are within the same class whereas, this point and this point might be closer than this point in this point.

So, if in an unsupervised fashion if you ask K-means clustering to work on this depending on where you start and so on, you might get different results. So, for example, if you start with three clusters you might in some instances find three clusters like this. So, though underneath the data is actually organized differently and if these happen to be two different classes in reality, in an unsupervised fashion when you run the K-means clustering algorithm you might say all of this belongs to one class; all of this belongs to another class and all of this belongs to another class and so on.

So, these kinds of issues could be there. Of course, as I said before there are other clustering algorithms which will quite easily handle data that is organized like this, but if you were thinking about K-means then these are some of the things to think about. So, with this we come to the conclusion of the theory part of this course on data science for engineers, there will be one more lecture which will demonstrate the use of K-means on

a case study. With that all the material that we intended to cover would have been covered um. I hope this was an useful course for you we tried to pick these data science techniques, so that you get a good flavor of the things that you think about and also actually techniques that you can use for some problems right away.

But, more importantly our hope has been that this has increased your interest in this field of data science and as you can see that there are several fascinating aspects to think about when one thinks about machine learning algorithms and so on. And, this course would have hopefully given you a good feel for the kind of thinking and aptitude that you might need to follow up on data science and also the mathematical foundations that are going to be quite important as you try to learn more advanced and complicated machine learning techniques either on your own or through courses such as this that are available.

Thank you very much and I hope a large number of you plan to take the exams and get a certification out of this course.

Thanks again.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 38**  
**Logistic Regression (Continued)**

We will continue our lecture on Logistic Regression that we introduced in the last lecture. And, if you recall from the last lecture, we modeled the probability as a sigmoidal function and the sigmoidal function that we used is given here. And notice that this is your hyperplane equation.

(Refer Slide Time: 00:27)

And in  $n$  dimensions this quantity is a scalar because you have  $x$  element,  $n$  elements in  $x$  and  $n$  elements in  $\beta_1$  and this becomes something like  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$ . So, this is a scalar. And, then we saw that if this quantity is a very large negative number then the probability is 0 and if this quantity is a very large positive number the probability is 1 and, the transition of the probability at 0.5.

So, remember I said you have to always look at it from one class's viewpoint. So, let us say if you want class 1 to have high probability and class 0 is a low probability case, then you need to have a threshold that we described before that you could convert this into a binary output by using a threshold. So, if you were to use a threshold of 0.5

because probabilities go from 0 and 1, then you notice that this  $p(x)$  becomes 0.5 exactly when  $\beta_0 + \beta_1 X$  equals 0. This is because  $p(x)$  then equals  $\frac{e^0}{1+e^0}$ , which is equal to 1 by 2.

Also notice another interesting thing that this equation is then the equation of the hyper plane. So, if I had data like this, and data like this and if I draw this line any point on this line is the probability equal to 0.5, a point. That basically says that any point on this line in this 2D case or hyperplane in the n dimensional case will have an equal probability of belonging to either class 0 or class 1 which makes sense from what we are trying to do. So, this model is what is called a Logit Model.

(Refer Slide Time: 02:51)

| $X_1$   | $X_2$ | $X_1$   | $X_2$ | $X_1$     | $X_2$ | Class |
|---------|-------|---------|-------|-----------|-------|-------|
| 1       | 1     | 6       | 3     | 1         | 3     | ?     |
| 2       | 1     | 7       | 3     | 2         | 3     | ?     |
| 3       | 1     | 8       | 3     | 4         | 4     | ?     |
| 4       | 1     | 9       | 3     | 5         | 4     | ?     |
| 5       | 1     | 10      | 3     | 3         | 3     | ?     |
| 1       | 2     | 6       | 4     | 6         | 2     | ?     |
| 2       | 2     | 7       | 4     | 9         | 2     | ?     |
| 3       | 2     | 8       | 4     | 8         | 1     | ?     |
| 4       | 2     | 9       | 4     | 7         | 2     | ?     |
| 5       | 2     | 10      | 4     | 10        | 1     | ?     |
| Class 0 |       | Class 1 |       | Test Data |       |       |

Let us take a very simple example to understand this. So, let us assume that we are given data. So, here we have data for class 0 and data for class 1 and then clearly this is a two-dimensional problem. So, the hyperplane is going to be a line. So, a line will separate this. And, in a typical case in these kinds of classification problems this is actually called as supervised classification problem. We call this a supervised classification problem because all of this data is labeled. So, I already know that all of this data is coming from class 0 and all of this data is coming from class 1. So, in other words I am being supervised in terms of what I should call as class 0 and what I should call as class 1.

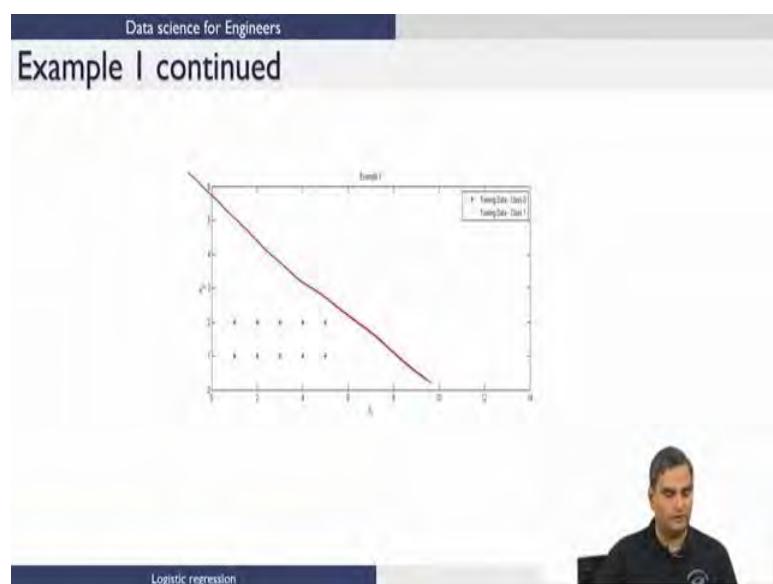
So, in these kinds of problems typically you have this and then you are given a new data which is called the test data. And then the question is what class does this test data belong

to. So, it is either class 0 or class 1 as far as we are concerned in this example. Just to keep in mind that there would be used problems like this. Remember, at the beginning of this course I talked about fraud detection and so on, where you could have lots of records of fraudulent let us say credit card use and all of those instances of fraudulent credit card use you could describe by certain attribute.

So, for example, the time of the day, whether the credit card was done at the place where the person lives, credit card transfer or a credit card use was done at the place the person lives and many other attributes. So, if those are the attributes let us say many attribute are there, and you have lots of records for normal use of credit card and some records for fraudulent use of credit card, then you could build a classifier which given a new set of attributes that is a new transaction that is being initiated could identify what likelihood it is of this transaction being fraudulent. So, that is one other way of thinking about the same problem.

So, nonetheless as far as this example is concerned what we need to do is we have to fill this column with 0s and 1s. If I fill a column with a row with 0 then that means, this data belongs to class 0 and if I fill it one then let us say this belongs to class 1 and so on. So, this is what we are trying to do. We do not know what the classes are.

(Refer Slide Time: 05:31)



So, just so, let me see this it is a very simple problem. We have plotted the same data that was shown in the last table and you would notice that if you wanted a classifier something

like this would do. So, this problem is linearly separable, so it could you could come up with a line that does it. So, let us see what happens if we use logistic regression to solve this problem.

(Refer Slide Time: 06:02)

| Test Results   |                |        |       |
|----------------|----------------|--------|-------|
| X <sub>1</sub> | X <sub>2</sub> | Prob   | Class |
| 1              | 3              | 0.0002 | 0     |
| 2              | 3              | 0.004  | 0     |
| 4              | 4              | 0.999  | 1     |
| 5              | 4              | 0.999  | 1     |
| 3              | 3              | 0.076  | 0     |
| 6              | 2              | 0.0172 | 0     |
| 9              | 2              | 0.991  | 1     |
| 8              | 1              | 0.0002 | 0     |
| 7              | 2              | 0.251  | 0     |
| 10             | 1              | 0.0667 | 0     |

So, if you did a logistic regression solution then in this case it turns out that the parameter values are these. And how did we get these parameter values? These parameters values are guard through the optimization formulation, where one is maximizing log likelihood with  $\beta_0$ ,  $\beta_{11}$  and  $\beta_{12}$  as decision variables. And as we see here there are 3 decision variables because this was a two-dimensional problem; so, one coefficient for each dimension and then one constant.

Now, once you have this then what you do is you have your expression for  $p(x)$  which is as written before the sigmoid. So, this is a sigmoidal function that we have been talking about then whenever you get a test data let us say 1, 3, you plug this into this sigmoidal function and you get a probability. Let us say the first data point when you plug in you get a probability this. So, if we use a threshold of 0.5, then what we are going to say is anything less than 0.5 is going to belong to class 0, and anything greater than 0.5 is going to belong to class 1. So, you will notice that this is 0, class 0, class 1, class 1, class 0, class 0, class 1, class 0, class 0, class 0, right.

So, as I mentioned in the previous slide what we wanted was to fill this column and if you go across row then it says that particular sample belongs to which class. So, now, what we

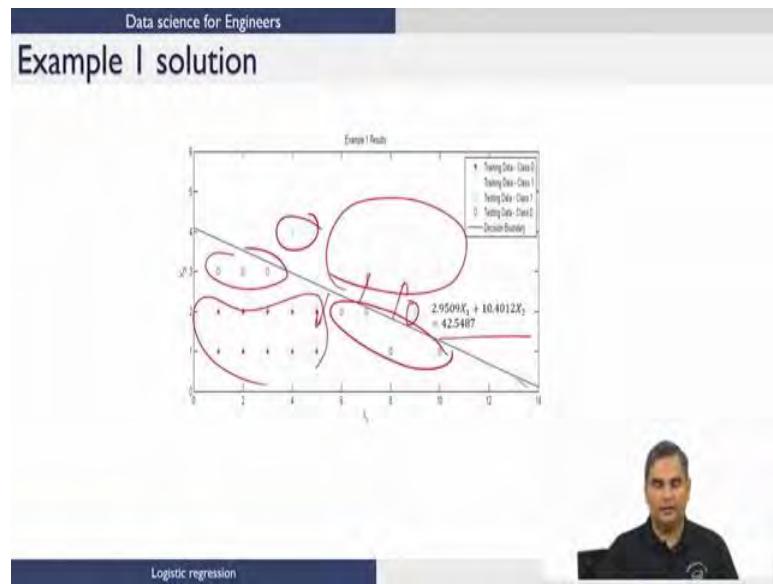
have done is we have classified these test cases which the classifier did not see while you were identifying these parameters. So, the process of identifying these parameters is what is usually called in machine learning algorithms as training. So, you are training the classifier to be able to solve test cases later. On the data that you use while these parameters are being identified are called the training data and this is called the test data, that you are testing a classifier with.

So, typically, what you do is if you have lots of data with class labels already given. One of the good things you know that one should do is to split this into training data and the test data and the reason for splitting this into training and test data is the following. In this case, if you look at it, we built a classifier based on some data and then we tested it on some other data, but we have no way of knowing whether these results are right or wrong, right. So, we just have to take the results as it is.

So, ideally what you would like to do is you would like to use some portion of the data to build a classifier and then you want to retain some portion of the data for testing and the reason for retaining this is because the labels are already known in this. So, if I just give this portion of the data to the classifier the classifier will come up with some classification, now that can be compared with the already established labels for those data points, right. So, from verifying how good your classifier is, it is always a good idea to split this into training and testing data; what proportion of data you use for training, what proportion of data use for testing, and so on are things to think about.

Also, there are many different ways of doing this validation as one would call it with this data; there are techniques such as k fold validation and so on. So, there are many ways of splitting the data into train and test and then verifying how good your classifier is. Nonetheless, the most important idea to remember is that one should always look at data and partition the data into training and testing, so that you get results that are consistent.

(Refer Slide Time: 10:21)



So, if one were to draw these points again that that we use this in this exercise. So, these are all class 1 data points, these are class 0 data points and this is your hyperplane that a logistic regression model figured out and these are the test points that we tried with this classifier. So, you can see that in this case everything seems to be working well, but as I said before you can look at results like this in two dimensions quite easily.

However, when there are multiple dimensions it is very difficult to visualize where the data point lies and so on. Nonetheless so, it gives you an idea of what logistics regression doing. It is actually doing a linear classification here. However, based on the distance in some sense from this hyperplane we also assign a probability for the data being in a particular class.

(Refer Slide Time: 11:37)

Data science for Engineers

## Regularization

- General objective

$$\min_{\theta} -L(\theta)$$

where  $L(\theta) = \left( \sum_{i=1}^n y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i)) \right)$

- When large number of independent variables are present, logistic regression tends to over-fit
- To prevent over-fitting, we need to penalize the coefficients
- This is known as regularization

$\beta_0 + \beta_1 x_1 + \beta_2 x_2$  (with a red circle around the sum of coefficients)

Logistic regression

Now, there is one more idea that we want to talk about in logistic regression. This idea is what is called as regularization. The idea here is the following. If you notice the objective function that we used in the general logistics regression which is what we call as a log likelihood objective function. Here theta again speaks to the constants in the hyperplane are the decision variables and this is the form of the equation that we saw in the previous lecture and in the beginning of this lecture also I believe. Now, if you have n variables in your problem or n features or n attributes then the number of decision variables that you are identifying are n + 1.

So, one constant for each variable and the constant if this n becomes very large then there are large number of variables that represent, but then what happens is this logistic regression models can over fit because there are so many parameters that you could tend to over fit the data. So, to prove this what we want to do is somehow we want to say though you have this n + 1 decision variables to use, one would want these decision variables to be used sparingly. So, whenever you use a coefficient for a variable for the classification problem then we want ensure that you get the maximum value for using that variable in the classification problem.

So, in other words, if let us say there are two variables I say  $\beta_0 + \beta_{11}x_1 + \beta_{12}x_2$ , then for this classifier I am using both let us say variables  $x_1$  and  $x_2$  as being important. What one would like to do is make sure that I use these only if they really contribute to the

solution or to the efficacy of the solution. So, one might say that for every term that you use you should get something in return or in other words if you use a term and get nothing in return I want to penalize this term. So, I want to penalize these coefficients. This is what is typically called as regularization.

(Refer Slide Time: 14:20)

**Regularization**

- Regularization helps in building non-complex models that avoids capturing noise in model due to over-fitting
- The objective now becomes  $\min_{\theta} -L(\theta) + \lambda * h(\theta)$  where  $\lambda$  is regularization parameter and  $h(\theta)$  is regularization function
- Depending on  $h(\theta)$ , the regularization can be classified as  $L_1$  or  $L_2$  type
- $h(\theta) = \theta^T \theta$  for  $L_2$  type regularization
- Larger the value of  $\lambda$ , more is the regularization strength
- Regularization helps the model work better on test data due to the fact that over-fitting is minimized on training data

Logistic regression

So, regularization avoids building complex models or it helps in building non-complex models so, that your fitting effects can be reduced. So, how do we penalize this? So, notice that what we are trying to do is we are trying to minimize a log likelihood. So, what we do here is we add another term to the objective and lambda is called the regularization parameter and this  $h$  theta is some regularization function.

So, what we want to do is when I choose the values of theta to be very large, I want this function to be large, so that the penalty is more or whenever I choose a variable, right away a penalty kicks in. And this penalty should be offset by the improvement I have in this term of the objective function, so that is the basic idea behind regularization.

Now, this function could be of many types if you use this function to be  $\theta^T \theta$  then this is called  $L_2$  type regularization. So, in the previous example this will turn out to be  $\theta$  would be  $[\beta_0 \beta_{11} \beta_{12}]$ , transpose times  $[\beta_0 \beta_{11} \beta_{12}]$ . So, in this case  $h$  of  $\theta$  will become  $\beta_0^2 + \beta_{11}^2 + \beta_{12}^2$ .

Now, there are other types of regularization that you can use. You can use this is what is called the L 2 type or L 2 norm. You can also use something called an L 1 type or an L 1 norm. And larger the value of this coefficient that is multiplying this and the more is regularization strength that is you are penalizing for use of variables lot more. And, one general rule is regularization helps the model work better with test data because you avoid over fitting on the trained data, so that is in general something that one can keep in mind, as one does these kinds of problems.

So, with this the portion on logistic regression comes to an end. What we are going to do next is we are going to show you an example case study, where logistics regression is used for a solution. However, before we do this case study; since all the case studies on classification and clustering, will involve looking at the output from the arc code. I am going to take a typical output from the arc code and there are several results that will show up, these are called performance measures of a classifier.

I am going to describe what these performance measures are and how you should interpret these performance measures once you use a particular technique for any case study. So, in the next lecture we will talk about these performance measures. And then following that will be the lecture on a case study using logistic regression, right.

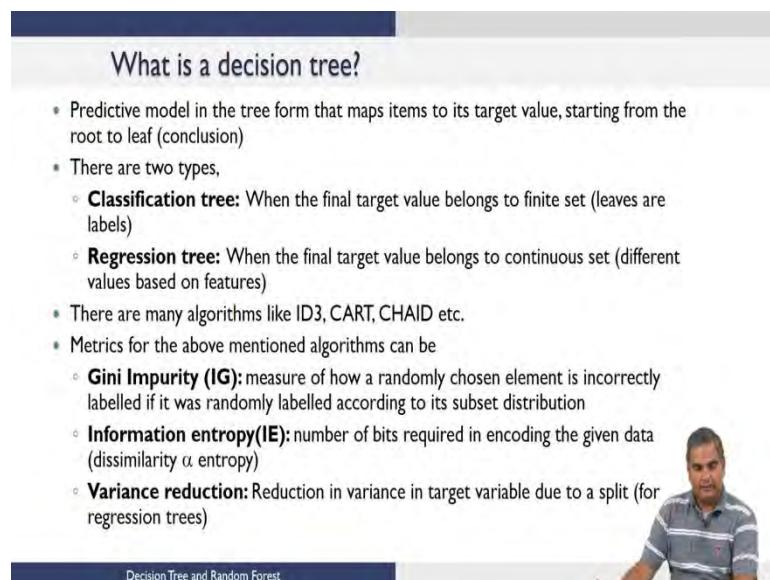
So, thank you for listening to this lecture. And, I will see you in the next lecture.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 39**

I am going to teach couple of machine learning techniques called Decision Trees and random for us in this lecture.

(Refer Slide Time: 00:25)



What is a decision tree?

- Predictive model in the tree form that maps items to its target value, starting from the root to leaf (conclusion)
- There are two types,
  - **Classification tree:** When the final target value belongs to finite set (leaves are labels)
  - **Regression tree:** When the final target value belongs to continuous set (different values based on features)
- There are many algorithms like ID3, CART, CHAID etc.
- Metrics for the above mentioned algorithms can be
  - **Gini Impurity (IG):** measure of how a randomly chosen element is incorrectly labelled if it was randomly labelled according to its subset distribution
  - **Information entropy(IE):** number of bits required in encoding the given data (dissimilarity  $\alpha$  entropy)
  - **Variance reduction:** Reduction in variance in target variable due to a split (for regression trees)

We will start with a decision tree. So, decision tree as the name suggest is a tree like structure that you generate from data to do some machine learning tasks. And, as we have seen before, the most common machine learning tasks are either classification or regression or function approximation.

So, you can generate these decision trees for both these types of problems, both classification problems and regression problems. Just to recap for people who have not seen this many times. A classification problem is one, where when you are given a data you are trying to classify it to one of the predefined classes that you have. And, regression problem is one where given a data you are trying to predict an output or target feature value. Now, you could build a decision tree for either a classification problem our regression problem.

And there are multiple algorithms for building these trees, which are all mentioned here. But, what I would like to mention is at the end of it what you are going to get is like a tree structure which is going to let you either predict a value for a target or provide a classification. The reason why I mentioned many algorithms is that when you use a particular software or a package these would become options that you can choose. And there are multiple metrics that are used in building this decision tree; and I will demonstrate how one metric works in this lecture and you can see that the other metrics will work similarly.

So, once you understand one of these you will be able to read on your own and understand the other metrics and how they might be used. So, these metrics basically, we will talk about them in more detail as we go along. These metrics basically allow you to unravel the tree so, to speak. So, if you think about a decision tree as a tree, with some top node and then there are multiple nodes that are being unraveled and formed; there must be some logical procedure for this unraveling and making this tree from data.

And, there could be multiple matrix and Gini impurity information entropy or variance reduction or metrics that are used. Typically Gini impurity and information entropy are used when you have classification problems. And variance reduction is a metric that is used, when you have regression problems that you are trying to model. So, in this lecture I will focus on classification problems and Gini impurity isometric. But what you need to see through this lecture is how this tree is formed? And once you understand that it will be very easy to see how you can develop a tree like that, for regression problems.

(Refer Slide Time: 03:41)

### Describing decision trees through an example

- Consider the following classification problem
- Discriminate between three different species of Iris flower
- The training data contains 49-setosa, 50-versicolor and 50- virginica species
- The features that are available are sepal length, sepal width, petal length and petal width
- The ranges for these feature values are

|     | setosa    | versicolor | virginica |
|-----|-----------|------------|-----------|
| S.L | [4.3,5.8] | [4.9,7]    | [4.9,7.9] |
| S.W | [2.3,4.4] | [2,3.4]    | [2.2,3.8] |
| P.L | [1,1.9]   | [3,5.1]    | [4.5,6.9] |
| P.W | [0.1,0.6] | [1,1.8]    | [1.4,2.5] |

Decision Tree and Random Forest



So, the way I am going to teach this development of a decision tree or understanding decision trees is through an example. So, this will make it a lot more concrete in terms of your understanding in terms of what the key ideas are in these decision trees ok. So, let us start with a classification problem as I mentioned in previous slide, I am going to focus on a classification problem. So, let us assume that we have this problem this is a very very well known problem.

If you just search for this you will see this being described in multiple papers it used to be a data set that many people tried the algorithms on for has been for a while. Now so, the idea is there is this iris flower and there are multiple species of this, given a new Iris flower would you be able to classify it as one of the three species that we are considering here. So, the flower type is iris and the species are setosa, versicolor or virginica species. So now, when we say that given as a flower would you be able to classify it into one of these categories, we should have some training data to do this.

So, let us assume in this case for me to explain decision trees to you. Let us assume that there are above 149 data points and; that means, I have 149 samples of this flower which are pre classified as being setosa, versicolor or virginica species. Now when we say data what does that mean? So, for each sample what we might do is we might compute certain characteristics of that sample. In this data set there are four features there are computed for each sample. So, one is called the sepal length, the other one is sepal width petal length

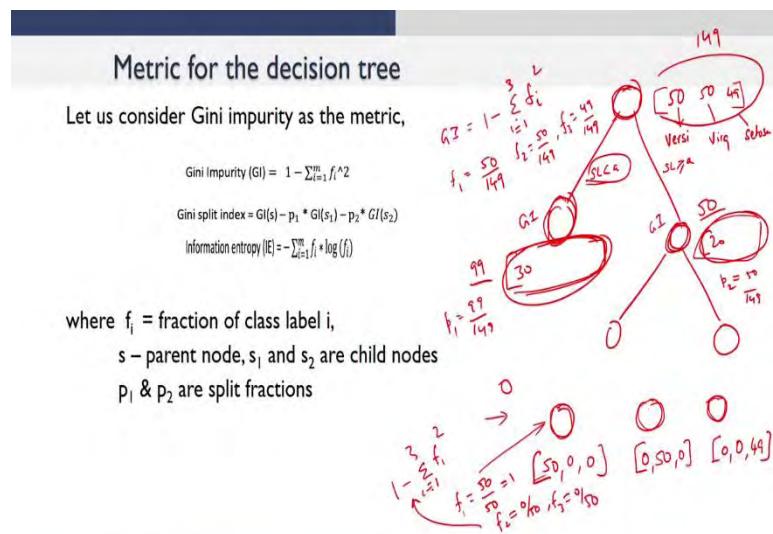
and petal width. So, these are actual geometric features that you compute for the sample. And then let us say you have this 49 already I characterized setosa you take each sample measure this and then put it into data and then say this is from setosa and so on.

Now, the problem is I have to come up with a classifier, which will learn how to classify the data when I have a new data point. Based on already pre-classified data that is used to train my algorithm; and the algorithm that we are going to look at in this case is a decision tree because which is what we are going to use in the example that we are going to look at. Now, the ranges for these feature values are several length if its a setosa flower, we notice that it is between 4.3 and 5.8.

If its versicolor 4.9 and 7; virginica 4.9 and 7.9. This is basic inspection of data and then seeing what are the ranges of these values for each of these attributes for each of these classes. So, this is one class this is another class another class; and the attributes are distributed in these ranges for this data. So, this is basic explanation of the problem that we are trying to solve. Now, you will notice as I described decision tree starting with this problem we will see that just from this data, we are going to generate a tree which is going to allow us to classify new data points.

So, the key things that I want you to notice as we go through this example or how this tree is generated at the end of it. If a new data point is given how would that tree help us classify into one of these three classes is the important thing that I would like you to notice.

(Refer Slide Time: 07:23)



So, let me explain decision trees through this example of this iris classification. So, decision tree as a name suggests is tree like structure where you have nodes which open out into other nodes which might open into other nodes and so on. And, you start with one node and then you come to let us say certain number of nodes at the end.

And, how do you generate this tree and how it is useful for classification is what we are going to see in this lecture. So, to understand this easily let us think about, what each node means? So, what each node means is that each node has a collection of data points. So, if I start with the root node or the very first node, this node is going to have all the data points that are there in the problem.

So, in this case we know that there are 50 data points from versicolor; 50 data points from virginica; and 49 data points of setosa. Now if we did not build this tree at all and we just sat with this data; and then I am going to say let me do a classification whenever you give me a new data point; then the best way to do this would be to look at this data. And then see which class is the most occurring class in this data point under the assumption that the global data set will also has similar proportions.

So, whatever is occurring the most here is the most likely species type in terms of the total data set. Then we will say it has to be either versicolor or virginica. And if you are forced to give one decision you might randomly break the tie and then say a new data point is either versicolor or virginica right. So, if you were to just sit at the root node and then do this is the solution that you will come up with.

But this is not useful from a classification viewpoint what we are trying to do is, we are trying to use the features of these samples. And, then do some computations so that we get a good classification with a high accuracy is what we are looking at right. Because, if I just kept saying versicolor as the class without doing any more development of this decision tree. Then in the 149 sample points I will get 50 of them right because every time I am just going to say versicolor.

So, 50 I will get right, but 99 times I will get it wrong. So, that is very poor accuracy which is what you do not want. So, basically the idea is to somehow develop this tree. So, that whichever node I stopped and give an answer the accuracy improves. So, that is basic idea of this decision tree. The same notion also works with regression trees, if you had one

target value at the first node if I if you ask you what is likely to be the target value for a new data point.

If you do not want to do anything at all you might simply take the average of all the training data points and say that is a likely value for any new data point right, but that will give you a very poor model. So, in that case also you will try to develop this tree expand this tree so that your regression problem you can solve much better. So, the same idea works for both classification and regression problem is what I wanted to tell you.

Now we are going to learn how we are going to develop this tree starting with just this data node the first node. And these are concepts that we are going to introduce for us to explain to you how this tree is developed? But before we do that to understand this better, let us look at what might be the best case scenario for this example. If I were to develop a tree supposing I were to develop this tree at the end of it let us say I get three nodes.

And, this node has data of all versicolor and virginica is 0 this is 0 and this node has all data corresponding to 0 versicolor 50 virginica 0 setosa; and this has data corresponding to this 0, 0, 49. If this is the case then what happens is, as you start from here and go through these nodes we will explain what going through this nodes mean. As you start from the base node and use the features of the new data that has been given the feature values and you traverse this tree. And if you end up here, then you will say the new data belongs to versicolor sample. If you end up here you will say it belongs to virginica and if you end up here you will say it belongs to setosa.

Now, the question is how do you develop the tree? So, that this partition happens and what does traversing this tree mean. So, how do I start from here and decide whether I should go to this node or this node is basically what I am going to teach and that is the basic idea of decision tree as an algorithm itself. Now, remember that you might not always be able to classify this data into such distinct nodes, there might be some overlap which you might simply not be able to get rid of and all of this depends on the data.

So, in some data sets you can get a complete separation; and in some data set whatever you do you might not get complete separation. And actually you know from your machine learning knowledge that getting complete separation and training said by itself might not be very helpful, because basically we are over learning the training set. So, the ability to generalize when I get a new data point might be much less if this happens ok. Now, that

we have set up this basic tree structure and then explain to you the ideas of how we are going to use this tree to make the classification.

Now, it is easy for me to describe how this tree itself is generated purely from data; and how these algorithms work to develop this tree. A tree that is developed like this from data is called a decision tree and it is called a decision tree because at every node based on the feature value we make a decision traverse the tree till you stop at one point which you where you give the final decision in terms of what class this data belongs to ok. So, it basically mimics how we take decisions if this is this and that is that then do this and so on. The same notion is captured here in the tree ok.

So, now to develop this tree we are going to introduce some basic terminology as I mentioned before, I am going to describe this notion of Gini impurity. So, if you notice here Gini impurity for every node so, you can you can define a Gini impurity for every node in the tree. And, for every node in the tree Gini impurity is different using this equation. Let us try and understand what this equation means. So, to understand that equation, let us take this node here and then we will come back to one of these nodes and then kind of contrast what happens.

So, if I take a Gini index for this node, this formula basically says Gini index =  $1 - \sum_{i=1}^3 f_i^2$ . So, basically we have to define what  $f$  is. So, there are three classes, each  $f$  basically computes a fraction of samples from a particular class in the total set. So, for example, if we compute a fraction for versicolor  $f_1$ ; the number of samples in this data set of versicolor is 50, the total number of samples is 149. So,  $f_1$  will be 50 by 149 and  $f_2$  will also be 50 by 149. And  $f_3$  will be 49 by 149.

So, once we have these three numbers we can compute a Gini impurity for this node as  $1 - \left(\frac{50}{149}\right)^2 - \left(\frac{50}{149}\right)^2 - \left(\frac{49}{149}\right)^2$ . So, that is how you compute the Gini impurity of this node ok. Similarly, once we get to each of these nodes based on the number of data points of these classes in the total data we can compute Gini impurity. And, you will notice and in this example it will be nicely seen as we go through this. As we go to different nodes because your partitioning this data this Gini impurity will keep changing for each one of these nodes.

Now, if you are defining Gini impurity in a decision tree for each of these nodes we might also ask this question as to what is a best node right in this decision tree? So, if you go back and look at these nodes so, for example, let us try and compute a Gini impurity for this node. Why am I picking this node this node is what I am going to call as pure node because if you look at this data point. If I come to this node I can categorically say the sample belongs to versicolor right.

If I somehow land up in this node, then I can say the sample belongs to virginica. And if I land up here I will say the sample belongs to setosa right. So, these are what are called pure nodes; that means, these nodes have some more collected data, corresponding to only very specific classes where the other classes are excluded in the data. So, if I have a node like this a pure node a how would I compute the Gini index I use the same formula?

But what will that value be we can see this here. In this case again I have  $1 - \sum_{i=1}^3 f_i^2$ . In this case now,  $f_1$  for versicolor is 50 divided by 50 because  $50 + 0 + 0$ . There are only 50 samples this is equal to 1,  $f_2$  will be 0 by 50 and  $f_3$  will be 0 by 50 ok. So, the fractions will be 1 0 0; and once you use this and then calculate the Gini impurity index. So, the Gini impurity will be 0 ok. So, whenever the Gini impurity is 0; that means, we have got pure sets were only one class is represented and the other classes are all left out. So, ideally then the goal of the whole decision tree is to start with the data itself which gives me a Gini impurity based on doing nothing with this data.

And then somehow unravel this tree and get two nodes at the bottom, which are all pure are as close to pure as possible right. So, here the top of this decision tree there will be a positive value for this Gini impurity. And our goal is to come down to the lowest level of the tree where all the nodes have Gini impurity of 0 or pure nodes. So, in other words what we are trying to do is we are trying to keep reducing this Gini impurity as we go down this tree to get to 0 Gini impurity so that is what we are trying to do.

So, now we come to this question of how do we decide how to traverse this tree? So, then we ask this question what does it mean when we say I want to traverse this tree. So, basically as I said before each one of this is a decision. So, at this point I have to take a decision and the way I take a decision here is I pick one feature from the data set and then do some computation with that feature. And, the result of this computation allows me to go either here or here.

So, I have one example might be take one feature and if that feature is greater than 5; go to this node and if the feature is less than 5 go to this node might be one way to do this. So, in other words whenever we are trying to traverse or unravel this tree, what we are looking at is we are looking at a future value and making some decisions based on that future value. So, this whole tree itself is developed like that and each one of these decision points will have a feature and something that you compare it with, so that you can develop this decision tree.

Now, as you notice even in this case there are four features right petal width, petal length, sepal width and sepal length. So, you could choose any one of these four features for opening out this tree. And each of these features have different ranges of values as we saw in the previous slide. So, there has to be some partition point for those values so, that we make a decision to go to one or the other part of the tree. So, how do we do that ok, that is what we are going to see.

So, once we have this Gini impurity, we are also going to discuss something called Gini split index. So, let us assume I choose some feature and then decide to split this data ok. Now, what does it mean, when I say one feature and somehow I am going to make this decision to split the data we will see in the next slide? But for now bear with me I am just think about this supposing I say I am going to pick one feature in this case you know sepal length; and then I am going to say if sepal length is less than some value  $a$ ; go here and if it is greater than equal to some value  $a$ , go here right that is something that I can make a decision.

So, why should I only choose sepal length not sepal width why petal length petal width? And so on are questions that we are going to answer, but just to explain this whole process here I am explaining this ok. Then if this is how I am going to unravel this tree, then what I do is of all the samples here I find the samples that would satisfy this condition and then put them here ok; maybe of the 50 samples we will see the actual values I am just explaining this. So, maybe of this 50 samples you know if 30 of these samples are such that sepal length is less than  $a$ ; then the remaining 20 will go here.

So, this way you split the whole sample of 149 data points into some number. Now it might be just that this has maybe 99 data points come here 50 come here it all depends on how what we choose here. Maybe 60 of these data points come here and the remaining go there

and so on ok. So, now, you have this because we have already defined the Gini impurity for each node based on this we can actually compute a Gini impurity for this node. And based on what comes here we can compute a Gini impurity for this node.

So, now if we use this sepal length and this number a; as the choices to make this split; then we can compute something called Gini split index which is basically the difference between the Gini index of this original node minus  $p_1$  and  $p_2$  will come back to we can compute a Gini index for this node same formula based on how this split happened. And we can compute a Gini index for this node based on how split this split happened.

The only thing that we need to understand at this formula is what are  $p_1$  and  $p_2$  that is very simple. If I start with 149 data points, let us say 99 data points come here and 50 data points go here.  $p_1$  is the fraction of data points that came to this node which will be 99 by 149. And  $p_2$  will be the fraction of data points that went to this node which will be 50 by 149. So, we can compute this fraction and we can also compute the Gini index for the each of these once we have that we can compute this Gini split index ok. Now what this value is will depend very critically on what feature we chose here to unravel this. And also what is the value of the feature that we chose to make that partition.

Now, the whole notion of decision tree is how do you come up with the sequence of features so, that you unravel the tree and the values that you use to partition this all of those are automated. And these algorithms these packages will give you the best solution for these splits and so on. You do not have to do it, but I am explaining this, so that you understand what basically happens when you finally, see a result for a decision tree example that you might work with.

(Refer Slide Time: 24:39)

### Important rules for constructing trees

- Every parent node of higher Gini impurity / information entropy is split based on features in order to lower its Gini impurity (or information entropy or variance reduction in the case of regression trees)

Gini impurity of pure sets = 0

- The split which corresponds to higher Gini split index is always preferred.

That is if split index 1 = 0.5 and split index 2 = 0.25, then split corresponding to split index 1 will be chosen.

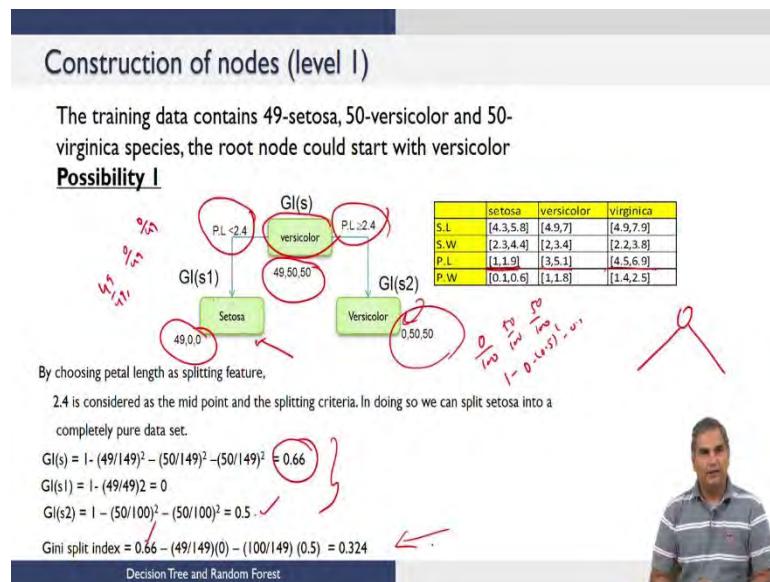
So, basically the important rules for constructing this is so, every parent node which will have a higher Gini impurity remember I said at the top is where I have not done any splitting. And ideally what I am looking for is the bottom level nodes which are pure so; that means, for then the Gini impurity is 0 there they are pure sets. So, basically Gini impurity keeps coming down as I go down the tree. And there are multiple options for doing this splitting.

So, what we do is, whenever there are multiple options we could enumerate all of those options or we could use some smart algorithm to pick options that are likely to be very good. And let us say I have multiple options, I will compute the Gini split index for each of these options and then I will always choose the Gini split index which is maximum right. Now, why do we want to choose a Gini split index which is maximum that comes from your previous equation here.

Now, if I am having a node with a certain Gini impurity and I want to make children nodes of those what I want to do is, I want to get to these pure nodes as quickly as possible right, only then I can do perfect classification. So, ideally if I could split in such a way that the two children node that come out of this main node, have Gini impurity of 0 right that would be the best solution. In which case, the Gini split index will be the Gini impurity value of the original node because this will be 0 this will be 0 right, so that is the best solution.

So, actually the split index should be as high as possible which basically means that these values are as low as possible. So, this has low impurity or there as close to pure sets are possible. So, when I have multiple possibilities one thing I can do is I can you enumerate these possibilities; and then for each one of these possibilities I compute a Gini split index. And, then I find out the possibility for which the Gini split index is a maximum and then I say this is the choice I am going to make ok.

(Refer Slide Time: 26:57)



So, that is what is seen here. So, let us look at this now with this example now that I have explained all of this, you will understand this hopefully much better. So, the root node as we showed before in this example is with this number of data points ok. So, we do 49, 50, 50 in this case for 49 setosa 50 versicolor and you know 50 virginica. So, if you know look at this node here you see these numbers of data points ok; and also you see this node has something. So, which says versicolor that basically means if you are sitting in the node for a data point and I asked you what do you classify the sampler.

So, you are going to say I am going to classify the samplers versicolor; that means, at the beginning if I do not do anything at all any new sample you are giving me I am just going to close my I sense and say its over versicolor. And why am I saying its versicolor I could have said versicolor are virginica because there are 50 data points I have just randomly broken this tie and then said its for versicolor ok.

So, this now you see in this node the none of the data is lost ok. So, all the data is still there here right, 49 setosa 50 versicolor and 50 virginica. Now, remember in the tree thing I said the first decision we have to take. So, the decision means I have to choose one of the features there are four different possibilities. What the algorithms do is that, they look at all these multiple possibilities and find the best split somehow to come up with the most compact tree that you can build. But here in this lecture I am trying to explain the ideas behind it. So, I am going to take a couple of examples to show you what happens.

So, for example: if the algorithm had actually chosen the petal length as basically the feature of choice here. Now, you look at petal length and then. So, you see this here. So, if its setosa the petal length range is 1 to 1.9 versicolor its 3.5 to 1 and this is 3 to 5.1 sorry. And this range is 4.5 to 6.9. Now, if you notice all of this are automatically done by the algorithm here I am just trying to explain this with this data so that you understand there is this logic behind how I break this and so on.

And, once you look at a tree you will be able to see what is actually happening here. Now if you look at this range 3 to 5.1, 4.5 to 6.9 you will see that there is an overlap right. So, any data value that I pick which partitions this, there is likely to be both versicolor and virginica and the partition data. But if you look at this here there is a clean partition because for setosa, petal length is between 1 and 1.9. For versicolor is 3 and 5.1 for virginica its 4.5 and 6.9.

So, if you take any value between 1.9 and 3 ok, you would be able to separate out setosa from the other species right. So, one value you can take is roughly in the middle of 1.9 and 3.5 so that you get good classification. So, you might say if petal length is less than 2.4 go to this node and if petal length is greater than 2.4 go to this node right. If that is the decision that you are making then let us see how the data will get partitioned right.

So, in the training data if you pick all the data points where petal length is less than 2.4 and bring it to this node, you will notice all the setosa data will come here to this node. And all the versicolor and virginica data will go to this node ok. So, remember this 149 data points now has been split it into two nodes. One node has retained 49 data points and the other node has retained 100 data points. Now you notice that this 49 data point node is a pure node whereas, the 100 data point node is not a pure node.

Nonetheless, if you make this decision that I am going to choose petal length as the first feature based on which I am going to separate this and 2.4 as the number then you will get this. And, now you can quite easily compute the Gini index for this which is sorry Gini impurity for this which is  $1 - \left(\frac{49}{149}\right)^2 - \left(\frac{50}{149}\right)^2 - \left(\frac{50}{149}\right)^2$  which is what I had mentioned, this will turn out to be 0.66. Now since this is a pure node, you will get the value to be 0 here because at the three fractions or 49 by 49, 0 by 49 and 0 by 49 so, you will get a value of 0 here. And, if you look at this here you will get a value of 0 by 100, 50 by 100 and 50 by 100 as the three fractions.

So, I will have 1 minus 0 minus 0.5 square minus 0.5 square which is what is shown here its 0.5. So, I have a Gini impurity for this I have Gini impurity for this I have a Gini impurity for this. Now, for this option of petal length and value of 2.4, if I were to compute a Gini split index remember Gini split inductors in indexes the Gini impurity value of this minus whatever fraction of data came here times the Gini impurity of this minus whatever fraction of data came here times the Gini impurity of this. We have already computed the Gini impurity of these three nodes here. So, the Gini split index is the original nodes Gini impurity minus 49 points came here.

So, 49 by 149 and the Gini impurity is 0 times 0 minus 100 points came here. So, 100 by 149 times the Gini impurity of this which is 0.5. So, if we compute this you get 0.324 ok. So, please look at this computation and if you understand this computation every other time the same computation is done there is no difference at all. So, all you need to know is that every node has a Gini impurity and then based on this partition you can compute a Gini split index.

(Refer Slide Time: 33:39)

### Construction of node (level 1) contd.

What happens if we split based on sepal length,

**Possibility 2**

| S.L | setosa      | versicolor | virginica   |
|-----|-------------|------------|-------------|
| S.L | [4.3, 5.8]  | [4.9, 7]   | [4.9, 7.9]  |
| S.W | [2, 3, 4.4] | [2, 3, 4]  | [2, 3, 3.8] |
| P.L | [1, 1.9]    | [3, 5, 1]  | [4.5, 6, 9] |
| P.W | [0, 1, 0.6] | [1, 1.8]   | [1, 4, 2.5] |

By choosing sepal length as splitting feature,  
It is observed that range values overlap, so in this case consider the edge value that corresponds to high split index. Therefore split value can be 5.8 or 4.9  
 $G(s) = 1 - (49/149)^2 - (50/149)^2 - (50/149)^2 = 0.66$   
 $G(s1) = 1 - (49/73)^2 - (21/73)^2 - (3/73)^2 = 0.4650$   
 $G(s2) = 1 - (29/76)^2 - (47/76)^2 = 0.4720$   
 $Gini \text{ split index} = 0.66 - (73/149)(0.465) - (76/149)(0.472) = 0.1915$

If split value is  $S.L = 4.9$ , the split index = 0.0746

Decision Tree and Random Forest

So, if I were to do this and said instead of petal length let me use sepal length ok; and then I decide sepal length is what I use. And then let us say I come up with this value of 5.8, if you asked me how did you come up with this 5.8 there are multiple heuristic in these things. And, you can use a very simple heuristic here to come up with 5.8. Now, the reason why I do not want to go too much into this heuristic is there are multiple possibilities and these algorithms automatically figure out what are good possibility.

So, we really do not need to worry about how this 5.8 comes about, but you want to vary think about how once I have a final solution how I understand that solution. So, let us say somehow I have come up with this 5.8 as the as the best split value here. So, basically I say sepal length is 5.8 remember this is again the top node, which is whatever we had before without any partition. Now, what I do is I partition data such that all the data which have sepal length less than 5.8, I bring to this node and all the data which have sepal length greater than 5.8 I bring to this node.

And it might turn out when I do this that 49 samples of setosa comes here and 21 from the next one and 3 from the next one. And notice what I do here ok, I basically have this node saying setosa. And why does it say setosa? Because if you look at this data point the maximum number of class from which this data comes is setosa right. So, if I land up here after at the end of my decision tree process then I will say that setosa that sample is most

likely to be setosa right. And if you look at this here this is virginica that is because 0 of setosa 29 of versicolor and 47 of virginica.

So, the maximum number of samples come from the virginica species. So, I will say the sample that I have is a virginica sample. Now, I do not want to go through these computations, if you do the same computations as I showed before you will get a split index value of 0.1915. And, you will notice the previous value was different from this. The key things that I want to notice look at the mechanics of this very simple all that is happening is your original data set is being split into multiple data sets right.

So, in the first case the original data set was split into two data set in a particular fashion and in the second choice the same data set is split into two data sets which have different from what I got for the first data set right. So, the way the decision tree works is actually splitting this data set into multiple nodes. And in each node depending on whatever class is preponderant in the data I am going to classify that that is it right. So, if I have more nodes in the tree if I start from here now what will happen is this data set will be split more and this data set will be split more.

So, you can think of this original data as being whittle down into smaller and smaller data sets in each of these nodes and the way that smaller and smaller data sets are developed is through these choices that we make. So, in this case we set sepal length less than 5.8 then we go to the training data set. And, say all instances where sepal length was less than 5.8 I combine into this node and greater than 5.8, I combine into the other node.

(Refer Slide Time: 37:23)

### Construction of nodes (level 2)

Considering all such possible splitting, for level 1, we identify that splitting according to P.L is the best

149 → 49

Level 2

P.W < 1.8      P.W. ≥ 1.8

G(s1)      G(s2)

versicolor      virginica

54      46

Possibility I

By choosing petal width as splitting feature,  
It is observed that range values overlap, so in this case consider the edge value that corresponds to high split index.  
Therefore split value can be 1.8 or 1.4

$G(s) = 1 - (50/100)^2 - (50/100)^2 = 0.5$   
 $G(s1) = 1 - (49/54)^2 - (5/54)^2 = 0.168$   
 $G(s2) = 1 - (1/46)^2 - (45/46)^2 = 0.0425$   
 Gini split index =  $0.5 - (54/100)(0.168) - (46/100)(0.0425) = 0.3897$

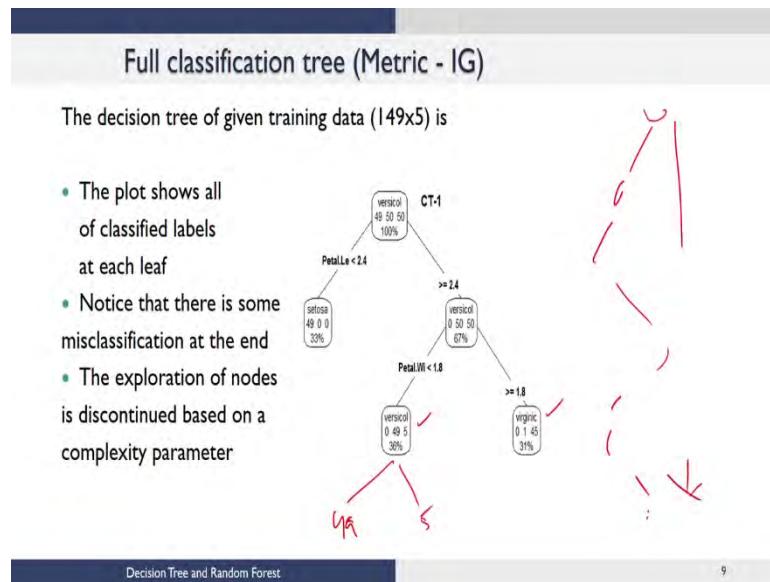
Decision Tree and Random Forest

Now, after doing all of this possibilities we might identify that petal length is the best possibility. So, which if you go back and notice here so, if I do petal length as the possibility then I have already classified all setosa samples. So, I do not need to explore this portion of the tree anymore, simply because there is nothing to do it is already a pure set right the only problem is here. So, what I want to do is I want to start exploring this part of the tree more; and if you notice this that what we are doing it.

So, we are starting with the node where I have this versicolor with 0 50 and 50. Now, again I have multiple choices right, again I can say I want to do petal length again there is nothing stopping you from that again I have this four choices and again I have to compare them some with some value. So, let us say I did choose petal width and then say less than 1.8. Now, what will happen is of the 149 samples 49 have already been classified. So, they are not any more under consideration. So, I start with this other 100 samples which is 50 versicolor and 50 virginica.

And, then what I do is I use this petal width and then say less than 1.8 I collect all of this data from this and see how many come here. So, about 54 of this 100 comes here and about 46 of the 100 comes here. So, 54 + 46 is 100 right. Now, notice what I call this node this node I call as versicolor because most of the samples in this node are from versicolor. And, this node I will call as virginica because most of the samples from this node are from virginica.

(Refer Slide Time: 39:09)



So, ultimately you do all of this and then basically you come up with this kind of decision tree. So, basically what it says is if I am here I have this versicolor as the decision this is setosa, this is versicolor versicolor virginica. So, now when I get a new data point, how will I use this decision tree? The way I will use this decision tree is, I will first take the data point and I will check what is the petal length in the data point right.

If that petal length is 2.4 then I will say for this new data point it is actually setosa and the problem is done right. Now, if it is greater than 2.4 then what I will do is I will look at the petal width of this new data point. And, if it is less than 1.8 I will say its versicolor if its greater than 1.8 I will say its virginica right. Knowing fully, well that I could have errors here and errors here. So, for example, you know 5 times when it is actually virginica this is being called versicolor even in the training data.

So, in the test data we do not know, that is where we will use a test data to check the decision tree and see how many times do I get this to be the correct number right. So, that is what I will look at. Now, there is always you know the possibility of some misclassification at the end the data might not be complete, so that you can clearly classify this problem all the time and you know you. If you are not happy with this and then you are saying ok, in this node its not still pure 49 and 5 can I actually break this down further. So, that I get 49 and 5 you have to look at other features.

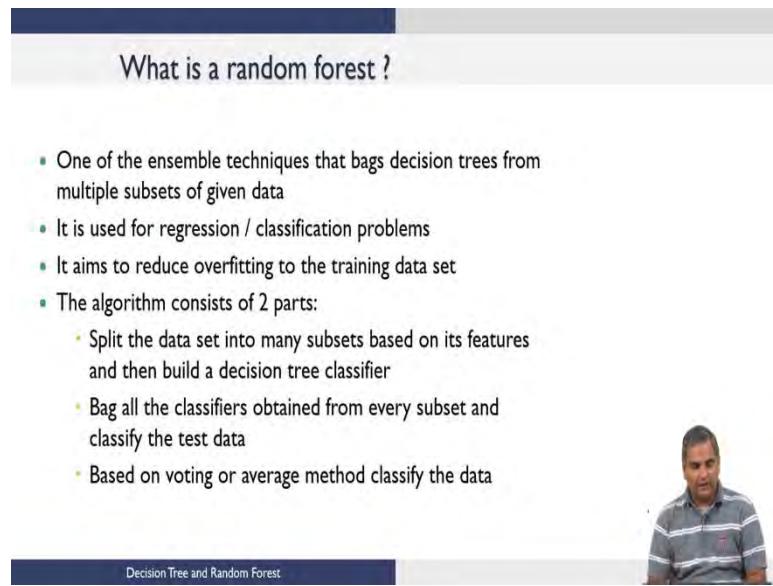
And, see whether the other features will allow you to do this in some cases they might allow you to do it in some cases the data might be such that you cannot do it. And, as I mentioned before even in cases where the data allows you to do this you do not want to keep building a tree which is very very complex right. So, you do not want to have a very very deep tree, which completely learns this training data set which will have no generalization capability.

So, you might want to stop at some point and then say in the training set I am willing to take certain amount of errors so that I have better generalization capability with a test data. Now, that we have seen decision trees random forest is a very very simple idea the key notion of random forest is the following. So, when you look at decision trees if there are minor changes in data or there are minor errors in data, the decision tree that comes about can be quite different.

So, decision trees are not generally very robust to errors in data right because you are choosing some numbers and in some cases if there are errors these numbers might not partition them very well and so on. So, in some sense what you want is if you give the data set and then you build a decision tree. And, if the data change set changes a little bit you do not want to see major changes in the decision tree. You do not want decision trees to completely change which is possible because of errors in data.

So, to avoid that and somehow give it robustness we come up with this idea of random for us; and as the name suggests random forest it is a collection of decision trees right. So, you might ask the question how do I make multiple decision trees from the same data set. So, the way random forest work is a following.

(Refer Slide Time: 42:55)



## What is a random forest ?

- One of the ensemble techniques that bags decision trees from multiple subsets of given data
- It is used for regression / classification problems
- It aims to reduce overfitting to the training data set
- The algorithm consists of 2 parts:
  - Split the data set into many subsets based on its features and then build a decision tree classifier
  - Bag all the classifiers obtained from every subset and classify the test data
  - Based on voting or average method classify the data

You have all this data, what you do is you make multiple data sets from your original data. So, how do you make multiple data sets from your original data? What you do is you sometimes sub select only a portion of the data right. And then say I am going to build a decision tree for this portion of the data, in some cases you might sub select only a certain features in the data. So, in the previous case we saw four features you might say I am going to drop one feature and then say that is my data set right.

So, you can sub select number of data points you can sub select number of features and so on. So, if I have one original data set from this I generate multiple data sets which are kind of marked form of the original data set, either through dropping some data or dropping some features and so on. Now, using the technique that I discussed for each of this data set you can come up with a decision tree right. So, if I make 10 different data sets from a original data then I can build 10 decision trees.

So, there those 10 decision trees together form a random forest ok. Now, you might ask the question has to I have now 10 decision trees which solution from which decision tree do I use?

(Refer Slide Time: 44:25)

### Example continued (subset 1)

Considering only a subset of given training data,

| Sepal.Length | Sepal.Width | Petal.Length | Species   |
|--------------|-------------|--------------|-----------|
| 5.1          | 3.5         | 1.4          | setosa    |
| 4.9          | 3           | 1.4          | setosa    |
| 4.7          | 3.2         | 1.3          | setosa    |
| 4.6          | 3.1         | 1.5          | setosa    |
| .            | .           | .            | .         |
| .            | .           | .            | .         |
| 6.8          | 3.2         | 5.9          | virginica |
| 6.7          | 3.3         | 5.7          | virginica |
| 6.7          | 3           | 5.2          | virginica |
| 6.3          | 2.5         | 5            | virginica |
| 6.5          | 3           | 5.2          | virginica |
| 6.2          | 3.4         | 5.4          | virginica |
| 5.9          | 3           | 5.1          | virginica |

Test data = [S.L = 4.9, S.W = 3, P.L = 1.4]

Prediction of given test data is "Setosa"

Decision Tree and Random Forest

So, in random forest what you do is, if you have 10 decision trees you go through all are those trees get a solution from all of them; and whatever is the majority decision of all these trees that is a solution of the random forest. So, for example, here you might get one subset of data where you only keep sepal length, sepal width and petal length and then build a tree. And, let us say you have a new test data point and let us say this three predicts setosa.

(Refer Slide Time: 44:43)

### Example continued (subset 2)

Considering only a subset of given training data,

| Sepal.Length | Petal.Length | Species   |
|--------------|--------------|-----------|
| 5.1          | 1.4          | setosa    |
| 4.9          | 1.4          | setosa    |
| 4.7          | 1.3          | setosa    |
| 4.6          | 1.5          | setosa    |
| .            | .            | .         |
| .            | .            | .         |
| 6.8          | 5.9          | virginica |
| 6.7          | 5.7          | virginica |
| 6.7          | 5.2          | virginica |
| 6.3          | 5            | virginica |
| 6.5          | 5.2          | virginica |
| 6.2          | 5.4          | virginica |
| 5.9          | 5.1          | virginica |

Test data = [S.L = 4.9, P.L = 1.4]

Prediction of given test data is "Setosa"

Decision Tree and Random Forest

Now, you could have built another tree with only sepal length and petal length and that is another decision tree. And, when you give a new data point you only pick the sepal length petal length attributes of that and run it through this tree and then say let us say that is also saying it is setosa.

(Refer Slide Time: 44:59)

### Example continued (subset 3)

Considering only a subset of given training data,

| Petal.Length | Petal.Width | Species   |
|--------------|-------------|-----------|
| 1.4          | 0.2         | setosa    |
| 1.4          | 0.2         | setosa    |
| 1.3          | 0.2         | setosa    |
| 1.5          | 0.2         | setosa    |
| .            | .           | .         |
| 5.9          | 2.3         | virginica |
| 5.7          | 2.5         | virginica |
| 5.2          | 2.3         | virginica |
| 5            | 1.9         | virginica |
| 5.2          | 2           | virginica |
| 5.4          | 2.3         | virginica |
| 5.1          | 1.8         | virginica |

**Test data = [P.L = 1.4, P.W = 0.2]**

Prediction of given test data is "Setosa"

Decision Tree and Random Forest

Now, you could have petal length and petal width and in these cases you could have dropped some data also right. So, you could sub-select data also and then send this new test data point and if that also says setosa.

(Refer Slide Time: 45:15)

### Final Thoughts

Random forest = function(given data, subset 1, ..., subset 3)

- The final decision observed from 3 different subsets using gini impurity split methods are {"setosa", "setosa", "setosa"}
- Therefore according to voting method, random forest function classifies the test data as "Setosa"

**Advantages of random forest**

- Stochasticity is included in various forms
  - Bagging decision trees
  - Splitting on the basis of random subsets
  - Splitting on the basis of random features (among top)

Decision Tree and Random Forest

Now, there are three trees and all of them said setosa so, the solution is set also. Now, if two of them had set setosa and one is virginica the solution is still setosa. So, basically that is what is called majority. Now, if one says setosa one says virginica and one says versicolor there is a problem, you have to break the tie somehow and then give a solution. But, in general when you have multiple trees you would hope that there will be some consensus among all of these three solutions and you say that solution is the solution for the random forest.

So, by doing this the stochasticity or the problems with the data can be addressed to a large extent in many problems. By basically bagging the trees there are multiple trees from which you are getting the result. And, each of these trees themselves are produced from splitting the data to through dropping of some data points are dropping of some features and so on. So, basically you try to make yourself immune to fluctuations in data by multiple data sets, that you generate and when all of them overwhelmingly say something is a result then it is likely to be correct than just depending on one decision tree, so that is the idea of random forests.

Now, all of this is for you to understand how these things work, but when you use one of these python packages they will do most of this work for you. All you need to know is when you look at it and see tree in to understand what that tree means. And you have to know the difference between random forest and decision tree and so on. So, hopefully this has been useful session for you, to understand decision trees and random for us.

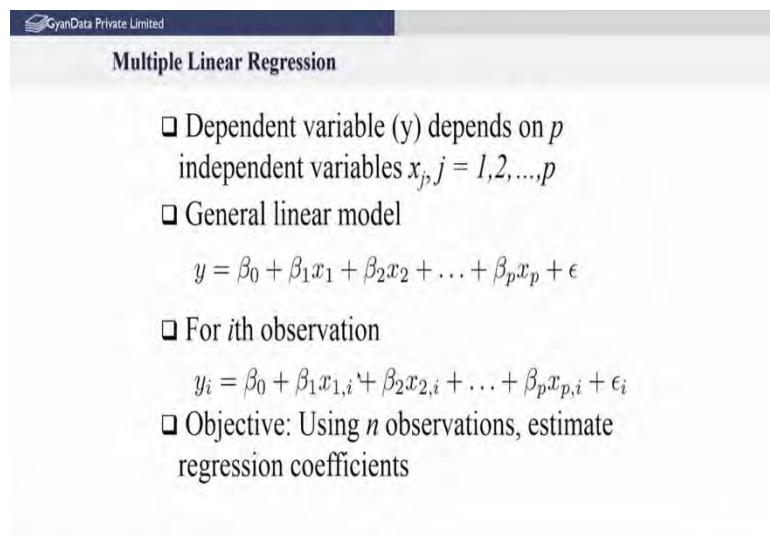
Thank you.

**Python for Data Science**  
**Prof. Ragunathan Rengasamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 40**  
**Multiple Linear Regression**

Welcome, everyone to this lecture on Multiple Linear Regression. In the preceding lectures, we saw how to regress a single independent variable to a dependent variable. Particularly, we were developing a linear model between the independent and dependent variable. We also saw various measures by which we can assess the model that we built. In this lecture, we will extend all of these ideas to multiple linear regression which consists of one dependent variable, but several independent variables.

(Refer Slide Time: 00:49)



The screenshot shows a presentation slide with the title 'Multiple Linear Regression'. Below the title, there is a bulleted list of four items:

- Dependent variable ( $y$ ) depends on  $p$  independent variables  $x_j, j = 1, 2, \dots, p$
- General linear model
- For  $i$ th observation
- Objective: Using  $n$  observations, estimate regression coefficients

Below the list, there is a mathematical equation for the general linear model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$

And below that, the equation for the  $i$ th observation:

$$y_i = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \dots + \beta_p x_{p,i} + \epsilon_i$$

So, as I said that we have a dependent variable which we denote by  $y$  and several independent variables which we denote by the symbols  $x_j$ , where  $j$  equals 1 to  $p$ . There are  $p$  independent variables which we believe affect their dependent variable. We will try to develop a linear model between the dependent variable  $y$  and these independent  $p$  independent variables  $x_j$ ;  $j$  equals 1 to  $p$ .

In general, we can write this linear model as before. We can say

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$

where  $\beta_1, \beta_2, \dots, \beta_p$  represents the slope parameters or the effect of the individual independent variables on the dependent variable.

In addition we also have an error. This error is due to error in the dependent variable, measurement of the dependent variable. In ordinary least squares we always assume that the independent variable measurements are perfectly measured and do not have any error whereas, the dependent variable may contain some error and that error is indicated as  $\epsilon$  we do not know what this quantity is. We have assumed that it is a random quantity with zero mean and some variance.

If we take the  $i$  th sample corresponding to this measurement of  $x_1$  to  $x_p$  and  $y$  corresponding  $y$  we can say that  $i$  th sample dependent variable as

$$y = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \dots + \beta_p x_{p,i} + \epsilon_i$$

and so on for  $i$  equals 1 to  $n$ .

We assume we have small  $n$  number of samples that we have obtained and our aim is to find the values best estimates of  $\beta_0, \beta_1, \beta_2, \dots, \beta_p$  using these  $n$  sample measurements of  $x$ 's and corresponding  $y$ . This is what we call multiple linear regression because we are fitting a linear model and there are many independent variables and we therefore, call the multiple linear regression problem.

(Refer Slide Time: 03:15)

 GyanData Private Limited

## Multiple Linear Regression

- Approach similar to simple regression  
*Minimize the sum of squares of the errors*
- Vector and matrix notations

$$\mathbf{y} = \begin{bmatrix} y_1 - \bar{y} \\ y_2 - \bar{y} \\ \vdots \\ y_n - \bar{y} \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_{1,1} - \bar{x}_1 & x_{2,1} - \bar{x}_2 & \cdots & x_{p,1} - \bar{x}_p \\ x_{1,2} - \bar{x}_1 & x_{2,2} - \bar{x}_2 & \cdots & x_{p,2} - \bar{x}_p \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,n} - \bar{x}_1 & x_{2,n} - \bar{x}_2 & \cdots & x_{p,n} - \bar{x}_p \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

- The linear model in matrix form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, E(\boldsymbol{\epsilon}) = \mathbf{0}, Var(\boldsymbol{\epsilon}) = \sigma^2 \mathbf{I}$$

- SSE

$$S(\boldsymbol{\beta}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Again in order to find the best estimates of the parameters  $\beta_0$  to  $\beta_p$ , we actually set up the minimization of the sum squared of errors. In order to set it up in a compact manner using vectors and matrices we define the following notations. Let us define the vector  $y$  where which consists of all the  $n$  measurements of the dependent variable  $y_1$  to  $y_n$ . We have also done one further things we have subtracted the mean value of all these measurements from each of the observations.

So, the first one represents the first sample value of the dependent variable  $y_1$  - the mean value of  $y$  over all the measurements  $\bar{y}$ . So, the first sample is mean shifted value of the first observation; the second coefficient or second value in this vector is the second sample value - the mean value of the dependent variable and so on for all the  $n$  observations we have. So, these are the mean shifted values of all the  $n$  samples for the dependent variable.

Similarly, we will construct a matrix  $X$  where the first column corresponds to variable independent variable 1; again, what we do is take the sample value of the first independent variable and subtract the mean value of the first independent variable. That means, we take the mean of all these  $n$  samples for the first variable and subtract it from each of the observations of the first independent variable.

So, the first coefficient here would be  $x_1$ , 1 represents the sample value of the first independent variable; first sample first independent variable - the mean value of the first independent variable, and we do this for all  $n$  measurements of the first independent variable. Similarly, we do this for the second independent variable and arrange it in the second column. So, this one represents the observation; the first observation of the second independent variable - the mean value of the second independent variable and we do this for all  $p$  variables independent variables.

So, this particular matrix  $X$  that we get will be a  $n \times p$  matrix;  $n$  is the number of rows,  $p$  is the number of columns. You can view the first row as actually the sample first sample of all independent variables for the first sample of course, we have been shifted that value and the second row is the second sample and so on and each column represents a variable. So, first column represents the first independent variable and the last column represents the  $p$  th independent variable.

So, similarly we will represent all the coefficients  $\beta$ , except  $\beta_0$ , in a vector form  $\beta_1$  to  $\beta_p$  as a column vector. Here basically as I am sorry a row vector. So,  $\beta_1$  is the first coefficient,  $\beta_p$  is the coefficient corresponding the  $p^{\text{th}}$  variable. So, we have  $\beta$  vector which is a  $p \times 1$  vector. We can also define the  $\epsilon$  the noise vector as  $\epsilon_1$  to  $\epsilon_n$  corresponding to all the  $n$  observations.

Now, having defined this rotation we can write our linear model in the form  $y$  equals  $X$  times  $\beta + \epsilon$ . Notice that we have not included  $\beta_0$ , we have eliminated that indirectly by doing this mean subtraction. I will show you how that happens, but you can take in that right now we have only interested in the slope parameters. This linear model only involves the slope parameters  $\beta_1$  to  $\beta_p$  does not involve the  $\beta_0$  parameter because that has been effectively removed from the linear model using this mean subtraction idea.

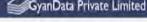
So, we can write our linear model compactly as  $y = X\beta + \epsilon$  and we also make the user assumptions about the error that it is a zero mean vector in this case because it is a multivariate vector,  $0$  is a vector. So, expected value of  $\epsilon_0$  implies  $\epsilon$  is a random vector with  $0$  mean and the variance; covariance matrix of  $\epsilon$  is assumed to be  $\sigma^2 I$ .

Sigma squared identity in this form it means all the  $\epsilon_s$ ,  $\epsilon_1$  to  $\epsilon_n$  have all have the same variance sigma squared homoscedastic assumption. And we also assume that  $\epsilon_1$  and  $\epsilon_2$  are uncorrelated or  $\epsilon_i$  and  $\epsilon_j$  are uncorrelated, if  $i$  is not equal to  $j$ . In which case we can write the covariance matrix of  $\epsilon$  as  $\sigma^2 I$ .

Now, under this assumption we can go ahead and say we want to find the estimates of  $\beta$  so as to minimize the sum square of the errors. So,  $\epsilon$  transpose  $\epsilon$  is a compact way of saying the sum of all errors error squared of all the errors in all the  $n$  measurements. So, expanding this is nothing, but sum of  $\epsilon_i^2$ ,  $i$  equals 1 to  $n$  that is compactly written like this and this is what we want to minimize, but  $\epsilon$  itself can be written as  $(y - X\beta)$ . So, we can write this whole thing as a  $(y - X\beta)^T(y - X)$ .

We want to minimize this which is a function of  $\beta$  by finding the best value of  $\beta$ .

(Refer Slide Time: 08:53)



## Multiple Linear Regression

- ❑ Minimization of the SSE leads to the normal equations  
$$(\mathbf{X}^T \mathbf{X}) \hat{\boldsymbol{\beta}} = \mathbf{X}^T \mathbf{y}$$
- ❑ Assumption:  $(\mathbf{X}^T \mathbf{X})$  is of full rank  $p$  (invertible)
- ❑ The coefficients vector  
$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}; \beta_0 = \bar{y} - \bar{x}^T \hat{\boldsymbol{\beta}}$$
- ❑ The properties of the estimators  
$$E(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$$
$$Var(\hat{\boldsymbol{\beta}}) = \sigma^2 (\mathbf{X}^T \mathbf{X})^{-1}$$
- ❑  $\hat{\boldsymbol{\beta}}$  is the best linear unbiased estimator (BLUE)

Data Analytics

55

So, if we set up this optimization problem to minimize the sum squared errors to find  $\boldsymbol{\beta}$ , we will we can show we can by differentiating that objective function with respect to  $\boldsymbol{\beta}$  and setting it equal to 0; we get what are called the first order conditions and these first order conditions will result in the following set of linear equations. We will get

$$(\mathbf{X}^T \mathbf{X}) \hat{\boldsymbol{\beta}} = \mathbf{X}^T \mathbf{y}.$$

Now, this is a  $p$  cross remember  $\mathbf{X}$  is a  $n$  cross  $p$  matrix. So,  $\mathbf{X}$  transpose is  $p$  cross  $n$ . So, this is a square matrix  $\mathbf{X}$  transpose  $\mathbf{X}$  is a square matrix of size  $p$  cross  $p$  and multiplied by the  $p$  cross 1 vector and similarly it is  $p$  cross 1 on the right hand side. So, these are  $p$  equations in  $p$  variables. Their linear equations in  $\boldsymbol{\beta}$  is all known,  $\mathbf{y}$  is known. So, right hand side is like the, if you, what is that interpreted as  $A\mathbf{x} = \boldsymbol{\beta}$ .. It is a set of linear equations  $p$  equations in  $p$  variables which can be easily solved if  $A$  is invertible.

So, we assume that  $\mathbf{X}^T \mathbf{X}$  is a full rank matrix invertible, the meaning of this will become a little clearer later and if it is not invertible we will have to do other things which we will again talk in another lecture. But, for the time being let us assume that  $\mathbf{X}^T \mathbf{X}$  which is a square matrix is invertible, it is a full rank matrix and then you can easily find the solution for  $\hat{\boldsymbol{\beta}}$  solve this linear set of equations by taking  $A^{-1}B$  which is exactly  $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ . So,  $\hat{\boldsymbol{\beta}}$  the coefficient vector can be found by this thing and this is the solution that minimizes the sum squared errors, this objective function that we have written.

So, once we get  $\beta_1$  the slope parameters,  $\beta_0$  can be estimated as the  $\bar{y} - \bar{x}^T \hat{\beta}$ . Notice that this is very similar to what we have in the univariate case, where it says  $\beta_0$  estimate is nothing, but  $\bar{y} - \bar{x}^T \beta_1$ . So, it is very similar to that, you can see.

You can also compare the solution for the slope parameters with the univariate case which says  $\widehat{\beta_1} = Xy$  divided by  $XX$ . Notice that  $X^T y$  represents  $Xy$  and  $X^T X$  represents  $XX$  in the univariate case you are dividing  $X y$  by  $XX$ , in the multivariate case division is represented by inverse. So, you get  $X$  transpose  $X$  inverse times  $X$  transpose  $y$ .

So, you can see it is very very similar to the solution for the univariate case except that these are matrices and vectors and therefore, you have to be careful. You cannot simply divide, it is matrix times inverse times a vector that is a solution for  $\beta$  which is slope parameters. You can also estimate  $\beta_0$  and  $\beta$  by doing what is called augmentation of the  $x$  vector with a constant value 1 1 1 in the final thing, but I did not use that approach.

Because, the main subtraction approach is a much better approach for estimating whether if for estimating  $\beta_0$  and  $\beta$  slope parameters because this is applicable even to another case called the totally squares. The augmentation approach is valid only for ordinary least squares you cannot use it for total least squares which we will see again later. So, that is why I use the means subtraction route in order to obtain the estimates of the slope parameter first followed by the estimation of  $\beta_0$  using the estimate of the slope parameters in this value.

Now, you can also derive properties of these parameters  $\beta$ , we can show that the expected value of  $\beta$  is  $\beta$  which just means it is an unbiased estimate just as in the univariate case and we can also get the variance of this  $\beta$ . In this case it is a covariance matrix because it is a vector and we can show that the covariance matrix is  $\sigma^2$  times this  $X$  transpose  $X$  inverse. Now, again you can go back and look at the univariate case, there the variance of  $\beta_1$  the slope parameter will be  $\sigma^2$  by  $S_{xx}$ .

In this case it is  $\sigma^2(X^T X)^{-1}$ . So,  $X^T X$  represents  $XX$ .  $\sigma^2$  is the variance of the error corrupting the dependent variables. We may have a priori knowledge sometimes in most cases we may not be able to know this value of sigma square; we may not be given this so, we have to estimate the sigma squared from data and we will show how to get this.

These two parameters that actually we can show the first parameter says that the estimates of  $\beta_1$  the slope parameters are unbiased. So,  $\hat{\beta}$  are unbiased estimator is an unbiased estimator of the true value  $\beta$ . Moreover it can show that among all linear estimators because  $\hat{\beta}$  is a linear function of  $y$ .

Notice that  $(X^T X)^{-1} X^T$  is nothing but a matrix which basically multiplies the measurements  $y$ ; so,  $\hat{\beta}$  can be interpreted as a linear combination of the measurements. Therefore, it is known as a linear estimator. Among all such linear estimators we can show that  $\hat{\beta}$  has the least variance. Therefore, it is called a blue estimator or unbiased estimator with the best linear unbiased estimator that is what it blue represents; best in the sense of having the least variance.

(Refer Slide Time: 14:51)

**Multiple Linear Regression**

- Estimate of the error variance

$$\hat{\sigma}^2 = \frac{\sum(y_i - \hat{y}_i)^2}{n-p-1}$$

where  $(n-p-1)$  is the degrees of freedom (df)

- 1- $\alpha$  confidence intervals for  $\beta_j, j = 0, 1, \dots, p$

$$\hat{\beta}_j \in [\hat{\beta}_j - t_{(n-p-1, \alpha/2)} s.e.(\hat{\beta}_j), \hat{\beta}_j + t_{(n-p-1, \alpha/2)} s.e.(\hat{\beta}_j)]$$

$t_{(n-p-1, \alpha/2)}$  is the  $(1 - \alpha/2)$  percentile point of the  $t$ -distribution with  $(n-p-1)$  df

$$s.e.(\hat{\beta}_j) = \hat{\sigma} \sqrt{c_{jj}}$$

$$C = (X^T X)^{-1}$$

**Data Analytics**

Now, we can also estimate as I said sigma squared from the data and that sigma square estimate is nothing, but the after you fit the linear model you can take the predicted value for the  $i$ th sample from the linear model and compute this residual  $y_i - \hat{y}_i$  which is the measured value - the predicted value for the  $i^{\text{th}}$  sample, square it, take the sum over all possible samples  $n$  samples divided by  $n - p - 1$ . Again, if you go back to your linear case or univariate case you will find that the denominator is  $n - 2$ .

Here you have  $n - p - 1$  because you are fitting  $p + 1$  parameters;  $p$  is slope parameters + 1 offset parameter. Therefore, out of the  $n$  measurements  $p + 1$  are taken away for the deriving the estimates only the remaining things are the degrees of freedom or the

variability in the residuals is caused by the remaining  $n - p - 1$  measurements and that is why you are dividing by  $n - p - 1$ ; whereas, in the univariate case you would have divided by  $n - 2$  because you are estimating only two parameters there.

So, you can see a one-to-one similarity between the univariate regression problem and the multiple linear regression problem in every derivation that we have given here. Now, once you have estimated  $\hat{\sigma}$ , the variance of the error used from the data you can go back and construct confidence intervals for each slope parameter. We can show that the true slope parameter lies in this confidence interval for any confidence interval you might choose;  $1 - \alpha$ ;  $\alpha$  represents like a level of significance.

So, if you say  $\alpha$  is equal to 0.05.  $1 - \alpha$  would represent 0.95. So, that will be a 95 percent confidence interval, ok. Correspondingly, I will find the critical value from the t distribution  $n$  with  $n - p - 1$  degrees of freedom and this represents  $\alpha$  by 2 the lower value probability value from the t distribution and this upper critical value where the probability area under the curve beyond the value is  $\alpha$  by 2.

So,  $n - p - 1$  represents degrees of freedom. Notice that in the univariate case it would have been  $n - 2$ , very very similar. So, the confidence interval for  $\beta_j$  for any given  $\alpha$  can be computed using this particular formula and the term here s.e ( $\hat{\beta}_j$ ) represents the standard deviation of the estimate of  $\hat{\beta}_j$ .

(Refer Slide Time: 17:40)

The screenshot shows a presentation slide with the following content:

- GyanData Private Limited**
- Multiple Linear Regression**
- Multiple correlation coefficient**
- $$Cor(y, \hat{y}) = \frac{\sum(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum(y_i - \bar{y})^2 \sum(\hat{y}_i - \bar{\hat{y}})^2}}$$
- The coefficient of determination  $R^2$**
- $$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$
- Adjusted R-squared,  $R_a^2$**
- $$R_a^2 = 1 - \frac{SSE/(n-p-1)}{SST/(n-1)}$$

A small video player window in the bottom right corner shows a man speaking.

And, that is given by the diagonal element here of this quantity with sigma square replaced by the estimate here. So, we have computed the standard deviation of the parameter  $\hat{\beta}_j$ , estimated parameter  $\hat{\beta}_j$ , by using the estimated value of sigma multiplied by the diagonal element of  $X^T X$ . So, we are fitting the diagonal elements of the covariance matrix of  $\beta$  parameters, ok, that is all we have done.

So, this represents the diagonal element or the square root of the diagonal element which represents the standard deviation of the estimated value of  $\beta$  which is what is used in order to construct this confidence interval. So, every one of this can be computed from the data as you can see and you can construct. Now, in the confidence level can later be used for testing whether the estimated parameter  $\beta$  is significant or insignificant as we will see later.

Now, we will also compute the correlation between  $y$  and  $\hat{y}$  which tells you whether the predicted value from the linear model is resembles or closely related to the measured value. So, typically we will draw a line between the  $y$  the measured value and the predicted value and see whether it is these things fall on the 45 degree line and if it does then we think that the fit is good.

Another way of doing this is to find the correlation coefficient between  $y$  and  $\hat{y}$  which is simply using the standard thing  $y_i - \bar{y}$  multiplied by  $\hat{y}_i - \bar{\hat{y}}$  summed over all quantities divided by the standard deviation of  $y$  and the standard deviation in  $\hat{y}$ , that is for normalization.

We could also use the coefficient of determination  $R^2$ ; just as we did for the univariate case we can compute  $R^2$  as  $1 - (\text{sum squared error})/(\text{sum square total})$  which is nothing, but  $1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$ . So, if we take  $1 -$  this we will actually we can show whether using the independent variables have we been able to get a better fit? If we have obtained a very good fit, then the numerator will be close to 0 and  $R^2$  will be close to one.

On the other hand, if we had not improved the fit because of  $X$ 's any of the  $X$ 's then the numerator will be almost equal to the denominator and therefore, this one will be close to 0. So, a value of  $R^2$  close to 1 as before represents indication of a good linear fit whereas, a value close to 0 indicates the fit is not good. We can also compute the adjusted

R square to account for the degrees of freedom notice that the numerator has  $n - p - 1$  degrees of freedom whereas, the denominator has  $n - 1$  degrees of freedom.

Therefore, we can do an adjusted R squared which divides the SSE by the appropriate degrees of freedom. We can say this is the error due to per degree of freedom that is there in the fit whereas, the denominator represents the error because we have fitted only the offset parameter there are  $n - 1$  degrees of freedom, this is the error per degree of freedom. So, this kind of a thing is also a good indicator instead of using R squared we can use adjusted value of R square.

So, these are all very similar again to the univariate linear regression problem.

(Refer Slide Time: 21:16)

The screenshot shows a presentation slide with the following content:

GyanData Private Limited

## Multiple Linear Regression

- ❑ Fitted model is adequate or can be reduced further?
  - ❑ Test significance of individual coefficient  $\hat{\beta}$
  - ❑ A general unified test on the full model (FM) vs the reduced model (RM)
- ❑ Hypothesis testing
  - $H_0$ : Reduced model is adequate
  - $H_1$ : Full model is adequate

A small video thumbnail of a man speaking is visible in the bottom right corner.

So, we can use; what we can check R squared and see whether the value is close to 1 and if it is we can say maybe the linear model is good to fit the data, but that is not a confirmatory test. We have to do the residual plot as we did in linear regression univariately in linear regression. And, that is what we are going to do further.

So, we are going to find whether the fitted model is adequate or it can be reduced further; what this reduced further means we will explain. In the univariate case there is only one independent variable, but here there are several independent variables. Maybe not all independent variables have an effect on  $y$ ; some of the independent variables may be

irrelevant. So, one way of trying to find whether a particular independent variable has an effect is to test the corresponding coefficient.

Notice we have already defined the confidence interval for each coefficient and we can see whether the confidence interval contains 0 in which case we can say the corresponding independent variable does not have a significant effect on the dependent variable and we can perhaps drop it, ok.

Or we can also do what we call the test F test just as we did in the univariate regression problem we can test whether the full model is better than the reduced model. The reduced model contains no independent variables whereas, the full model can contain all or some of the independent variables you can do many kinds of tests and we will do this. So, we can test whether the reduced model which contains only the constant intercept parameter is a good fit as opposed to including all the independent variable some or all the independent variables that is what we call the full model.

(Refer Slide Time: 23:02)

GyanData Private Limited

## Multiple Linear Regression

- Testing two models: RM with  $k$  parameters
- F-statistic

$$F_o = \frac{[SSE(RM) - SSE(FM)] / (p+1-k)}{SSE(FM) / (n-p-1)}$$

Degrees of freedom

- Note that  $SSE(RM) \geq SSE(FM)$

- For  $\alpha$ -significance level: Reject  $H_0$  if  
 $F_o \geq F_{(p+1-k, n-p-1; \alpha)}$   
where F-statistic for the given dfs from the table

Navigation icons: back, forward, search, etc.

Data Analytics

We will consider a specific case here where we do the F test statistic for the case when we have a reduced model and compare it with the full model. The reduced model we will consider with  $k$  parameters. Specifically, let us consider the reduced model with only one parameter which means that we have only the constant intercept parameter we would not include any of the independent variables and compare it with the full model which contains all of the independent variables including the intercept.

So, the reduced model is one which contains only the offset parameter and no independent variables, the full model is the case where we consider all the independent variables and the intercept parameter. So, the number of parameters we are estimating in the reduced model is only one. So,  $k$  equals 1 and the full model is the case where we have all the independent variables  $p$  independent variables. So, we are estimating  $p + 1$  parameters in the full model, ok.

So, what we do is perform a fit and compute the sum squared errors which is nothing, but the difference between  $y$ , the measured value and the predicted value. So, we will first take the model containing only the offset or the intercept parameter and estimate. In this case of course,  $\bar{y}$  will be the best estimate and we will compute sum squared errors which is nothing, but the variance of the measurements for the dependent variable.

Then, we would also perform a linear regression containing all the parameters independent variables. And, in this case if we compute the difference between  $y$  and  $y$  predicted and take the sum squared errors that is the SSE of the full model. So, when we want to compare whether we want to accept the full model as compared to the Reeves model what we do is take the difference in the sum squared errors. Remember, the sum squared errors for the reduced model will be greater than the sum squared errors for the full model because the full model contains more number of parameters and therefore, you will get a better fit.

So, the difference in the fit, which is difference in the sum squared errors between the reduced model fit and the full model fit that is the numerator divided by what we call the degrees of freedom. Notice, the full model has  $p + 1$  parameters.  $p$  independent variable + the offset and the reduced model in this particular case contains only one parameter. So,  $k$  equals 1. So, the degrees of freedom will be  $p$ .

So, you divide this difference in the sum squared errors by  $p$ . Denominator is the sum squared errors of the full model which contains  $n - p - 1$  degrees of freedom because  $p + 1$  parameters have been fitted. Therefore, the degrees of freedom is to total number of measurements -  $p - 1$ . And, so, we divide that sum squared errors for the denominator by the number of degrees of freedom and then take this ratio as defined and that is your F statistic.

Now, in order to reject if we want to reject the null hypothesis or if you want to test the null hypothesis against its alternative, we find the test criteria for the  $\alpha$  level of

significance. We would take it from the F distribution where the numerator degrees of freedom is  $p + 1 - k$  for this particular case it is exactly  $p$ . And the denominator degrees of freedom is  $n - p - 1$  and  $\alpha$  level of significance we use and we compute the test criteria critical value from the F distribution.

Then, we compare the test statistic with the critical value and if the test statistic exceeds the critical value at this level of significance then we reject the null hypothesis that is we will say the full model is a better choice and the independent variables do make a difference. And, this is a standard thing that R function will provide. This particular comparison between the reduced model which has no independent variables and the full model which contains all the independent variables in multi linear regression. Of course, you can choose different reduced models and compare with the full model.

For example, you can take the reduced model by leaving out only one of the independent variables so, that we will have  $p$  parameters. We can compare it with the full model and again perform a test to decide whether the inclusion of that independent variable makes a difference or not. So, this kind of combination can be done depending on what stage you are and that will be you using in what we call the sequential method for subset selection that will be discussed in a later lecture.

But, essentially the R functions only provide a comparison between the reduced model which contains no independent variable and the full model which contains all of the independent variables.

(Refer Slide Time: 27:50)

Multiple Linear Regression

Menu pricing in Restaurants of NYC

$y$  : Price of dinner  
 $x_1$ : Customer rating of the food (Food)  
 $x_2$ : Customer rating of the décor (Décor)  
 $x_3$ : Customer rating of the service (Service)  
 $x_4$ : If the restaurant is east or west (East)

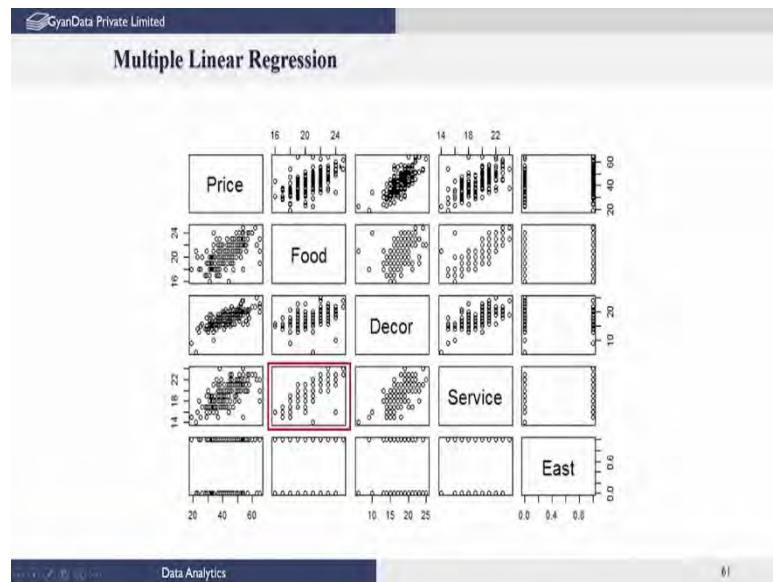
Objective: Build a model

Data Analytics

Let us go through a simple example in order to what are called revisit these ideas. So, in this case we have what is called the a price data where customers have been asked to rate the food than the other aesthetics of a particular restaurant. And, we also and the cost of the particular dinner also a data were obtained for these restaurants and the location of these restaurants whether on they are on the east side of a particular street in New York or the west side. Typically, in New York west side is probably a little poorer whereas, the east side probably is a little richer neighbourhood. So, location of the restaurant also would indicate would have an effect on the price.

So, these are the four independent variables people data was obtained on the quality of the food, the decor and service all this was rated by the customers and at the location of this restaurant and the price of dinner in that served in that restaurant was also taken. So, you would expect that the quality of the food the service level all of this would have a very direct influence on the price in the restaurant. And, a linear model was built between  $y$  and the independent variables  $x_1$  to  $x_4$ .

(Refer Slide Time: 29:16)



So, before we build a model we do a scatter plot as usual any visualization and here you because there are several independent variables we have not just one plot scatter plot between y and x 1. For example, in this case remember price is y, y versus x 1 this particular plot shows the correlation or the scatter plot for y versus x 1 or y versus food, price versus food. The second one is the scatter plot price and decor the third one is the scatter plot between price and service and the last one is price versus location, ok.

And, similarly you can actually develop a scatter plot between food and decor which is here or food and service and so on. Even though we consider all these variables that we have obtained like food decor service location as independent, it is possible when we select these variables they are not truly independent. There might be inter dependencies between the what we call the so called independent variables, that can give rise to problem in a regression which we will see later that what we call the effect of co-linearity.

But, a scatter plot may reveal some interdependencies between the independent so called independent variables. So, for example, if we look at the scatter plot between food and decor it is seems to be completely randomly distributed this does not seem to be any quite correlation. However, food and service seems to be very strongly correlated. There seems to be a linear relationship between food and service. So, perhaps you do not need to include both these variables. We will see later that is this true, but in this just a scatter plot itself reveals some interesting features.

And, so, we will now go ahead and say perhaps a linear model between price and food then decorous is seems to be pointed out or indicated by the scatter plots let us go ahead and build one.

(Refer Slide Time: 31:16)

Multiple Linear Regression

Regression output from R

| Coefficients: | Estimate   | Std. Error | t value | Pr(> t )     |
|---------------|------------|------------|---------|--------------|
| (Intercept)   | -24.023800 | 4.708359   | -5.102  | 9.24e-07 *** |
| Food          | 1.538120   | 0.368951   | 4.169   | 4.96e-05 *** |
| Decor         | 1.910087   | 0.217005   | 8.802   | 1.87e-15 *** |
| Service       | -0.002727  | 0.396232   | -0.007  | 0.9945       |
| East          | 2.068050   | 0.946739   | 2.184   | 0.0304 *     |

---

Signif. codes:  
0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.738 on 163 degrees of freedom  
Multiple R-squared: 0.6279, Adjusted R-squared: 0.6187  
F-statistic: 68.76 on 4 and 163 DF, p-value: < 2.2e-16

$$\hat{y}_i = -24.024 + 1.538x_1 + 1.910x_2 - 0.003x_3 + 2.068x_4$$

Remove  $x_3$

And, if we apply the R function lm to this data set and we examine the output, we will get this output from R and tells that the intercept term is - 24.02 and the slope parameters the coefficient multiplying food is 1.5, the coefficient multiplying decor is 1.9 and so on so forth.

It also gives you the standard error for each coefficient as well as the offset parameter which is nothing, but the sigma value for the estimated quantities and it also gives you the probability values p-values as we call them. And, notice if the p-value is very low, it means that this coefficient is significant. We cannot take it that this value is 0, ok. Any low value of this indicates that the corresponding coefficient is significantly different from 0.

So, in this case the first three has very low p-values and therefore, the significant, but service has a high p-value. Therefore, it seems to indicate that this coefficient is insignificant is equal almost equal to 0 that is what this indicates. If you look at the east which is this independent location parameter that has does not have a very low p-value, but it is still not bad 0.03 and therefore, it is significant only is insignificant only if you take a level of significance of 0.025 or something like that. If you take 0.1 or 0.05 and so

on you will still consider this east this coefficient to be significant and that is what this is basically pointing out, this star indicates that.

So, now we will go ahead and try to actually look at the F value also, the F statistic says that the full model as compared to the reduced model of using only the intercept is actually significant; which means, the constant model is not good and including these variables results in a better fit or explanation of the price and therefore, you should actually include this.

Whether you should include all of them are only some of them we can do different kinds of tests to find out what we have done in this particular case is only compare the model without any of these independent variables which is called the constant model with all of these variables included. That is the only two model comparisons we are made the reduced model is one containing only the interceptor and the full model is one which contains intercept and all four independent variables and that is the p-value it has given the corresponding F statistic.

So, we are saying that including these independent variables is important in explaining the price, ok. So, but it may turn out that all of them is not necessary and that we will we will examine further. So, the corresponding fit that we obtain is this. As I said that from the, what you call the as the confidence interval for the slope parameter for service we can say that we can remove this it is insignificant and perhaps we can remove this and try the fit. For the time being let us actually remove this and try the fit.

(Refer Slide Time: 34:42)

The screenshot shows a presentation slide with the title 'Multiple Linear Regression'. Below the title, it says 'Regression output from R without Service variable'. The output includes:

|             | Estimate | Std. Error | t value | Pr(> t )     |
|-------------|----------|------------|---------|--------------|
| (Intercept) | -24.0269 | 4.6727     | -5.142  | 7.67e-07 *** |
| Food        | 1.5363   | 0.2632     | 5.838   | 2.76e-08 *** |
| Decor       | 1.9094   | 0.1900     | 10.049  | < 2e-16 ***  |
| East        | 2.0670   | 0.9318     | 2.218   | 0.0279 *     |

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.72 on 164 degrees of freedom  
Multiple R-squared: 0.6279, Adjusted R-squared: 0.6211  
F-statistic: 92.24 on 3 and 164 DF, p-value: < 2.2e-16

$$\hat{y}_i = -24.027 + 1.536x_1 + 1.910x_2 + 2.067x_4$$

*Caution: Removing several predictors may have a dramatic effect on the coefficients in the reduced model*

We have done that. We have only included now food, decor in east and done the regression again and it turns out that the regression thing is still what you call the R squared value is improved, not improved significantly, but not reduced. And, F value is significant and we get the more or less the same coefficients for the other parameters also the intercept term and the slope parameters.

It indicates that x 3 is not adding any value to the prediction of y. The reason for this as we said if you look at the scatter plot service and food are very strongly correlated. Therefore, only either food or service needs to be included in order to explain price and not both right. And, in this case service is being removed, but you can try removing food as the variable and try to fit between price, decor and decor service and east and you will find that the regression is as good as retaining food and eliminating service.

(Refer Slide Time: 35:44)

GyanData Private Limited

### Multiple Linear Regression: Diagnostics

- ❑ Residual plots: Standardized residuals for assessing
  - ❑ Linear vs nonlinear model
  - ❑ Normality of the errors
  - ❑ Homoscedastic vs heteroscedastic errors

Similar to  
Simple  
regression



Data Analytics

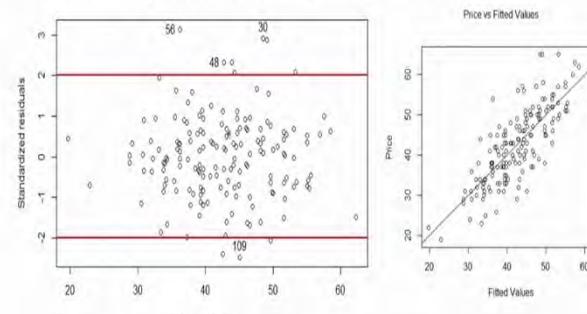
R squared value and the F statistics seems to indicate that we can go ahead with the linear model, but we should further examine the standardized residual plot for concluding whether the linear model is or not. There should be no pattern in the residuals.

(Refer Slide Time: 36:00)

GyanData Private Limited

### Multiple Linear Regression: Testing for linearity

- ❑ Residuals plot: standardized residuals vs fitted values



No Pattern: Based on this and other measures (R<sup>2</sup>, F-test)  
we can conclude that a linear model is acceptable

Data Analytics

So, let us actually do the residual plot. Here we have taken the standardized residuals and plotted it against what is called the predicted price value or the fitted value. Remember, this is  $\hat{y}_i$ ;  $\hat{y}_i$  is only one variable so, you need to generate only one plot and we have also shown here the in red lines the confidence interval for the standardized residuals

and anything above this outside of this interval indicates outliers. So, for example, 56, sample number 48, sample number 30 and 109 and so on so forth may be possible outliers. And, but there is no pattern in the standardized residuals it spread randomly within this boundary and therefore, we can say since there is no pattern a linear fit is acceptable.

So, here the quality of the fit is shown here. So, the actual price, the measured value versus the  $\hat{y}_i$  predicted value is shown and a linear model seems to explain the data reasonably well. The last thing is, we have these outliers; if you want to improve the fit you may want to remove let us say the outlier which is farthest away from the boundary. For example, you may want to remove 56 and redo the linear regression multi multiple linear regression and again repeat it until there are no outliers that will improve the R squared value and the fit quality of the fit a little more, ok.

So, we have not done that we leave this as an exercise for you. So, what we have done is we have seen that whatever was valid for the univariate regression can be extended to the multiple linear regression except that scalars there will get replaced by vectors and matrices corresponding. What was the variance there will become a variance covariance matrix here, what was a vector the scalar there might mean scalar might become a mean vector here.

So, you will see a one to one correspondence, but the residuals plot and interpretation of confidence interval for  $\beta$  all of this the F statistic are more or less similar, except that understand in the multiple linear regression there are several independent variables. All of them may not be relevant. We may be able to take only a subset and I will actually handle subset selection as a separate lecture. For the time being we are just an a significance test on the coefficient in order to identify the irrelevant independent variables, but there are better approaches and we will take that up in the following lectures.