TOOLS AND TECHNIQUES OF DATA SCIENCE

(R18A1209)

DIGITAL NOTES

B. TECH
IV Year - II Sem
(2022-23)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING &TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
Sponsored by CMR Educational Society
(Affiliated to JNTU, Hyderabad, Approved by AICTE- Accredited by NBA& NAAC–'A'Grade-ISO9001:2008Certified)

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

IV Year B.Tech IT –II Sem	**L**	**T/P/D**	**C**
	3	**-/-/-**	**3**

(R18A1209) TOOLS AND TECHNIQUES OF DATA SCIENCE

Course Objectives :

- To understand the various concepts in Data Science process.
- To study the applications of Data Science.
- To learn to setup the data science tools environment and implement in Python and R
- To learn to write programs in Python and R for data science projects.
- To know the process of data visualization¬ & data manipulation w.r.to data science.

UNIT - I

Introduction to Data Sciences – The data science process – Roles in data science project – Stages of Data Science Project – Defining the goal – Data collection and Management – Modelling – Model evaluation – Presentation and documentation – Model deployment and Maintenance

Applying Data science in Industry – Benefits from Business centric Data Science – Data Analytics and Types – Common Challenges in Analytics – Distinguishing between Business Intelligence and Data Science

UNIT-II

Using Data Science to Extract meaning from Data – Machine learning Modeling with instances

Data science tools environment - Python – overview - Setting up Data science toolbox

UNIT – III

Usage of Data science tools environment: Essential concepts and tools-Obtaining – Managing your Data workflow – Drake.

Techniques using Python Tools - k – NearestNeighbours – Naive Bayes

UNIT-IV

Techniques using R Tools - R programming overview - Loading data into R – Modeling methods – choosing and evaluating models –Linear and logistic Regression.

UNIT-V

Data Manipulation using pandas: Installing and using Pandas- Introducing pandas objects- Data Indexing and selection – handling missing data, merge and joining sets, Aggregation and grouping.

Data Visualization using Matplotlib – Simple Line Plots- Simple Scatter plots, Multiple subplots, Visualization with seaborn.

TEXT BOOKS:

1. J. Janssens, Data science at the command line, First edition. Sebastopol, CA: O‘Reilly, 2014..

2. J. Grus, Data Science from Scratch: First Principles with Python, 1 edition. Sebastopol, CA: O'Reilly Media, 2015.
3. N. Zumel and J. Mount, Practical data science with R. Shelter Island, NY: Manning Publications Co, 2014.

REFERENCE BOOKS:

1. L. Pierson and J. Porway, Data science, 2nd edition. Hoboken, NJ: John Wiley and Sons, Inc, 2017.
2. C. O'Neil and R. Schutt, Doing Data Science: Straight Talk from the Frontline, 1 edition. Beijing ; Sebastopol: O'Reilly Media, 2013.
3. J. VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data, First edition. Shroff/O'Reilly, 2016.
4. S. R. Das, Data Science: Theories, Models, Algorithms, and Analytics. https://srdas.github.io/MLBook/.

Course Outcomes:

Students will be able to

- Demonstrate the basic knowledge of data science process.
- Setup the software environment for python and R Lanaguage and apply various techniques to work with data.
- Manipulate and visualize the data using tools like pandas and matplotlib.
- Develop simple data science applications.
- Analyze the various data science related projects.

INDEX

UNIT - I

Introduction to Data Science – The data science process – Roles in data science project – Stages of Data Science Project – Defining the goal – Data collection and Management – Modelling – Model evaluation – Presentation and documentation – Model deployment and Maintenance

Applying Data science in Industry – Benefits from Business centric Data Science – Data Analytics and Types – Common Challenges in Analytics – Distinguishing between Business Intelligence and Data Science

Introduction to Data Sciences

What is data science?

The statistician William S. Cleveland defined data science as an interdisciplinary field larger than statistics itself. We define data science as managing the process that can transform hypotheses and data into actionable predictions. Typical predictive analytic goals include predicting who will win an election, what products will sell well together, which loans will default, or which advertisements will be clicked on. The data scientist is responsible for acquiring the data, managing the data, choosing the modeling technique, writing the code, and verifying the results.

Because data science draws on so many disciplines, it's often a "second calling." Many of the best data scientists we meet started as programmers, statisticians, business intelligence analysts, or scientists. By adding a few more techniques to their repertoire, they became excellent data scientists.

The data science process

The data scientist is responsible for guiding a data science project from start to fin- ish. Success in a data science project comes not from access to any one exotic tool, but from having quantifiable goals, good methodology, cross-discipline interactions, and a repeatable workflow.

The roles in a data science project

Data science is not performed in a vacuum. It's a collaborative effort that draws on a number of roles, skills, and tools. Before we talk about the process itself, let's look at the roles that must be filled in a successful project. Project management has been a central concern of software engineering for a long time, so we can look therefor guidance.

Project roles

Let's look at a few recurring roles in a data science project in table 1.1.

Table 1.1 Data science project roles and responsibilities

Role	Responsibilities
Project sponsor	Represents the business interests; champions the project
Client	Represents end users' interests; domain expert
Data scientist	Sets and executes analytic strategy; communicates with sponsor and client
Data architect	Manages data and data storage; sometimes manages data collection
Operations	Manages infrastructure; deploys final project results

Sometimes these roles may overlap. Some roles—in particular client, data architect, and operations—are often filled by people who aren't on the data science project team, but are key collaborators.

PROJECT SPONSOR:

The most important role in a data science project is the project sponsor. The sponsor is the person who wants the data science result; generally, they represent the business interests. The sponsor is responsible for deciding whether the project is a success or failure. The data scientist may fill the sponsor role for their own project if they feel they know and can represent the business needs, but that's not the optimal arrangement. The ideal sponsor meets the following condition: if they're satisfied with the project outcome, then the project is by definition a success. Getting sponsor sign-off becomes the central organizing goal of a data science project.

KEEP THE SPONSOR INFORMED AND INVOLVED It's critical to keep the sponsor informed and involved. Show them plans, progress, and intermediate successes or failures in terms they can understand. A good way to guarantee project failure is to keep the sponsor in the dark.

To ensure sponsor sign-off, you must get clear goals from them through directed interviews. You attempt to capture the sponsor's expressed goals as quantitative statements. An example goal might be "Identify 90% of accounts that will go into default at least two months before the first missed payment with a false positive rate of no more than 25%." This is a precise goal that allows you to check in parallel if meeting the goal is actually going to make business sense and whether you have data and tools of sufficient quality to achieve the goal.

CLIENT

While the sponsor is the role that represents the business interest, the client is the rolethat represents the model's end users' interests. Sometimes the sponsor and client roles may be filled by the same person. Again, the data scientist may fill the client roleif they can weight business trade-offs, but this isn't ideal.

The client is more hands-on than the sponsor; they're the interface between the technical details of building a good model and the day-to-day work process into which the model will be deployed. They

aren't necessarily mathematically or statistically sophisticated, but are familiar with the relevant business processes and serve as the domain expert on the team.

As with the sponsor, you should keep the client informed and involved. Ideally, you'd like to have regular meetings with them to keep your efforts aligned with the needs of the end users. Generally, the client belongs to a different group in the organization and has other responsibilities beyond your project. Keep meetings focused, present results and progress in terms they can understand, and take their critiques to heart. If the end users can't or won't use your model, then the project isn't a success,in the long run.

DATA SCIENTIST

The next role in a data science project is the data scientist, who's responsible for taking all necessary steps to make the project succeed, including setting the project strategy and keeping the client informed. They design the project steps, pick the data sources, and pick the tools to be used. Since they pick the techniques that will be tried, they have to be well informed about statistics and machine learning. They're also responsible for project planning and tracking, though they may do this with a project management partner. At a more technical level, the data scientist also looks at the data, performs statistical tests and procedures, applies machine learning models, and evaluates results—the science portion of data science.

DATA ARCHITECT

The data architect is responsible for all of the data and its storage. Often this role is filled by someone outside of the data science group, such as a database administrator or architect. Data architects often manage data warehouses for many different projects, and they may only be available for quick consultation.

OPERATIONS

The operations role is critical both in acquiring data and delivering the final results. The person filling this role usually has operational responsibilities outside of the data science group. For example, if you're deploying a data science result that affects how products are sorted on an online shopping site, then the person responsible for running the site will have a lot to say about how such a thing can be deployed. This person will likely have constraints on response time, programming language, or data size that you need to respect in deployment. The person in the operations role may already be supporting your sponsor or your client, so they're often easy to find (though their time may be already very much in demand).

Stages of a data science project

The ideal data science environment is one that encourages feedback and iteration between the data scientist and all other stakeholders. This is reflected in the lifecycleof a data science project. Often, you'll loop back and forth between two or more stages before moving forward in the overall process. This is shown in figure 1.1.

Even after you complete a project and deploy a model, new issues and questions can arise from seeing that model in action. The end of one project may lead into a follow-up project.

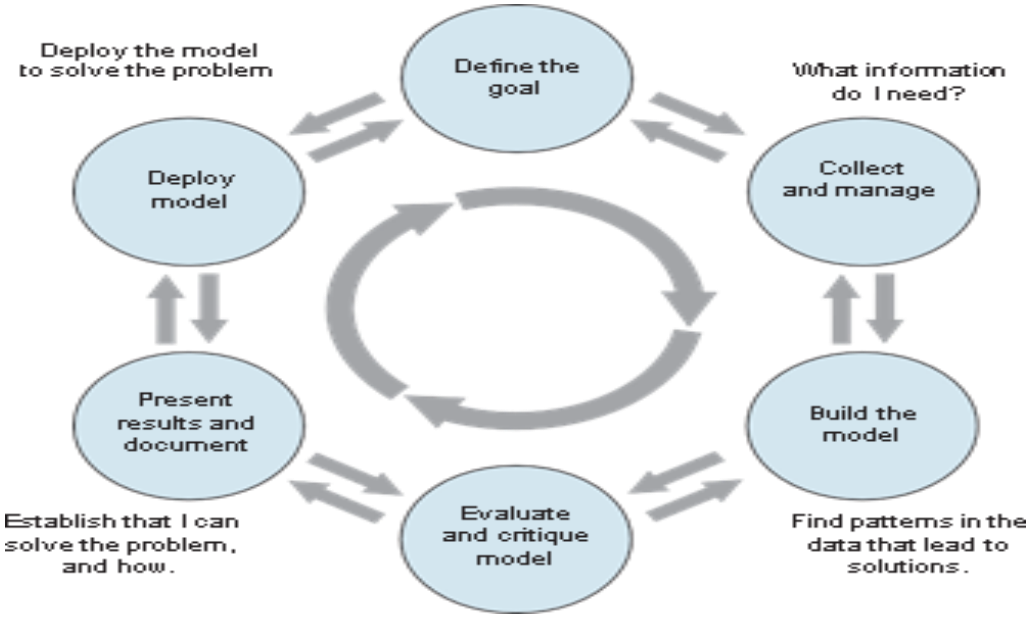


Figure 1.1 The lifecycle of adata science project: loops within loops

Let's look at the different stages shown in figure 1.1. As a real-world example, sup- pose you're working for a German bank.[1] The bank feels that it's losing too muchmoney to bad loans and wants to reduce its losses. This is where your data scienceteam comes in.

Defining the goal

The first task in a data science project is to define a measurable and quantifiable goal.At this stage, learn all that you can about the context of your project:

Why do the sponsors want the project in the first place? What do they lack, andwhat do they need?

What are they doing to solve the problem now, and why isn't that good enough?

What resources will you need: what kind of data and how much staff? Will youhave domain experts to collaborate with, and what are the computational resources?

How do the project sponsors plan to deploy your results? What are the constraints that have to be met for successful deployment?

Let's come back to our loan application example. The ultimate business goal is to reduce the bank's losses due to bad loans. Your project sponsor envisions a tool to help loan officers more accurately score loan applicants, and so reduce the number ofbad loans made. At the same time, it's important that the loan officers feel that they have final discretion on loan approvals.

Once you and the project sponsor and other stakeholders have established preliminary answers to these questions, you and they can start defining the precise goal of the project. The goal should be specific and measurable, not "We want to get better atfinding bad loans," but instead, "We want to reduce our rate of loan charge-offs by atleast 10%, using a model that predicts which loan applicants are likely to default."

A concrete goal begets concrete stopping conditions and concrete acceptance criteria. The less specific the goal, the likelier that the project will go unbounded, because no result will be "good enough." If you don't know what you want to achieve, you don't know when to stop trying—or even what to try. When the project eventually terminates—because either time or resources run out—no one will be happy with the outcome.

This doesn't mean that more exploratory projects aren't needed at times: "Is theresomething in the data that correlates to higher defaults?" or "Should we think about reducing the kinds of loans we give out? Which types might we eliminate?" In this situation, you can still scope the project with concrete stopping conditions, such as a time limit. The goal is then to come up with candidate hypotheses. These hypotheses can then be turned into concrete questions or goals for a full-scale modeling project. Once you have a good idea of the project's goals, you can focus on collecting datato meet those goals.

Data collection and management

This step encompasses identifying the data you need, exploring it, and conditioning it to be suitable for analysis. This stage is often the most time-consuming step in the process. It's also one of the most important:

What data is available to me?

Will it help me solve the problem?

Is it enough?

Is the data quality good enough?

Imagine that for your loan application problem, you've collected a sample of representative loans from the last decade (excluding home loans). Some of the loans havedefaulted; most of them (about 70%) have not. You've collected a variety of attributesabout each loan application, as listed in table 1.2.

Table 1.2 Loan data attributes

`Status.of. existing. checking. account` *(at time of application)*
`Duration.in.month` *(loan length)*
`Credit.history`
`Purpose` *(car loan, student loan, etc.)*
`Credit.amount` *(loan amount)*
`Savings.Account.or. bonds` *(balance/amount)*
`Present. Employment. since`
`Installment.rate.in.percentage.of. disposable. Income`
`Personal.status.and.sex`
`Cosigners Present.residence.since Collateral` *(car, property, etc.)*
`Age.in.years`
`Other.installment. plans` *(other loans/lines of credit—the type)*
`Housing` *(own, rent, etc.)*
`Number.of.existing.credits.at.this. bank`
`Job` *(employment type)*
`Number.of. dependents`
`Telephone` *(do they have one)*

`Good.Loan` *(dependent variable)*

In your data, `Good.Loan` takes on two possible values: `GoodLoan` and `BadLoan`. For the purposes of this discussion, assume that a `GoodLoan` was paid off, and a `BadLoan`defaulted.

As much as possible, try to use information that can be directly measured, rather than information that is inferred from another measurement. For example, you mightbe tempted to use income as a variable, reasoning that a lower income implies more difficulty paying off a loan. The ability to pay off a loan is more directly measured byconsidering the size of the loan payments relative to the borrower's disposable income. This information is more useful than income alone; you have it in your dataas the variable `Installment.rate.in.percentage.of. disposable. Income`.

This is the stage where you conduct initial exploration and visualization of the data. You'll also clean the data: repair data errors and transform variables, as needed.In the process of exploring and cleaning the data, you may discover that it isn't suit-able for your problem, or that you need other types of information as well. You maydiscover things in the data that raise issues more important than the one you originally planned to address. For example, the data in figure 1.2 seems counterintuitive.

Why would some of the seemingly safe applicants (those who repaid all credits tothe bank) default at a higher rate than seemingly riskier ones (those who had been delinquent in the past)? After looking more carefully at the data and sharing puzzling findings with other stakeholders and domain experts, you realize that this sample is inherently biased: *you only have loans that were actually made (and therefore already accepted)*. Overall, there are fewer risky-looking loans than safe-looking ones in thedata. The probable story is that risky-looking loans were approved after a much stricter vetting process, a process that perhaps the safe-looking loan applicationscould bypass. This suggests that if your model is to be used downstream of the currentapplication approval process, credit history is no longer a useful variable. It also suggests that even seemingly safe loan applications should be more carefully scrutinized. Discoveries like this may lead you and other stakeholders to change or refine the project goals. In this case, you may decide to concentrate on the seemingly safe loanapplications. It's common to cycle back and forth between this stage and the previousone, as well as between this stage and the modeling stage.

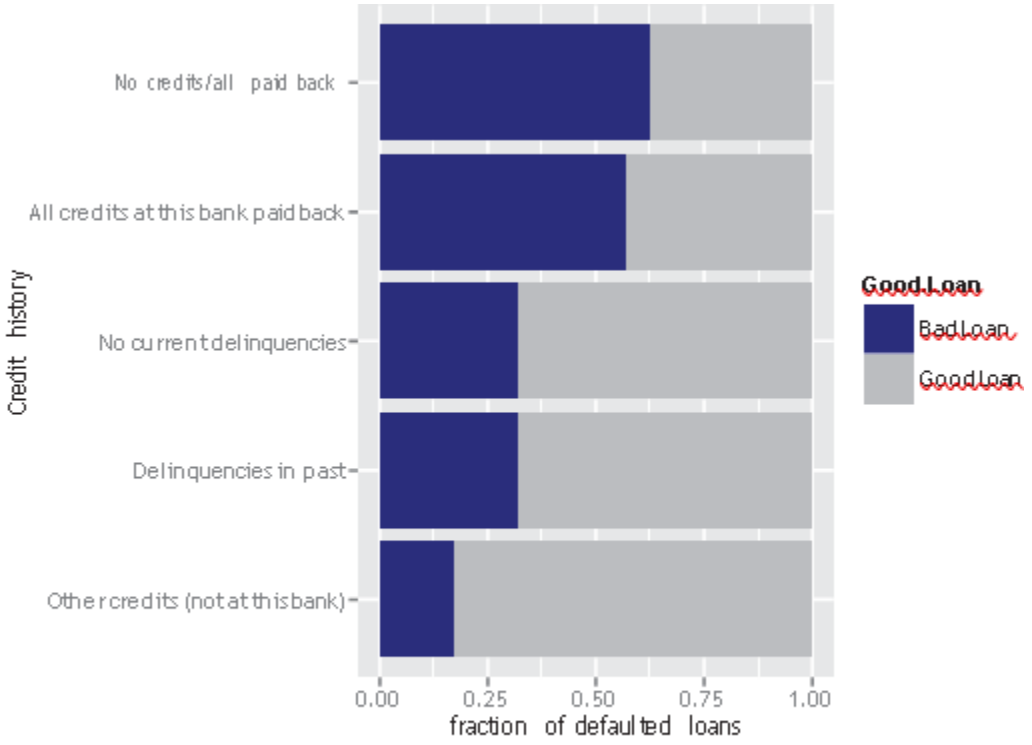


Figure 1.2 The fraction of defaulting loans by credit history category. The dark region of each bar represents the fraction of loans in that category that defaulted.

Modeling

You finally get to statistics and machine learning during the modeling, or analysis, stage. Here is where you try to extract useful insights from the data in order to achieveyour goals. Since many modeling procedures make specific assumptions about data distribution and relationships, there will be overlap and back-and-forth between the modeling stage and the data cleaning stage as you try to find the best way to representthe data and the best form in which to model it.

The most common data science modeling tasks are these: *Classification—*

Deciding if something belongs to one category or another *Scoring—Predicting* or

estimating a numeric value, such as a price or probability*Ranking*—Learning to *order*

items by preferences

Clustering—Grouping items into most-similar groups

Finding relations—Finding correlations or potential causes of effects seen in the data

Characterization—Very general *plotting* and *report generation* from data

For each of these tasks, there are several different possible approaches.

The loan application problem is a classification problem: you want to identify loanapplicants who are likely to default. Three common approaches in such cases are logistic regression, Naive Bayes classifiers, and decision trees. You've been in conversation with loan officers and others who would

be using your model in the field, so you know that they want to be able to understand the chain of reasoning behind the model's classification, and theywant an indication of how confident the model is in its decision: is this applicant highly likely to default, or only somewhat likely? Given the preceding data, youdecide that a decision tree is most suitable

Let's suppose that you discover the model shown in figure 1.3.

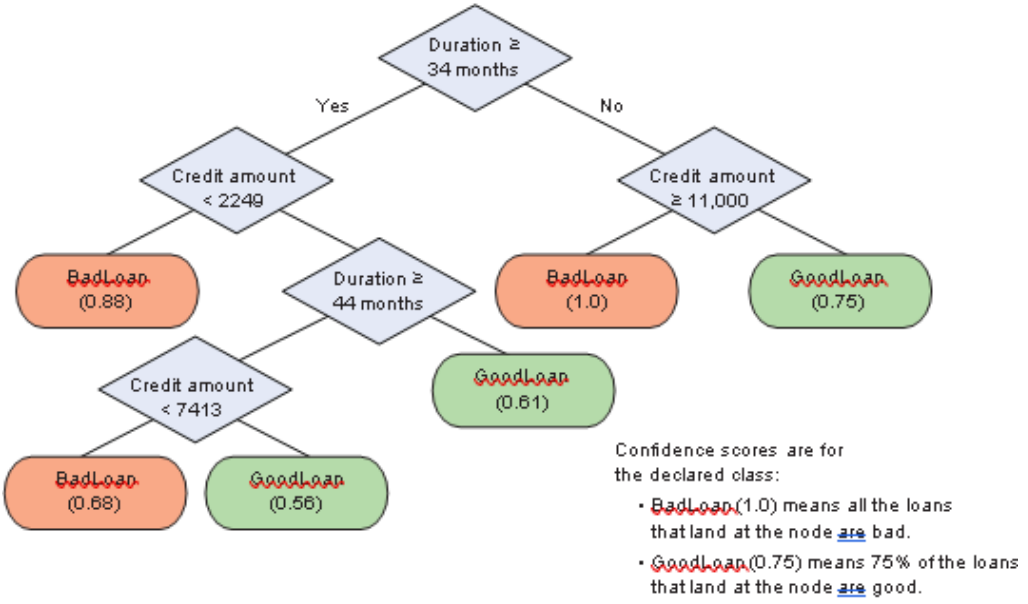


Figure 1.3 A decision tree model for finding bad loan applications, with confidence scores

Model evaluation and critique

Once you have a model, you need to determine if it meets your goals:

Is it accurate enough for your needs? Does it generalize well?

Does it perform better than "the obvious guess"? Better than whatever estimateyou currently use?

Do the results of the model (coefficients, clusters, rules) make sense in the con-text of the problem domain?

If you've answered "no" to any of these questions, it's time to loop back to the modeling step—or decide that the data doesn't support the goal you're trying to achieve. Noone likes negative results, but understanding when you can't meet your success criteria with current resources will save you fruitless effort. Your energy will be better spent on crafting success. This might mean defining more realistic goals or gathering the additional data or other resources that you need to achieve your original goals.

Returning to the loan application example, the first thing to check is that the rulesthat the model discovered make sense. Looking at figure 1.3, you don't notice any obviously strange rules, so you can go ahead and evaluate the model's accuracy. A good summary of classifier accuracy is the *confusion matrix*, which tabulates actual classifications against predicted ones.

The model predicted loan status correctly 73% of the time—better than chance (50%). In the original dataset, 30% of the loans were bad, so guessing `GoodLoan` all the time would be 70% accurate (though not very useful). So you know that the model does better than random and somewhat better than obvious guessing.

Overall accuracy is not enough. You want to know what kinds of mistakes are being made. Is the model missing too many bad loans, or is it marking too many good loansas bad? *Recall* measures how many of the bad loans the model can actually find. *Precision* measures how many of the loans identified as bad really are bad. *False positive rate*measures how many of the good loans are mistakenly identified as bad. Ideally, you want the recall and the precision to be high, and the false positive rate to be low. What constitutes "high enough" and "low enough" is a decision that you make together with the other stakeholders. Often, the right balance requires some trade-off between recall and precision.

Presentation and documentation:

Once you have a model that meets your success criteria, you'll present your results toyour project sponsor and other stakeholders. You must also document the model for those in the organization who are responsible for using, running, and maintaining themodel once it has been deployed.

Different audiences require different kinds of information. Business-oriented audiences want to understand the impact of your findings in terms of business metrics. In our loan example, the most important thing to present to business audiencesis how your loan application model will reduce charge-offs (the money that the bank loses to bad loans). Suppose your model identified a set of bad loans that amountedto 22% of the total money lost to defaults. Then your presentation or executive summary should emphasize that the model can potentially reduce the bank's losses by that amount, as shown in figure 1.4.

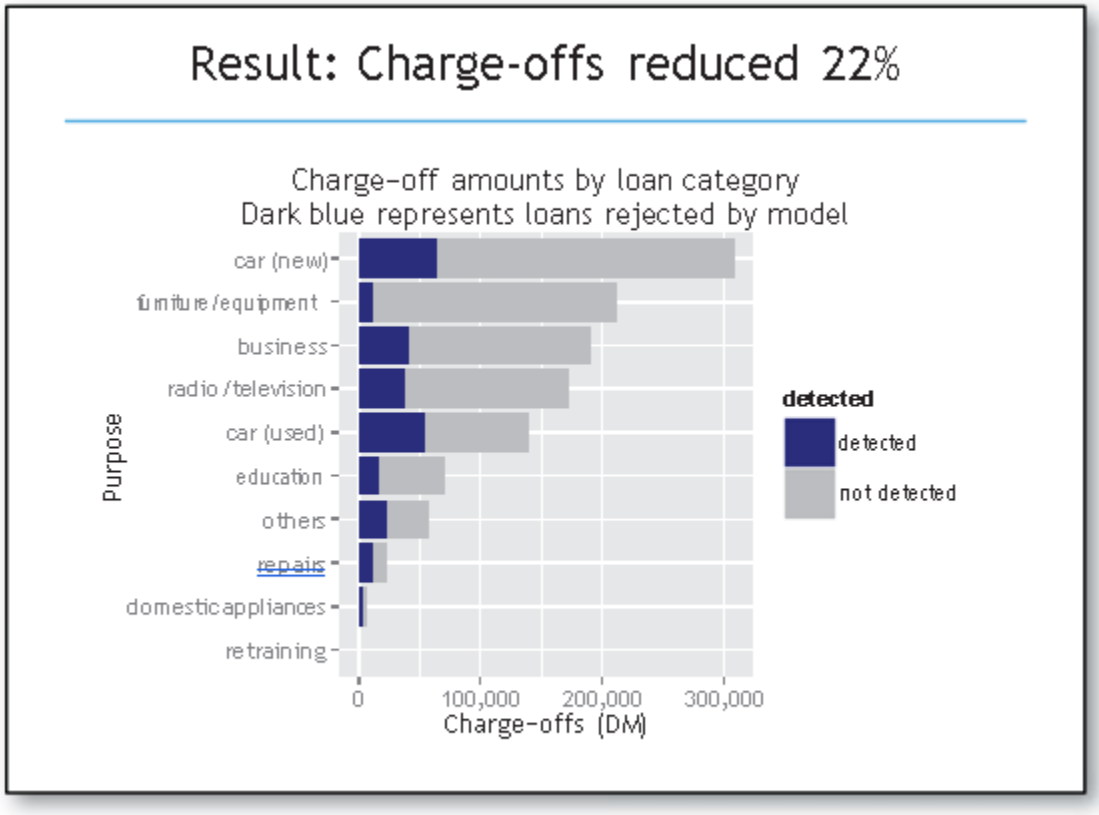


Figure 1.4 Notional slide from an executive presentation

You also want to give this audience your most interesting findings or recommendations, such as that new car loans are much riskier than used car loans, or that most losses are tied to bad car loans and bad equipment loans (assuming that the audiencedidn't already know these facts). Technical details of the model won't be as interesting to this audience, and you should skip them or only present them at a high level.

A presentation for the model's end users (the loan officers) would instead empha-size how the model will help them do their job better:

How should they interpret the model?

What does the model output look like?

If the model provides a trace of which rules in the decision tree executed, howdo they read that?

If the model provides a confidence score in addition to a classification, howshould they use the confidence score?

When might they potentially overrule the model?

Model deployment and maintenance

Finally, the model is put into operation. In many organizations this means the data scientist no longer has primary responsibility for the day-to-day operation of the model.But you still should ensure that the model will run smoothly and won't make disastrous unsupervised decisions. You also want to make sure that the model can be updated as its environment changes. And in many situations, the model will initiallybe deployed in a small pilot program. The test might bring out issues that you didn't anticipate, and you may have to adjust the model accordingly.

For example, you may find that loan officers frequently override the model in certain situations because it contradicts their intuition. Is their intuition wrong? Or is your model incomplete? Or, in a more positive scenario, your model may perform sosuccessfully that the bank wants you to extend it to home loans as well.

Applying Data-Driven Insights to Business and Industry:

The modern business world is absolutely deluged with data. That's because every line of business, every electronic system, every desktop computer, every laptop, every company-owned cellphone, and every employee is continually creating new business-related data as a natural and organic output of their work.

This data is structured or unstructured some of it is big and some of it is small, fast or slow; maybe it's tabular data, or video data, or spatial data, etc. But though there are many varieties and variations between the types of datasets produced, the challenge is only one — to extract data insights that add value to the organization when acted upon.

Benefits from Business centric Data Science:

Business is complex. Data science is complex. That's why, in all areas of business, it's extremely important to stay focused on the end goal. Ultimately, no matter what line of business you're in, true north is always the same: business profit growth. Whether you achieve that by creating greater efficiencies or by increasing sales rates and customer loyalty, the end goal is to create a more stable, solid profit-growth rate for your business. The following list describes some of the ways that you can use business-centric data science and business intelligence to help increase profits:

» Decrease financial risks. A business-centric data scientist can decrease financial risk in e-commerce business by using time series anomaly-detection methods for real-time fraud detection — to decrease Card-Not-Present fraud and to decrease the incidence of account takeovers, to take two examples.

» Increase the efficiencies of systems and processes. This is a business systems optimization function that's performed by both the business-centric data scientist and the business analyst. Both use analytics to optimize business processes, structures, and systems, but their methods and data sources differ. The end goal here should be to decrease needless resource expenditures and to increase return on investment for justified expenditures.

» Increase sales rates. To increase sales rates for your offerings, you can employ a business-centric data scientist

to help you find the best ways to upsell and cross-sell, increase customer loyalty, increase conversions in each layer of the funnel, and exact-target your advertising and discounts. It's likely that your business is already employing many of these tactics, but a business centric data scientist can look at all data related to the business and, from that, derive insights that supercharge these efforts.

Converting Raw Data into Actionable Insights with Data Analytics:

Turning your raw data into actionable insights is the first step in the progression from the data you've collected to something that actually benefits you. Business centric data scientists use data analytics to generate insights from raw data.

Types of Analytics-

Listed here, in order of increasing complexity, are the four types of data analytics you'll most likely encounter:

» Descriptive analytics: This type of analytics answers the question, "What happened?" Descriptive analytics are based on historical and current data. A business analyst or a business-centric data scientist bases modern-day business intelligence on descriptive analytics.

» Diagnostic analytics: You use this type of analytics to find answers to the question, "Why did this particular something happen?" or "What went wrong?" Diagnostic analytics are useful for deducing and inferring the success or failure of subcomponents of any data-driven initiative.

» Predictive analytics: Although this type of analytics is based on historical and current data, predictive analytics go one step further than descriptive analytics. Predictive analytics involve complex model-building and analysis in order to predict a future event or trend. In a business context, these analyses would be performed by the business-centric data scientist.

» Prescriptive analytics: This type of analytics aims to optimize processes, structures, and systems through informed action that's based on predictive analytics — essentially telling you what you should do based on an informed estimation of what will happen. Both business analysts and business-centric data scientists can generate prescriptive analytics, but their methods and data sources differ.

Ideally, a business should engage in all four types of data analytics, but prescriptive analytics is the most direct and effective means by which to generate value from data insights.

Common Challenges in Analytics:

Analytics commonly pose at least two challenges in the business enterprise. First, organizations often have difficulty finding new hires with specific skill sets that include analytics. Second, even skilled analysts often have difficulty communicating complex insights in a way that's understandable to management decision makers. To overcome these challenges, the organization must create and nurture a culture that values and accepts analytics products. The business must work to educate all levels of the organization so that management has a basic concept of analytics and the success that can be achieved by implementing them. Conversely, business centric data scientists must have a solid working knowledge about business in general and, in particular, a solid understanding of the business at hand. A strong business knowledge is one of the three main requirements of any business centric data scientist; the other two are a strong coding acumen and strong

quantitative analysis skills via math and statistical modeling.

Data wrangling –

Data wrangling is another important portion of the work that's required in order to convert data to insights. To build analytics from raw data, we need to use data wrangling — the processes and procedures that you use to clean and convert data from one format and structure to another so that the data is accurate and in the format that analytics tools and scripts require for consumption.

The following list highlights a few of the practices relevant to data wrangling:

» Data extraction: The business-centric data scientist must first identify which datasets are relevant to the problem at hand and then extract sufficient quantities of the data that's required to solve the problem. (This extraction process is commonly referred to as data mining.)

» Data preparation: Data preparation involves cleaning the raw data extracted through data mining and then converting it into a format that allows for a more convenient consumption of the data.

When preparing to analyze data, follow this 6-step process for data preparation:

1. Import. Read relevant datasets into your application.
2. Clean. Remove strays, duplicates, and out-of-range records
3. Transform. In this step, you treat missing values, deal with outliers, and scale your variables.
4. Process. Processing your data involves data parsing, recoding of variables, concatenation, and other methods of reformatting your dataset to prepare it for analysis.
5. Log in. In this step, you simply create a record that describes your dataset. This record should include descriptive statistics, information on variable formats, data source, collection methods, and more. Once you generate this log, make sure to store it in a place you'll remember, in case you need to share these details with other users of the processed dataset.
6. Back up. The last data preparation step is to store a backup of this processed dataset.

Distinguishing between Business Intelligence and Data Science:

Business-centric data scientists and business analysts who do business intelligence are like cousins: They both use data to work toward the same business goal, but their approach, technology, and function differ by measurable degrees. In the following sections, let us define, compare, and distinguish between business intelligence and business-centric data science.

Business intelligence, defined -The purpose of business intelligence is to convert raw data into business insights that business leaders and managers can use to make data-informed decisions. Business analysts use business intelligence tools to create decision-support products for business management decision making. If you want to build decision support dashboards, visualizations, or reports from complete medium-size sets of structured business data, you can use business intelligence tools and methods to help you.

Business intelligence (BI) is composed of

» Mostly internal datasets: By internal, we mean business data and information that's supplied by our organization's own managers and stakeholders.

» Tools, technologies, and skillsets: Examples here include online analytical processing, ETL (extracting, transforming, and loading data from one database into another), data warehousing, and information technology for business applications.

The kinds of data used in business intelligence- Insights that are generated in business intelligence (BI) are derived from standard-size sets of structured business data. BI solutions are mostly built off of transactional data — data that's generated during the course of a transaction event, like data generated during a sale or during a money transfer between bank accounts, for example. Transactional data is a natural byproduct of business activities that occur across an organization, and all sorts of inferences can be derived from it.

The following list describes the possible questions you can answer by using BI to derive insights from these types of data:

» Customer service: "What areas of business are causing the largest customer wait times?"

» Sales and marketing: "Which marketing tactics are most effective and why?"

» Operations: "How efficiently is the help desk operating? Are there any immediate actions that must be taken to remedy a problem there?"

» Employee performance: "Which employees are the most productive? Which are the least?"

Technologies and skillsets that are useful in business intelligence- To streamline BI functions, make sure that your data is organized for optimal ease of access and presentation. You can use multidimensional databases to help you. Unlike relational, or flat databases, multidimensional databases organize data into cubes that are stored as multidimensional arrays. If you want your BI staff to be able to work with source data as quickly and easily as possible, you can use multidimensional databases to store data in a cube rather than store the data across several relational databases that may or may not be compatible with one another.

This cubic data structure enables Online Analytical Processing (OLAP) — a technology through which you can quickly and easily access and use your data for all sorts of different operations and analyses. To illustrate the concept of OLAP, imagine that you have a cube of sales data that has three dimensions: time, region, and business unit. You can slice the data to view only one rectangle — to view one sales region, for instance. You can dice the data to view a smaller cube made up of some subset of time, region(s), and business unit(s). You can drill down or drill up to view either highly detailed or highly summarized data, respectively. And you can roll up, or total, the numbers along one dimension — to total business unit numbers, for example, or to view sales across time and region only

OLAP is just one type of data warehousing system — a centralized data repository that you can use to store and access your data. A more traditional data warehouse system commonly employed in business intelligence solutions is a data mart — a data storage system that you can use to store one particular focus area of data, belonging to only one line of business in the enterprise. Extract, transform, and load (ETL) is the process that you'd use to extract data, transform it, and load it into your database or data warehouse. Business analysts

generally have strong backgrounds and training in business and information technology. As a discipline, BI relies on traditional IT technologies and skills.

Defining Business-Centric Data Science- Business-centric data science is multidisciplinary and incorporates the following elements:

» Quantitative analysis: Can be in the form of mathematical modeling, multivariate statistical analysis, forecasting, and/or simulations. The term multivariate refers to more than one variable. A multivariate statistical analysis is a simultaneous statistical analysis of more than one variable at a time.

» Programming skills: You need the necessary programming skills to analyze raw data and to make this data accessible to business users.

» Business knowledge: You need knowledge of the business and its environment so that you can better understand the relevancy of your findings.

Data science is a pioneering discipline. Data scientists often employ the scientific method for data exploration, hypotheses formation, and hypothesis testing (through simulation and statistical modeling). Business-centric data scientists generate valuable data insights, often by exploring patterns and anomalies in business data.

Data science in a business context is commonly composed of

» Internal and external datasets: Data science is flexible. You can create business data mash-ups from internal and external sources of structured and unstructured data fairly easily. (A data mash-up is combination of two or more data sources that are then analyzed together in order to provide users with a more complete view of the situation at hand.

» Tools, technologies, and skillsets: Examples here could involve using cloud-based platforms, statistical and mathematical programming, machine learning, data analysis using Python and R, and advanced data visualization.

Like business analysts, business-centric data scientists produce decision-support products for business managers and organizational leaders to use. These products include analytics dashboards and data visualizations, but generally not tabular data reports and tables.

Kinds of data that are useful in business-centric data science -You can use data science to derive business insights from standard-size sets of structured business data (just like BI) or from structured, semi-structured, and unstructured sets of big data. Data science solutions are not confined to transactional data that sits in a relational database; you can use data science to create valuable insights from all available data sources.

These data sources include

» Transactional business data: A tried-and-true data source, transactional business data is the type of structured data used in traditional BI and it includes management data, customer service data, sales and marketing data, operational data, and employee performance data.

» Social data related to the brand or business: A more recent phenomenon, the data covered by this rubric includes the unstructured data generated through emails, instant messaging, and social networks such as Twitter, Facebook, LinkedIn, Pinterest, and Instagram.

» Machine data from business operations: Machines automatically generate this unstructured data, like SCADA data, machine data, or sensor data. The acronym SCADA refers to Supervisory Control and Data Acquisition. SCADA systems are used to control remotely operating mechanical systems and equipment. They generate data that is used to monitor the operations of machines and equipment.

» Audio, video, image, and PDF file data: These well-established formats are all sources of unstructured data.

Differentiating between Business Intelligence and Business-Centric Data Science—

The similarities between BI and business-centric data science are glaringly obvious; it's the differences that most people have a hard time discerning. The purpose of both BI and business-centric data science is to convert raw data into actionable insights that managers and leaders can use for support when making business decisions.

BI and business-centric data science differ with respect to approach. Although BI can use forward-looking methods like forecasting, these methods are generated by making simple inferences from historical or current data. In this way, BI extrapolates from the past and present to infer predictions about the future. It looks to present or past data for relevant information to help monitor business operations and to aid managers in short- to medium-term decision making.

In contrast, business-centric data science practitioners seek to make new discoveries by using advanced mathematical or statistical methods to analyze and generate predictions from vast amounts of business data. These predictive insights are generally relevant to the long-term future of the business. The business centric data scientist attempts to discover new paradigms and new ways of looking at the data to provide a new perspective on the organization, its operations, and its relations with customers, suppliers, and competitors. Therefore, the business-centric data scientist must know the business and its environment. She must have business knowledge to determine how a discovery is relevant to a line of business or to the organization at large.

Other prime differences between BI and business-centric data science are

» Data sources: BI uses only structured data from relational databases, whereas business-centric data science may use structured data and unstructured data, like that generated by machines or in social media conversations.

» Outputs: BI products include reports, data tables, and decision-support dashboards, whereas business-centric data science products involve either dashboard analytics or another type of advanced data visualization, but rarely tabular data reports. Data scientists generally communicate their findings through words or data visualizations, but not tables and reports. That's because the source datasets from which data scientists work are generally more complex than a typical business manager would be able to understand.

» Technology: BI runs off of relational databases, data warehouses, OLAP, and ETL technologies, whereas business-centric data science often runs off of data from data-engineered systems that use Hadoop, MapReduce, or massively parallel processing.» Expertise: BI relies heavily on IT and business technology expertise, whereas business-centric data science relies on expertise in statistics, math, programming, and business.

UNIT-II

Using Data Science to Extract meaning from Data – Machine learning — Modeling with instances

Data science tools environment – Python – overview — Setting up Data science toolbox

Using Data Science to Extract meaning from Data:

Machine learning:

Defining Machine Learning and Its Processes- Machine learning is the practice of applying algorithmic models to data, in an iterative manner, so that your computer discovers hidden patterns or trends that you can use to make predictions. It's also called algorithmic learning. Machine learning has a vast and ever-expanding assortment of use cases, including

» Real-time Internet advertising

» Internet marketing personalization

» Internet search

» Spam filtering

» Recommendation engines

» Natural language processing and sentiment analysis

» Automatic facial recognition

» Customer churn prediction

» Credit score modeling

» Survival analysis for mechanical equipment

steps of the machine learning process -Three main steps are involved in machine learning: setting up, learning, and application.

Setting up involves acquiring data, preprocessing it, feature selection (the process of selecting the most appropriate variables for the task at hand), and breaking the data into training and test datasets. You use the training data to train the model, and the test data to test the accuracy of the model's predictions.

The learning step involves model experimentation, training, building, and testing.

The application step involves model deployment and prediction.

A good rule of thumb when breaking data into test and training sets is to apply random sampling to take two-thirds of the original dataset to use as data to train the model. Use the remaining one-third of the data as test data, for evaluating the model's predictions.

machine learning terms-

» Instance: The same as a row (in a data table), an observation (in statistics), and a data point. Machine learning practitioners are also known to call an instance a case.

» Feature: The same as a column or field (in a data table) and a variable (in statistics). In regression methods, a feature is also called an independent variable (IV).

» Target variable: The same as a predictant or dependent variable (DV) in statistics.

Learning Styles- There are three main styles within machine learning: supervised, unsupervised, and semi supervised. Supervised and unsupervised methods are behind most modern machine learning applications, and semi supervised learning is an up-and-coming star.

Learning with supervised algorithms: Supervised learning algorithms require that input data has labeled features. These algorithms learn from known features of that data to produce an output model that successfully predicts labels for new incoming, unlabeled data points. You use supervised learning when you have a labeled dataset composed of historical values that are good predictors of future events. Use cases include survival analysis and fraud detection, among others. Logistic regression is a type of supervised learning algorithm.

Learning with unsupervised algorithms: Unsupervised learning algorithms accept unlabeled data and attempt to group observations into categories based on underlying similarities in input features, as shown in Figure 4-1. Principal component analysis, k-means clustering, and singular value decomposition are all examples of unsupervised machine learning algorithms. Popular use cases include recommendation engines, facial recognition systems, and customer segmentation.

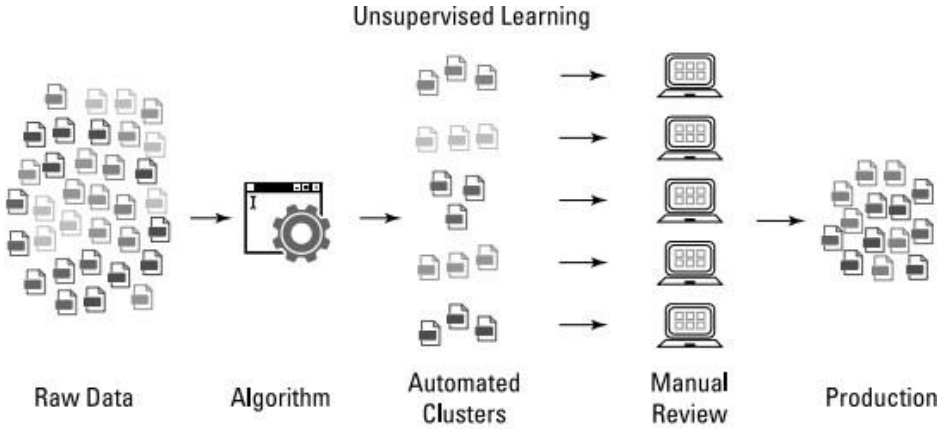


FIGURE 4-1: Unsupervised machine learning breaks down unlabeled data into subgroups.

Learning with reinforcement: Reinforcement learning (or semi supervised learning) is a behavior-based learning model. It is based on a mechanic similar to how humans and animals learn. The model is given "rewards" based on how it behaves, and it subsequently learns to maximize the sum of its rewards by adapting the decisions it makes to earn as many rewards as possible. Reinforcement learning is an up-and-coming concept in data science.

Selecting algorithms based on function- When you need to choose a class of machine learning algorithms, it's helpful to consider each model class based on its functionality. For the most part, algorithmic functionality falls into the categories shown in Figure 4-2.

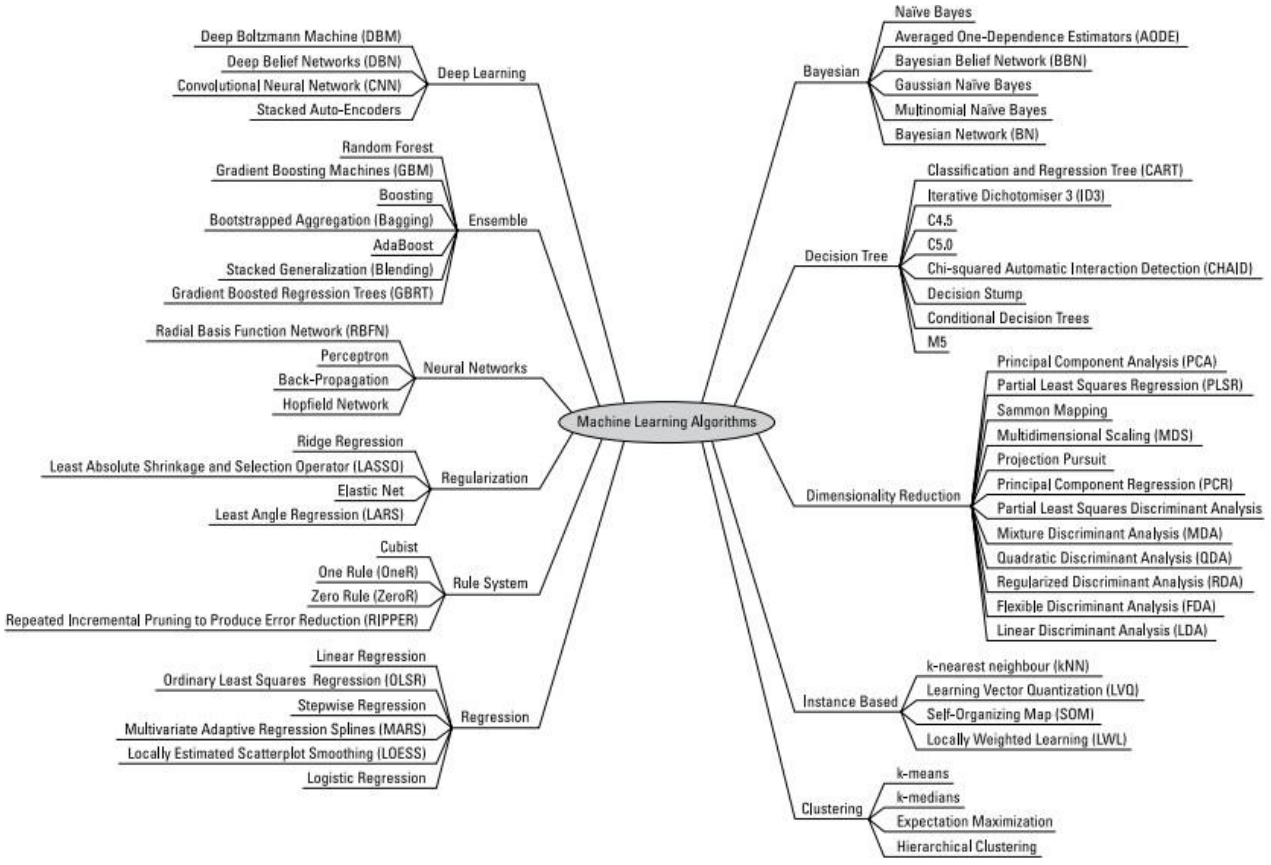


FIGURE 4-2:
Machine learning algorithms can be broken down by function.

» Regression algorithm: You can use this type of algorithm to model relationships between features in a dataset.

» Association rule learning algorithm: This type of algorithm is a rule-based set of methods that you can use to discover associations between features in a dataset.

» Instance-based algorithm: If you want to use observations in your dataset to classify new observations based on similarity, you can use this type. To model with instances, you can use methods like k-nearest neighbor classification

. » Regularizing algorithm: You can use regularization to introduce added information as a means by which to prevent model overfitting or to solve an ill-posed problem.

» Naïve Bayes method: If you want to predict the likelihood of an event occurring based on some evidence in your data, you can use this method, based on classification and regression.

» Decision tree: A tree structure is useful as a decision-support tool. You can use it to build models that predict for potential fallouts that are associated with any given decision.

» Clustering algorithm: You can use this type of unsupervised machine learning method to uncover subgroups within an unlabeled dataset. k-means clustering and hierarchical clustering are the clustering algorithms.

» Dimension reduction method: If you're looking for a method to use as a filter to remove redundant

information, noise, and outliers from your data, consider dimension reduction techniques such as factor analysis and principal component analysis.

» Neural network: A neural network mimics how the brain solves problems, by using a layer of interconnected neural units as a means by which to learn, and infer rules, from observational data. It's often used in image recognition and computer vision applications.

» Deep learning method: This method incorporates traditional neural networks in successive layers to offer deep-layer training for generating predictive outputs.

» Ensemble algorithm: You can use ensemble algorithms to combine machine learning approaches to achieve results that are better than would be available from any single machine learning method on its own.

Modeling with instances:

Data scientists use classification methods to help them build predictive models that they can then use to forecast the classification of future observations. Classification is a form of supervised machine learning: The classification algorithm learns from labeled data. Data labels make it easier for your models to make decisions based on the logic rules you've defined.

Instance-based learning classifiers are supervised, lazy learners — they have no training phase, and they simply memorize training data, in-memory, to predict classifications for new data points. This type of classifier looks at instances — observations within a dataset — and, for each new observation, the classifier searches the training data for observations that are most similar, and then classifies the new observation based on its similarity to instances in the training set.

Instance-based classifiers include

» k-nearest neighbor (kNN)

» Self-organizing maps

» Locally weighted learning

If you're unsure about your dataset's distribution, instance-based classifiers might be a good option, but first make sure that you know their limitations.

These classifiers are not well-suited for

» Noisy data (data with unexplainable random variation)

» Datasets with unimportant or irrelevant features

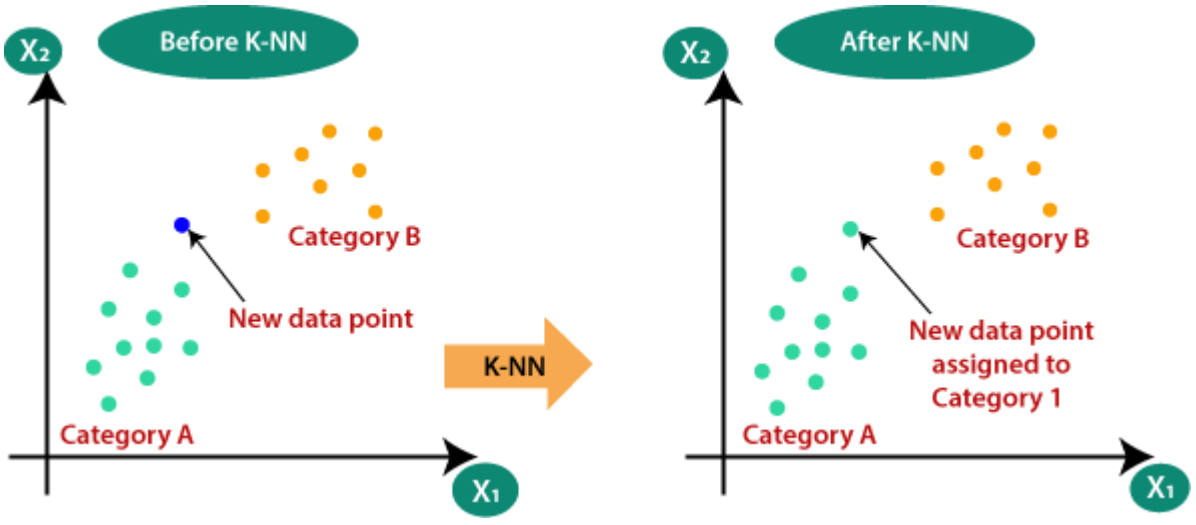
» Datasets with missing values

- **K-Nearest Neighbor Algorithm**- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



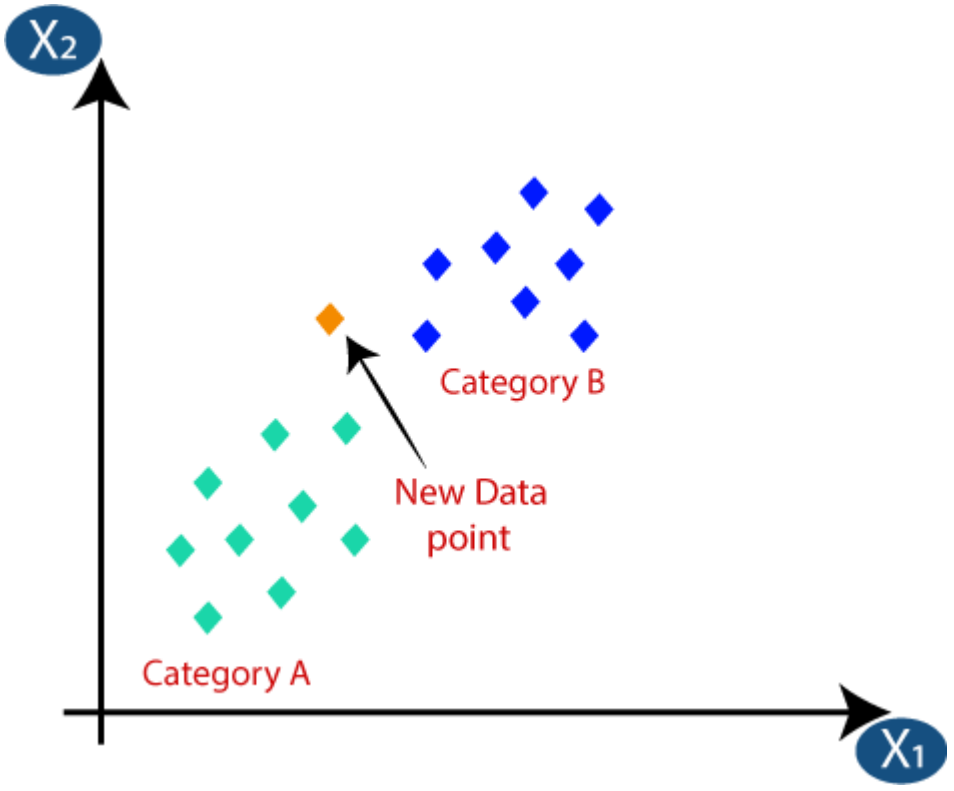
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

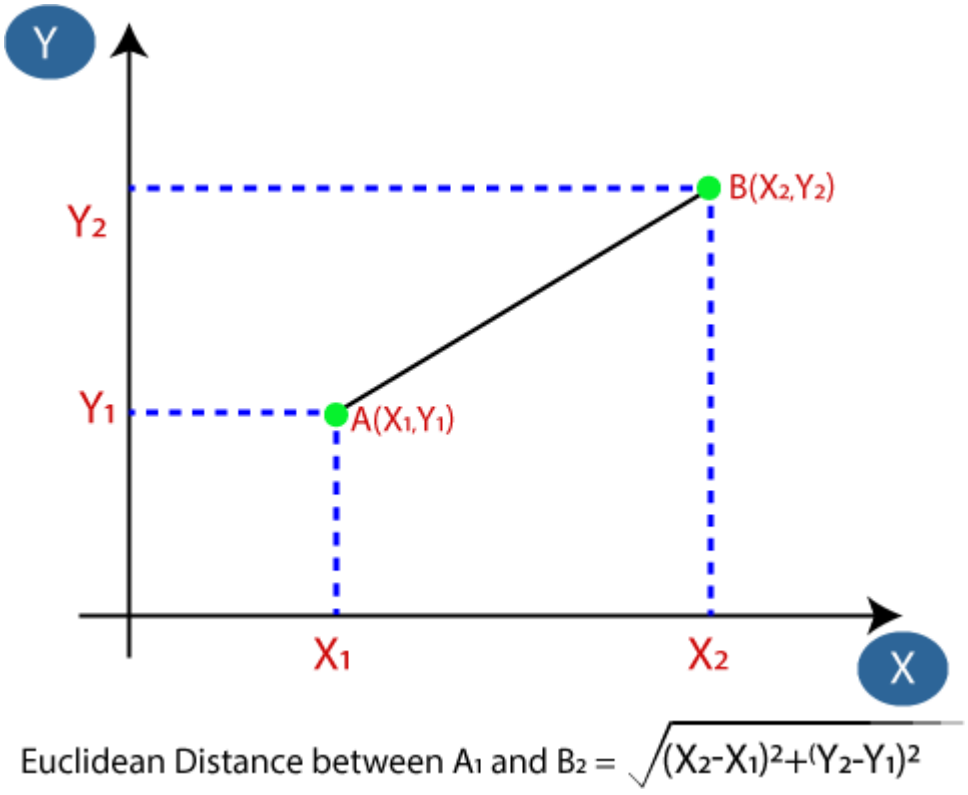
- **Step-1:** Select the number K of the neighbors

- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

Self-Organizing Map (SOM):

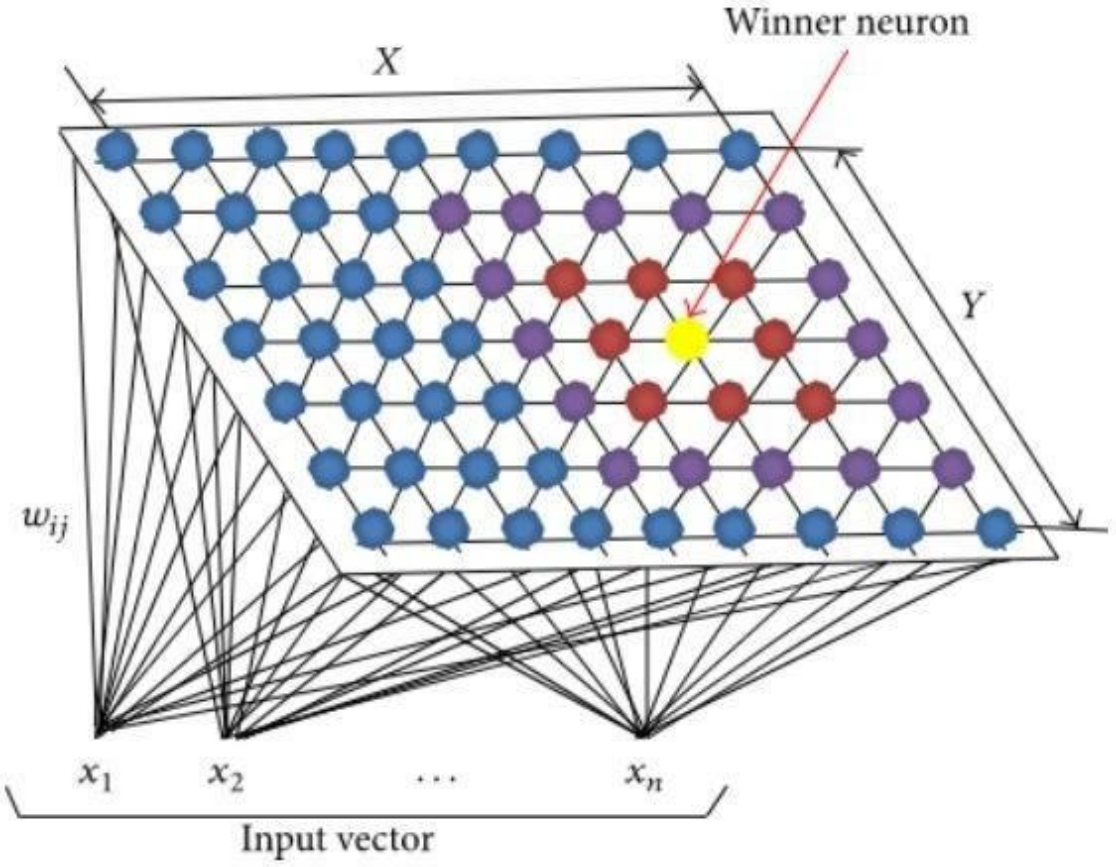
Self-Organizing Map (or Korhonen Map or SOM) is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s. It follows an unsupervised learning approach and trained its network through a competitive learning algorithm.

SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation.

The goal of the technique is to reduce dimensions and detect features. The maps help to visualize high-dimensional data. It represents the multidimensional data in a two- dimensional space using the self-organizing neural networks. The technique is used for data mining, face recognition, pattern recognition, speech analysis, industrial and medical diagnostics, anomalies detection.

The self-organizing maps structure is a lattice of neurons. In the grid, X and Y coordinate, each node has a specific topological position and comprises a vector of weights of the same dimension as that of the input variables. The input training data is a set of vectors of n dimensions: x1, x2, x3…xn, and each node consists of a corresponding weight vector W of n dimensions: w1, w2, w3… wn

In the illustration below, the lines between the nodes are simply a representation of adjacency and not signifying any connection as in other neural networks.

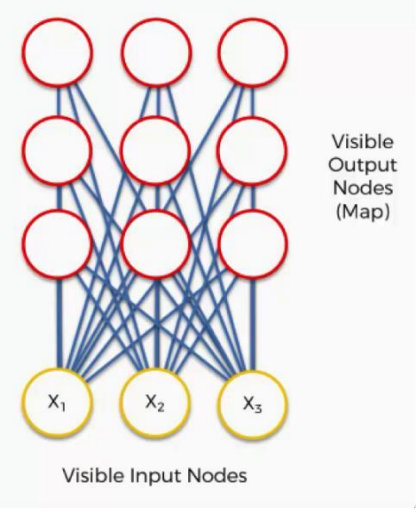


Self-Organizing Maps are a lattice or grid of neurons (or nodes) that accepts and responds to a set of input signals. Each neuron has a location, and those that lie close to each other represent clusters with similar properties. Therefore, each neuron represents a cluster learned from the training.

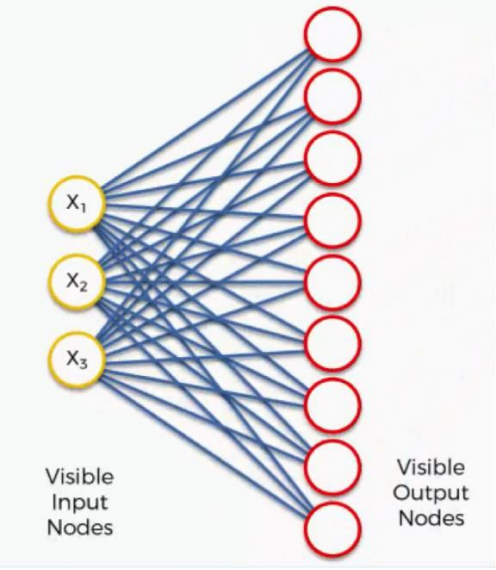
In SOMs, the responses are compared and a 'winning' neuron is selected from the lattice. This selected neuron is activated together with the neighborhood neurons. SOMs return a two-dimensional map for any number of indicators as output where there are no lateral connections between output nodes.

The underlying idea of the SOMs training process is to examine every node and find the one node whose weight is most like the input vector. The training is carried out in a few steps and over many iterations. Let's see this in detail below:

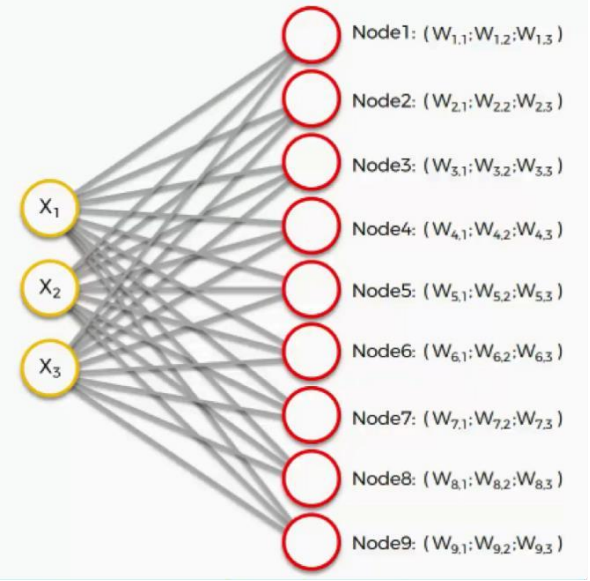
We have three input signals x1, x2, and x3. This input vector is chosen at random from a set of training data and presented to the lattice. So, we have a lattice of neurons in the form of a 3*3 two-dimensional array having nine nodes with three rows and three columns depicted like below:



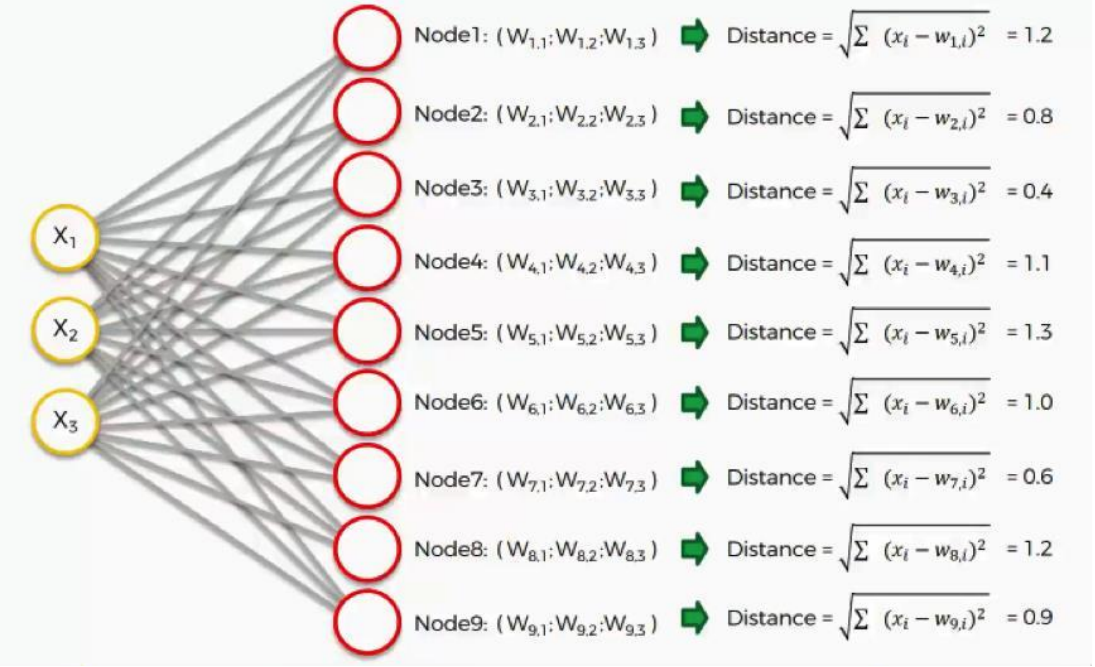
For understanding purposes, visualize the above 3*3 matrix as below:



Each node has some random values as weights. These weights are not of the neural network as shown as:



From these weights, can calculate the Euclidean distance as:



The training process of SOMs follows:

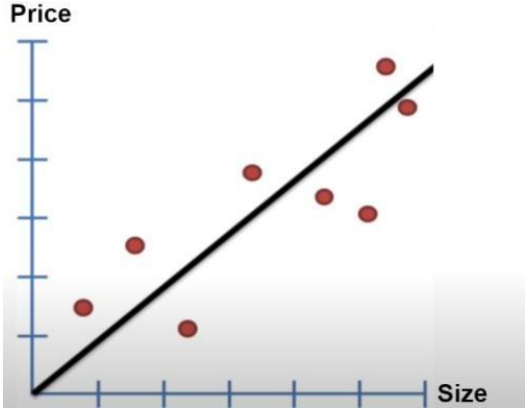
1. Firstly, randomly initialize all the weights.
2. Select an input vector x = [x1, x2, x3, … , xn] from the training set.

3. Compare x with the weights wj by calculating Euclidean distance for each neuron j. The neuron having the least distance is declared as the winner. The winning neuron is known as Best Matching Unit (BMU)
4. Update the neuron weights so that the winner becomes and resembles the input vector x.
5. The weights of the neighbouring neuron are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered. The parameters adjusted are learning rate and neighbourhood function.
6. Repeat from step 2 until the map has converged for the given iterations or there are no changes observed in the weights.

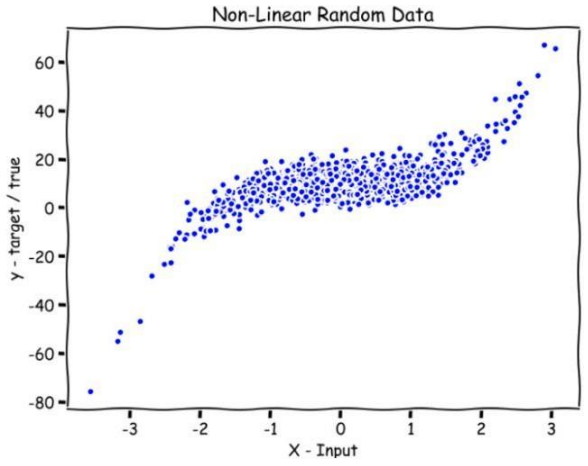
Locally weighted learning:

Locally Weighted Regression is a non-parametric algorithm, unlike a typical linear regression algorithm which is a parametric algorithm. A parametric algorithm is an algorithm that doesn't need to retain the training data when we need to make predictions.

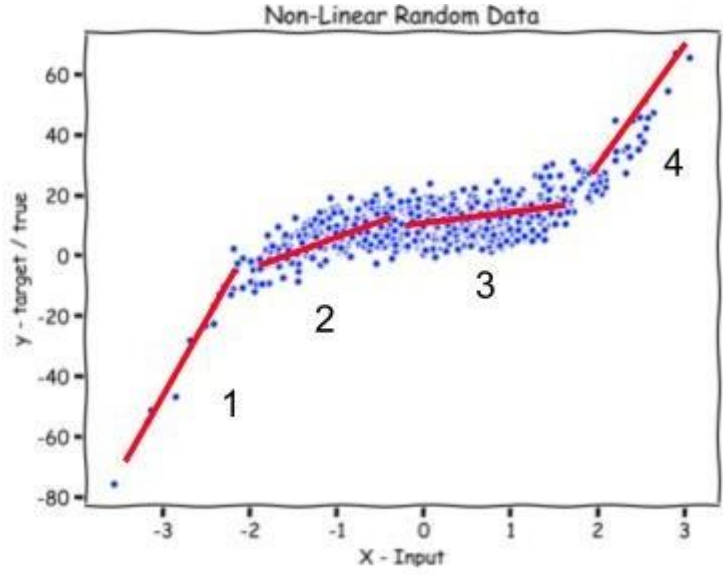
In the traditional linear regression algorithm, we aim to fit a general line on the given training dataset and predict some parameters that can be generalized for any input to make correct predictions.



Suppose, we have a non-linear dataset, as shown in the image below:



A standard line wouldn't fit entirely into this type of dataset. In such a case, we have to introduce a locally weighted algorithm that can predict a very close value to the actual value of a given query point. So another thought coming to our mind is, let's break the given dataset into a smaller dataset and say we have multiple smaller lines that can fit the smaller dataset. Together, these multiple lines fit the entire dataset. To understand better, look at the diagram below.



Look at the 4 small lines that have been tried to fit smaller datasets and fit the entire dataset together. But now, the question arises of how to select the best line that can be chosen to predict the output for a given query point.

Assigning weights to a line-We assume that there is a corresponding weight for that particular line given a query point. The one with the greatest weight will be the best fit line and will be used for predicting the output of a given query point.

We define a weight function which is given as,

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

x(i) = any general training point,
x = given query point and,
r = is bandwidth parameter
The bandwidth parameter controls how quickly the weight should fall with the distance of the point with the training point. In simpler terms, it controls the width of how varied the data is.

we see that if we are given a query point close to a general training point, the weight would be close to 1 as the

difference is close to 0. Similarly, if it's too far from the training point, then the weight would be close to 0.

Hence we choose this as our weight function.

Data science tools environment:

Python – overview:

The main data types in Python and the general forms they take are described in this list:

» Numbers: Plain old numbers

» Strings: '. . .' or ". . ."

» Lists: [. . .] or [. . ., . . ., . . .]

» Tuples: (. . .)

» Sets: Rarely used

» Dictionaries: {'Key': 'Value', . . .}.

Numbers and strings are the most basic data types. You can incorporate them inside other, more complicated data types. All Python data types can be assigned to variables. In Python, numbers, strings, lists, tuples, sets, and dictionaries are classified as both object types and data types.

Numbers in Python-

The numbers data type represents numeric values that you can use to handle all types of mathematical operations. Numbers come in the following types:

» Integers: A whole-number format

» Long: A whole-number format with an unlimited digit size

» Float: A real-number format, written with a decimal point

» Complex: An imaginary-number format, represented by the square root of –1

Strings in Python -Strings are the most often used data type in Python — and in every other programming language, for that matter. Simply put, a string consists of one or more characters written inside single or double quotes.

The following code represents a string:

```
>>> variable1='This is a sample string'
>>> print(variable1)
This is a sample string
```

In this code snippet, the string is assigned to a variable, and the variable subsequently acts like a storage container for the string value. To print the characters contained inside the variable, simply use the predefined function, print.

Python coders often refer to lists, tuples, sets, and dictionaries as data structures rather than data types.

Data structures are basic functional units that organize data so that it can be used efficiently by the program or application you're working with. Lists, tuples, sets, and dictionaries are data structures, but keep in mind that they're still composed of one or more basic data types (numbers and/or strings, for example).

Lists in Python -A list is a sequence of numbers and/or strings. To create a list, you simply enclose the elements of the list (separated by commas) within square brackets. Here's an example of a basic list:

```
>>> variable2=["ID","Name","Depth","Latitude","Longitude"]
>>> depth=[0,120,140,0,150,80,0,10]
>>> variable2[3]
'Latitude'
```

Every element of the list is automatically assigned an index number, starting from 0. You can access each element using this index, and the corresponding value of the list will be returned. If you need to store and analyze long arrays of data, use lists — storing your data inside a list makes it fairly easy to extract statistical information.

The following code snippet is an example of a simple computation to pull the mean value from the elements of the depth list created in the preceding code example:

```
>>> sum(depth)/len(depth)
62.5
```

In this example, the average of the list elements is computed by first summing up the elements, via the sum function, and then dividing them by the number of the elements contained in the list. See, it's as simple as 1 - 2-3!

Tuples in Python- Tuples are just like lists, except that you can't modify their content after you create them. Also, to create tuples, you need to use normal brackets instead of squared ones. Here's an example of a tuple:

```
>>> depth=(0,120,140,0,150,80,0,10)
```

In this case, you can't modify any of the elements, like you would with a list. If you want to ensure that your data stays in a read-only format, use tuples.

Sets in Python- A set is another data structure that's similar to a list. In contrast to lists, however, elements of a set are unordered. This disordered characteristic of a set makes it impossible to index, so it's not a commonly used data type.

Dictionaries in Python -Dictionaries are data structures that consist of pairs of keys and values. In a dictionary, every value corresponds to a certain key, and consequently, each value can be accessed using that key.

The following code snippet shows a typical key/ value pairing:

```
>>> variable4={"ID":1,"Name":"Valley City","Depth":0, "Latitude":49.6, "Longitude":-98.01}
>>> variable4["Longitude"]
-98.01
```

Loops in Python- You can use looping to execute the same block of code multiple times for a sequence of items. Consequently, rather than manually access all elements one by one, you simply create a loop to automatically iterate (or pass through in successive cycles) each element of the list.

You can use two types of loops in Python: the for loop and the while loop. The most often used looping technique is the for loop — designed especially to iterate through sequences, strings, tuples, sets, and dictionaries. The following code snippet illustrates a for loop iterating through the variable2 list :

```
>>>variable2=["ID","Name","Depth","Latitude","Longitude"]
```

```
>>> for element in variable2:print(element)
ID
Name
Depth
Latitude
Longitude
```

The other available looping technique in Python is the while loop. Use a while loop to perform actions while a given condition is true

Functions in python- In Python, a function is a group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

Example of a function:

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

How to call a function in python?

Once we have defined a function, we can call it from another function, program, or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

NumPy module in python:

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy is a Python package. It stands for ‘Numerical Python’. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

The basic ndarray is created using an array function in NumPy as follows-

numpy.array

It creates a ndarray from any object exposing an array interface, or from any method that returns an array.

Take a look at the following examples to understand better.

Example 1

```
import numpy as np
a = np.array([1,2,3])
print a
```

The output is as follows –

```
[1, 2, 3]
```

Example 2

```
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print a
```

The output is as follows −

```
[[1, 2]
 [3, 4]]
```

various array attributes of NumPy:

ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print a.shape
```

The output is as follows −(2, 3)

ndarray.ndim

This array attribute returns the number of array dimensions.

numpy.itemsize

This array attribute returns the length of each element of array in bytes.

Example 1

dtype of array is int8 (1 byte)

import numpy as np

x = np.array([1,2,3,4,5], dtype = np.int8)

print x.itemsize

The output is as follows −

1

Setting up Data science toolbox:

The Data Science Toolbox is a virtual environment that allows you to get started doing data science in minutes. The default version comes with commonly used software for data science, including the Python scientific stack and R together with its most popular packages. Additional software and data bundles are easily installed.
There are two ways to set up the Data Science Toolbox: (1) installing it locally using VirtualBox and Vagrant or (2) launching it in the cloud using Amazon Web Services. Both ways result in exactly the same environment.
The easiest way to install the Data Science Toolbox is on your local machine. Because the local version of the Data Science Toolbox runs on top of VirtualBox and Vagrant, it can be installed on Linux, Mac OS X, and Microsoft Windows.

Step 1: Download and Install VirtualBox
Browse to the VirtualBox (Oracle, 2014) download page and download the appropriate binary for your operating system. Open the binary and follow the installation instructions.
Step 2: Download and Install Vagrant
Similar to Step 1, browse to the Vagrant (HashiCorp, 2014) download page and download the appropriate binary. Open the binary and follow the installation instructions. If you already have Vagrant installed, please make sure that it's version 1.5 or higher.
Step 3: Download and Start the Data Science Toolbox
Open a terminal (known as the Command Prompt or PowerShell in Microsoft Win- dows). Create a directory, e.g., MyDataScienceToolbox, and navigate to it by typing:

```
$ mkdir MyDataScienceToolbox
$ cd MyDataScienceToolbox
```

In order to initialize the Data Science Toolbox, run the following command:

```
$ vagrant init data-science-toolbox/data-science-at-the-command-line
```

This creates a file named Vagrant file. This is a configuration file that tells Vagrant how to launch the virtual machine. This file contains a lot of lines that are commented out.

A minimal version is shown in Example 2-1.
Example 2-1. Minimal configuration for Vagrant

```
Vagrant.configure(2) do |config|
  config.vm.box = "data-science-toolbox/data-science-at-the-command-line"
end
```

By running the following command, the Data Science Toolbox will be downloaded and booted:

```
$ vagrant up
```

If everything went well, then you now have a Data Science Toolbox running on your local machine.

Step 4: Log In (on Linux and Mac OS X)
If you are running Linux, Mac OS X, or some other Unix-like operating system, you can log in to the Data Science Toolbox by running the following command in a terminal:
$ vagrant ssh
After a few seconds, you will be greeted with the following message:
Welcome to the Data Science Toolbox for Data Science at the Command Line

Step 4: Log In (on Microsoft Windows)
If you are running Microsoft Windows, you need to either run Vagrant with a graphical user interface (refer back to Step 2 on how to set that up) or use a third-party application in order to log in to the Data Science Toolbox. For the latter, we recommend PuTTY. Browse to the PuTTY download page and download putty.exe. Run PuTTY, and enter the following values:
• Host Name (or IP address): 127.0.0.1
• Port: 2222
• Connection type: SSH

If you want, you can save these values as a session by clicking the Save button, so that you do not need to enter these values again. Click the Open button and enter vagrant for both the username and the password.

Step 5: Shut Down or Start Anew
The Data Science Toolbox can be shut down by running the following command from the same directory as you ran vagrant up:
$ vagrant halt

In case you wish to get rid of the Data Science Toolbox and start over, you can type:

$ vagrant destroy

Then, return to the instructions for Step 3 to set up the Data Science Toolbox again.

UNIT-III

Usage of Data science tools environment - Essential concepts and tools—Obtaining Data — Managing your Data workflow – Drake

Techniques using Python Tools- k–Nearest Neighbors – Naive Bayes

Essential concepts and tools:

we now discuss several concepts and tools that you will need to know in order to feel comfortable doing data science at the command line.

The Environment:

So, you've just logged into a brand-new environment. Before we do anything, it's worthwhile to get a high-level understanding of this environment. The environment is roughly defined by four layers, which we briefly discuss from the top down:

1. Command-line tools: First and foremost, there are the command-line tools that you work with. We use them by typing their corresponding commands. There are different types of command-line tools. Examples of tools are ls, cat, and jq.
2. Terminal: The terminal, which is the second layer, is the application where we type our commands in. If you see the following text:

```
$ seq 3
1
2
3
```

then you would type seq 3 into your terminal and press <Enter>.

You do not type the dollar sign. It's just there to serve as a prompt and let you know that you can type this command. The text below seq 3 is the output of the command.

3. Shell: The third layer is the shell. Once we have typed in our command and pressed <Enter> , the terminal sends that command to the shell. The shell is a program that interprets the command. The Data Science Toolbox uses **Bash** as the shell, but there are many others available. Once you have become a bit more proficient at the command line, you may want to look into a shell called the Z shell. It offers many additional features that can increase your productivity at the command line.

4. Operating system: The fourth layer is the operating system, which is GNU/Linux in our case. Linux is the name of the kernel, which is the heart of the operating system. The kernel is in direct contact with the CPU, disks, and other hardware. The kernel also executes our command-line tools. GNU, which is a recursive acronym for GNU's Not Unix, refers to a set of basic tools. The Data Science Toolbox is based on a particular Linux distribution called Ubuntu.

Executing a Command-Line Tool

Now that you have an understanding of the environment, it's high time that you try out some commands. Type the following in your terminal (without the dollar sign) and press <Enter>:

```
$ pwd
/home/vagrant
```

This is as simple as it gets. You just executed a command that contained a single command-line tool. The command-line tool pwd prints the name of the directory where you currently are. By default, when you log in, this is your home directory. You can view the contents of this directory with ls :

```
$ ls
book
```

The command-line tool cd, which is a Bash builtin, allows you to navigate to a different directory:

```
$ cd book/ch02/
$ cd data
$ pwd
/home/vagrant/book/ch02/data
$ cd ..
$ pwd
/home/vagrant/book/ch02/
```

The part after cd specifies to which directory you want to navigate to. Values that come after the command are called command-line arguments or options. The two dots refer to the parent directory.

Five Types of Command-Line Tools-

each command-line tool is one of the following five types:

- A binary executable
- A shell built-in
- An interpreted script
- A shell function
- An alias

It's good to know the difference between the types. The command-line tools that come pre-installed with the Data Science Toolbox mostly comprise the first two types (binary executable and shell builtin). The other three types (interpreted script, shell function, and alias) allow us to further build up our own data science toolbox2 and become more efficient and more productive data scientists.

1. **Binary executable** -Binary executables are programs in the classical sense. A binary executable is created by compiling source code to machine code. This means that when you open the file in a text editor you cannot read its source code.
2. **Shell builtin** -Shell builtins are command-line tools provided by the shell, which is Bash in our case. Examples include cd and help. These cannot be changed. Shell builtins may differ between shells. Like binary executables, they cannot be easily inspected or changed.
3. **Interpreted script** -An interpreted script is a text file that is executed by a binary executable. Examples include: Python, R, and Bash scripts. One great advantage of an interpreted script is that you can read and change it.
4. **Shell function** -A shell function is a function that is executed by the shell itself; in our case, it is executed by Bash. They provide similar functionality to a Bash script, but they are usually (though not necessarily) smaller than scripts.
5. **Alias**- Aliases are like macros. If you often find yourself executing a certain command with the same parameters (or a part of it), you can define an alias for this. Aliases are also very useful when you continue to misspell a certain command. The following commands define two aliases:

```
$ alias l='ls -1 --group-directories-first'
$ alias moer=more
```

Now, if you type the following on the command line, the shell will replace each alias it finds with its value:

```
$ cd ~
$ l
book
```

Combining Command-Line Tools-

the command-line tool **grep** can filter lines, **wc** can count lines, and **sort** can sort lines. The power of the command line comes from its ability to combine these small yet powerful command-line tools. The most common way of combining command-line tools is through a so-called pipe. The output from the first tool is passed to the second tool. There are virtually no limits to this.

Consider, for example, the command-line tool **seq**, which generates a sequence of numbers. Let's generate a sequence of five numbers:

```
$ seq 5
1
2
3
4
5
```

The output of a command-line tool is by default passed on to the terminal, which displays it on our screen. We can pipe the output of seq to a second tool, called grep, which can be used to filter lines. Imagine that we only want to see numbers that contain a "3." We can combine seq and grep as follows:

```
$ seq 30 | grep 3
3
13
23
30
```

And if we wanted to know how many numbers between 1 and 100 contain a "3", we can use wc, which is very good at counting things:

```
$ seq 100 | grep 3 | wc -l
19
```

The -l option specifies that wc should only output the number of lines. By default it also returns the number of characters and words. You may start to see that combining command-line tools is a very powerful concept.

Redirecting Input and Output:

by default, the output of the last command-line tool in the pipeline is outputted to the terminal. You can also save this output to a file. This is called output redirection, and works as follows:

$ cd ~/book/ch02

$ seq 10 > data/ten-numbers

Here, we save the output of the **seq** tool to a file named **ten-numbers** in the directory **~/book/ch02/data.**

If this file does not exist yet, it is created. If this file already did exist, its contents would have been overwritten.

You can also append the output to a file with >>, meaning the output is put after the original contents:

$ echo -n "Hello" > Hello

$ echo " World" >> Hello World

The tool **echo** simply outputs the value you specify. The -n option specifies that echo should not output a trailing newline.

Working with Files:

As data scientists, we work with a lot of data, which is often stored in files. It's important to know how to work with files (and the directories they live in) on the command line. Every action that you can do using a graphical user interface, you can do with command-line tools (and much more).

You have already seen how we can create new files by redirecting the output with either > or >>. In case you need to move a file to a different directory, you can use mv.

$ mv hello-world data

You can also rename files with mv:

$ cd data

$ mv hello-world old-file

You can also rename or move entire directories. In case you no longer need a file, you delete (or remove) it with rm.

$ rm old-file

In case you want to remove an entire directory with all its contents, specify the -r option, which stands for recursive:

$ rm -r ~/book/ch02/data/old

Obtaining Data:

Copying Local Files to the Data Science Toolbox:

A common situation is that you already have the necessary files on your own computer.

This section explains how you can get those files onto the local version of the Data Science Toolbox.

Local Version of Data Science Toolbox

Toolbox is an isolated virtual environment. Luckily, there is one exception to that: files can be transferred in and out the Data Science Toolbox. The local directory from which you ran vagrant up (which is the one that contains the file Vagrant file) is mapped to a directory in the Data Science Toolbox. This directory is called /vagrant (note that this is not your home directory).

Let's check the contents of this directory:

$ ls -1 /vagrant

Vagrantfile

If you have a file on your local computer, and you want to apply some command-line tools to it, all you have to do is copy or move the file to that directory.

Let's assume that you have a file called logs.csv on your Desktop.

And if you're running Windows, you can run the following on the Command Prompt or PowerShell:

> cd %UserProfile%\Desktop

> copy logs.csv MyDataScienceToolbox\

The file is now located in the directory /vagrant.

It's a good idea to keep your data in a separate directory (here we're using ~/book/ch03/data, for example).

So, after you have copied the file, you can move it by running:

$ mv /vagrant/logs.csv ~/book/ch03/data

Decompressing Files-

If the original data set is very large or it's a collection of many files, the file may be a (compressed) archive. Data sets which contain many repeated values are especially well suited for compression.

Common file extensions of compressed archives are:.tar.gz, .zip, and .rar.

To decompress these, you would use the command-line tools tar and unrar respectively. There are a few more, though less common, file extensions for which you would need yet other tools.

For example, in order to extract a file named logs.tar.gz, you would use:

$ cd ~/book/ch03

$ tar -xzvf data/logs.tar.gz

Indeed, tar is notorious for its many options. In this case, the four options x, z, v, and f specify that tar should extract files from an archive, use gzip as the decompression algorithm, be verbose, and use the file logs.tar.gz.

Converting Microsoft Excel Spreadsheets-

For many people, Microsoft Excel offers an intuitive way to work with small data sets and perform calculations on them. As a result, a lot of data is embedded into Microsoft Excel spreadsheets. These spreadsheets are, depending on the extension of the filename, stored in either a proprietary binary format (.xls) or as a collection of compressed XML files (.xlsx). In both cases, the data is not readily usable by most command-line tools.

There is a command-line tool called in2csv, which is able to convert Microsoft Excel spreadsheets to CSV files. CSV stands for comma-separated values or character-separated values.

In a CSV file, each record is located on a separate line, delimited by a line break (CRLF).

For example:

aaa,bbb,ccc CRLF

zzz,yyy,xxx CRLF

Let's demonstrate in2csv using a spreadsheet that contains the top 250 movies from the Internet Movie Database (IMDb). The file is named imdb-250.xlsx and can be obtained from http://bit.ly/analyzing_top250_movies_list.

To extract its data, we invoke in2csv as follows:

$ cd ~/book/ch03

$ in2csv data/imdb-250.xlsx > data/imdb-250.csv

The format of the file is automatically determined by the extension, *.xlsx* in this case. If we were to pipe the data into `in2csv`, we would have to specify the format explicitly. Let's look at the data:

```
$ in2csv data/imdb-250.xlsx | head | cut -c1-80
Title,title trim,Year,Rank,Rank (desc),Rating,New in 2011 from 2010?,2010 rank,R
Sherlock Jr. (1924),SherlockJr.(1924),1924,221,30,8,y,n/a,n/a,
The Passion of Joan of Arc (1928),ThePassionofJoanofArc(1928),1928,212,39,8,y,n/
His Girl Friday (1940),HisGirlFriday(1940),1940,250,1,8,y,n/a,n/a,
Tokyo Story (1953),TokyoStory(1953),1953,248,3,8,y,n/a,n/a,
The Man Who Shot Liberty Valance (1962),TheManWhoShotLibertyValance(1962),1962,2
Persona (1966),Persona(1966),1966,200,51,8,y,n/a,n/a,
Stalker (1979),Stalker(1979),1979,243,8,8,y,n/a,n/a,
```

```
Fanny and Alexander (1982),FannyandAlexander(1982),1982,210,41,8,y,n/a,n/a,
Beauty and the Beast (1991),BeautyandtheBeast(1991),1991,249,2,8,y,n/a,n/a,
```

As you can see, CSV by default is not too readable. You can pipe the data to a tool called `csvlook` (Groskopf, 2014), which will nicely format the data into a table. Here, we'll display a subset of the columns using `csvcut` such that the table fits on the page:

```
$ in2csv data/imdb-250.xlsx | head | csvcut -c Title,Year,Rating | csvlook
|---------------------------------------+------+---------|
|  Title                                | Year | Rating  |
|---------------------------------------+------+---------|
|  Sherlock Jr. (1924)                  | 1924 | 8       |
|  The Passion of Joan of Arc (1928)    | 1928 | 8       |
|  His Girl Friday (1940)               | 1940 | 8       |
|  Tokyo Story (1953)                   | 1953 | 8       |
|  The Man Who Shot Liberty Valance (1962) | 1962 | 8    |
|  Persona (1966)                       | 1966 | 8       |
|  Stalker (1979)                       | 1979 | 8       |
|  Fanny and Alexander (1982)           | 1982 | 8       |
|  Beauty and the Beast (1991)          | 1991 | 8       |
|---------------------------------------+------+---------|
```

A spreadsheet can contain multiple worksheets. By default, in2csv extracts the first worksheet. To extract a different worksheet, you need to pass the name of worksheet to the --sheet option. The tools in2csv, csvcut, and csvlook are actually part of Csvkit, which is a collection of command-line tools to work with CSV data. If you're running the Data Science Toolbox, you already have Csvkit installed.

Querying Relational Databases-

Most companies store their data in a relational database. Examples of relational data- bases are MySQL, PostgreSQL, and SQLite. These databases all have a slightly different way of interfacing with them. Some provide a command-line tool or a command line interface, while others do not. Moreover, they are not very consistent when it comes to their usage and output.

Fortunately, there is a command-line tool called sql2csv, which is part of the Csvkit suite.

The output of sql2csv is, as its name suggests, in CSV format. We can obtain data from relational databases by executing a SELECT query on them.

To select a specific set of data from an SQLite database named iris.db, sql2csv can be invoked as follows:

```
$ sql2csv --db 'sqlite:///data/iris.db' --query 'SELECT * FROM iris '\
> 'WHERE sepal_length > 7.5'
sepal_length,sepal_width,petal_length,petal_width,species
7.6,3.0,6.6,2.1,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
```

Here, we're selecting all rows where sepal_length is larger than 7.5.

Downloading from the Internet-

The Internet provides without a doubt the largest resource for data. This data is available in various forms, using various protocols.

The command-line tool cURL is used for downloading data from the Internet.

When you access a URL, which stands for uniform resource locator, through your browser, the data that is downloaded can be interpreted. For example, an HTML file is rendered as a website, an MP3 file may be automatically played, and a PDF file may be automatically opened by a viewer. However, when cURL is used to access a URL, the data is downloaded as is, and is printed to standard output. Other command-line tools may then be used to process this data further.

The easiest invocation of `curl` is to simply specify a URL as a command-line argument. For example, to download Mark Twain's *Adventures of Huckleberry Finn* from Project Gutenberg, we can run the following command:

```
$ curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt | head -n 10
```

By default, curl outputs a progress meter that shows the download rate and the expected time of completion. If you are piping the output directly to another command-line tool, such as head, be sure to specify the -s option, which stands for silent, so that the progress meter is disabled. Compare, for example, the output with the following command:

```
$ curl http://www.gutenberg.org/cache/epub/76/pg76.txt | head -n 10
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
```

Note that the output of the second command, where we do not disable the progress meter, contains the unwanted text and even an error message. If you save the data to a file, then you do not need to necessarily specify the `-s` option:

```
$ curl http://www.gutenberg.org/cache/epub/76/pg76.txt > data/finn.txt
```

You can also save the data by explicitly specifying the output file with the `-o` option:

```
$ curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt -o data/finn.txt
```

When downloading data from the Internet, the URL will most likely use the protocols HTTP or HTTPS. To download from an FTP server, which stands for *File Transfer Protocol*, you use `curl` in exactly the same way. When the URL is password protected, you can specify a username and a password as follows:

```
$ curl -u username:password ftp://host/file
```

If the specified URL is a directory, `curl` will list the contents of that directory.

Managing Your Data Workflow:

The command line is a very convenient environment for doing data science. As a con- sequence of working at the command line, we:

• Invoke many different commands

• Create custom and ad-hoc command-line tools

• Obtain and generate many (intermediate) files

As this process is of an exploratory nature, our workflow tends to be rather chaotic, which makes it difficult to keep track of what we've done. It's very important that our steps can be reproduced, whether by ourselves or by others.

When we, for example, continue with a project from a few weeks earlier, chances are that we have forgotten which commands we have run, on which files, in which order, and with which parameters.

Imagine the difficulty of passing on your analysis to a collaborator. You may recover some lost commands by digging into your Bash history, but this is, of course, not a good approach. A better approach would be to save your commands to a Bash script, such as run.sh. This allows you and your collaborators to at least reproduce the analysis.

A shell script is, however, a suboptimal approach because:

• It's difficult to read and to maintain.

• Dependencies between steps are unclear.

• Every step gets executed every time, which is inefficient and sometimes undesirable.

This is where Drake comes in handy. Drake is command-line tool created by Factual that allows you to:

• Formalize your data workflow steps in terms of input and output dependencies

• Run specific steps of your workflow from the command line

• Use inline code (e.g., Python and R)

• Store and retrieve data from external sources (e.g., S3 and HDFS)

Introducing Drake-

Drake organizes command execution around data and its dependencies. Your data processing steps are formalized in a separate text file (a workflow).

Each step usually has one or more inputs and outputs. Drake automatically resolves their dependencies and determines which commands need to be run and in which order.

This means that when you have, say, an SQL query that takes 10 minutes, it only has to be executed when the result is missing or when the query has changed afterwards. Also, if you want to (re-)run a specific step, Drake only considers (re-)running the steps on which it depends. This can save you a lot of time.

The benefit of having a formalized workflow allows you to easily pick up your project after a few weeks and to collaborate with others.

Example:

- **Obtain Top Ebooks from Project Gutenberg**

we'll use the following task as a running example. Our goal is to turn the command that we use to solve this task into a Drake workflow.

Project Gutenberg is a project that has archived and digitized over 42,000 books that are available online for free. On its website you can find the top 100 most downloaded eBooks. Let's assume that we're interested in the top 5 downloads of Project Gutenberg. Because this list is available in HTML, it's straightforward to obtain the top 5 downloads:

```
$ cd ~/book/ch06
$ curl -s 'http://www.gutenberg.org/browse/scores/top' |  ❶
> grep -E '^<li>' |                                       ❷
> head -n 5 |                                             ❸
> sed -E "s/.*ebooks\/([0-9]+).*/\\1/" > data/top-5       ❹
```

This command:

❶ Downloads the HTML.

❷ Extracts the list items.

❸ Keeps only the top 5 items.

❹ Saves ebook IDs to *data/top-5*.

The output of the command is:

```
$ cat data/top-5
1342
76
11
1661
1952
```

Now, we'll convert the preceding command to a Drake workflow. A work- flow is just a text file. You'd usually name this file Drakefile because Drake uses that file if no other file is specified at the command line. A workflow with just a single step would look like in below Example.

Example: A workflow with just a single step (Drakefile)

```
data/top-5 <-                                                    ❶
    curl -s 'http://www.gutenberg.org/browse/scores/top' |       ❷
    grep -E '^<li>' |                                            ❸
    head -n 5 |                                                  ❹
    sed -E "s/.*ebooks\/([0-9]+).*/\\1/" > data/top-5            ❺
```

Let's go through this file. The first line, which contains the arrow pointing to the left, is our step definition. The left side of this arrow, which says *top-5*, is the name or output of this step. Any inputs to this step would appear on the right side of this arrow, but because this step has no input, it's empty. Defining inputs and outputs is what allows Drake to recognize the dependencies between steps, and to figure out whether and when which steps need to be executed in order to fulfill a certain output. This output is also known as a *target*. As you can see, the body of this step is literally our command from before but then indented.

❶ The arrow (<-) denotes the name of the step and its dependencies. More on this later.

❷ The body is indented.

❸ Select only list items.

❹ Get the first 5 items.

❺ Extract the ID, and save to file *top-5*. Note that *top-5* was already specified in the step definition and that *5* has now been used three times. We're going to address that later.

now, let's run Drake and see what it does with our first workflow:

```
$ drake
The following steps will be run, in order:
  1: data/top-5 <-  [missing output]
Confirm? [y/n] y
Running 1 steps with concurrence of 1...

--- 0. Running (missing output): data/top-5 <-
--- 0: data/top-5 <-  -> done in 0.35s
Done (1 steps run).
```

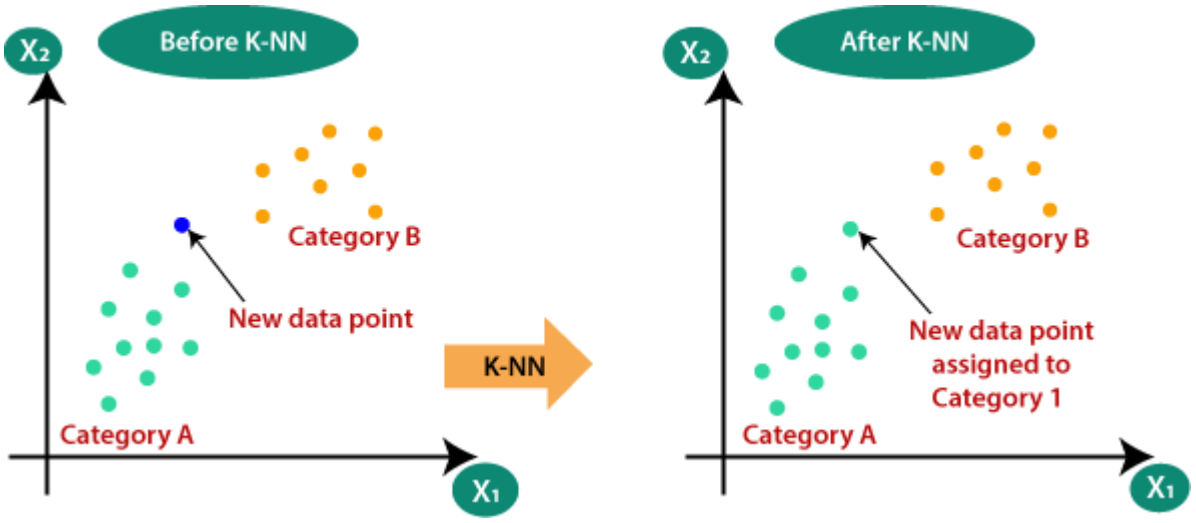
Techniques using Python Tools:

K-Nearest Neighbor (KNN) Algorithm:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

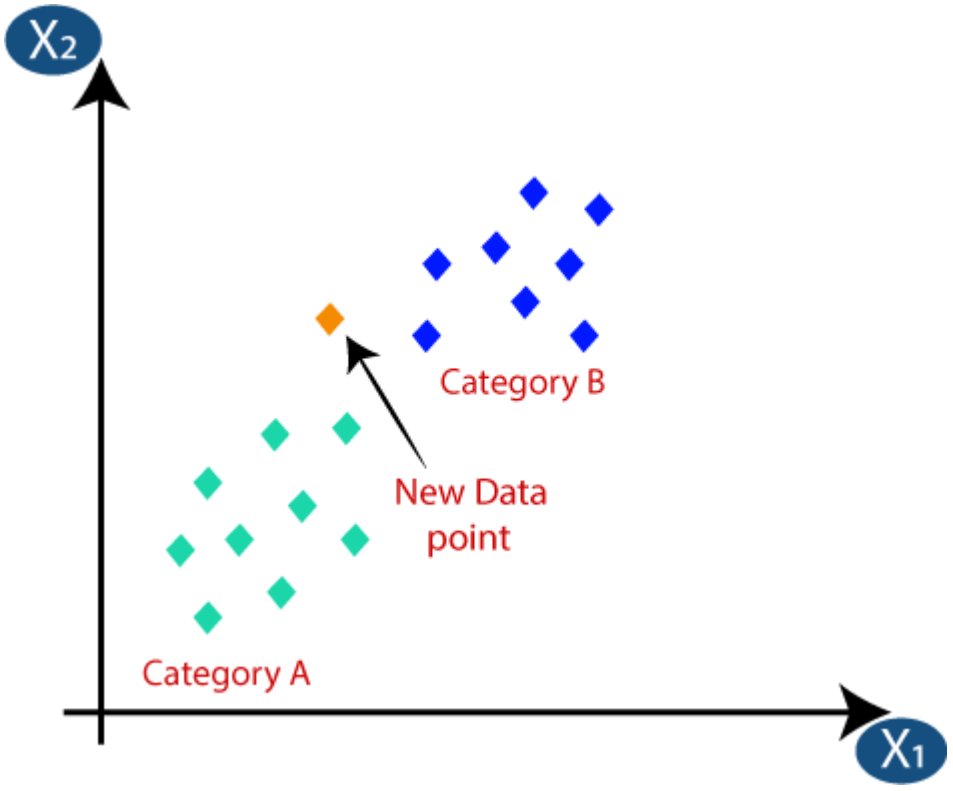


How does K-NN work?

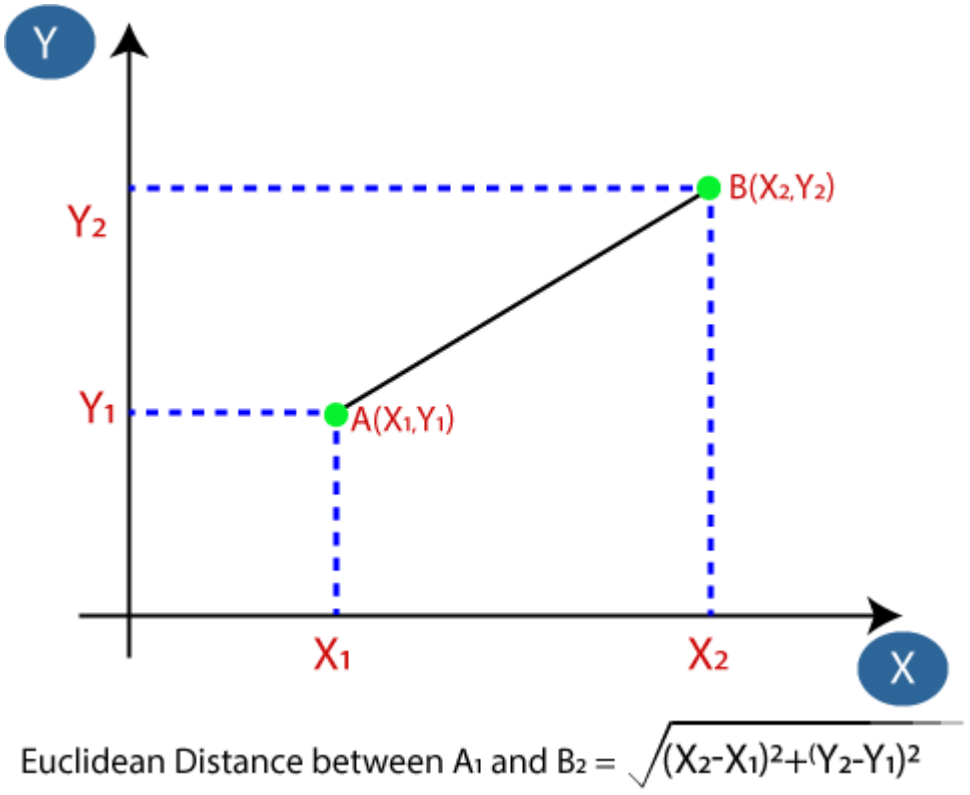
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

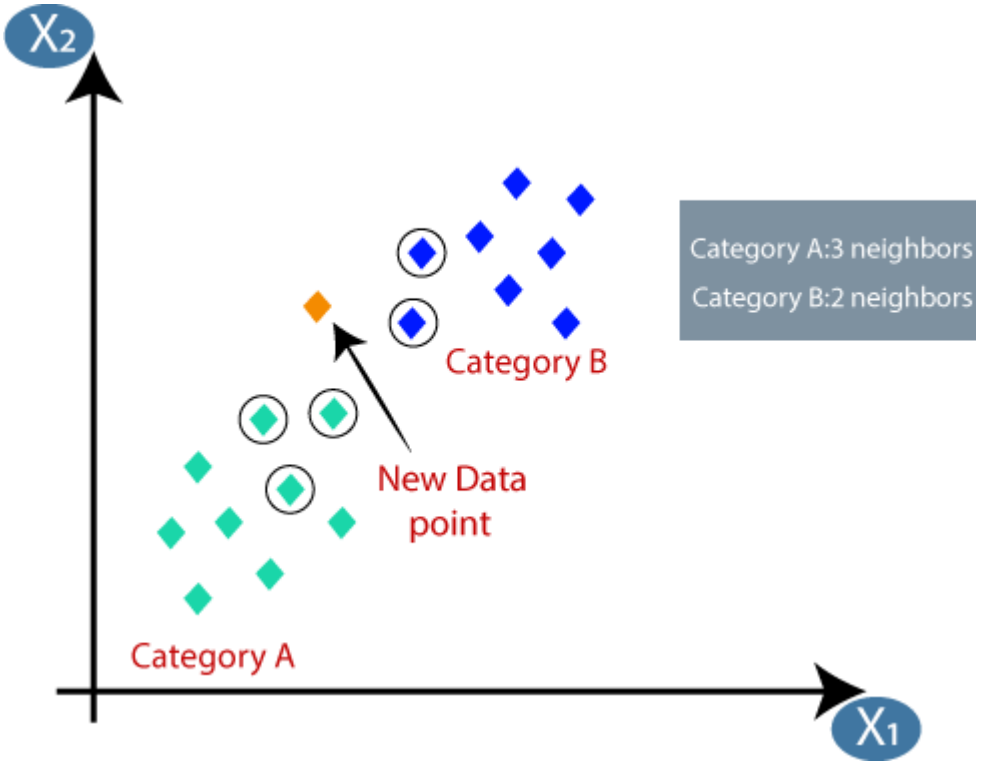
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Naïve Bayes Algorithm:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

UNIT-IV

Techniques using R Tools - R programming overview - Loading data into R – Modeling methods – choosing and evaluating models – Linear and logistic Regression

R programming overview:

"R is an interpreted computer programming language which was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand."

This programming language name is taken from the name of both the developers. The first project was considered in 1992. The initial version was released in 1995, and in 2000, a stable beta version was released.

It is also a software environment used to analyze **statistical information**, **graphical representation**, **reporting**, and **data modeling**.

R not only allows us to do branching and looping but also allows to do modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python, and FORTRAN languages to improve efficiency.

In the present era, R is one of the most important tool which is used by researchers, data analyst, statisticians, and marketers for retrieving, cleaning, analyzing, visualizing, and presenting data.

Features of R programming:

R is a domain-specific programming language which aims to do data analysis. It has some unique features which make it very powerful. The most important arguably being the notation of vectors. These vectors allow us to perform a complex operation on a set of values in a single command. There are the following features of R programming:

1. It is a simple and effective programming language which has been well developed.
2. It is data analysis software.
3. It is a well-designed, easy, and effective language which has the concepts of user-defined, looping, conditional, and various I/O facilities.
4. It has a consistent and incorporated set of tools which are used for data analysis.
5. For different types of calculation on arrays, lists and vectors, R contains a suite of operators.
6. It provides effective data handling and storage facility.
7. It is an open-source, powerful, and highly extensible software.
8. It provides highly extensible graphical techniques.
9. It allows us to perform multiple calculations using vectors.
10. R is an interpreted language.

Advantages of R:

1) Open Source

An open-source language is a language on which we can work without any need for a license or a fee. R is an open-source language. We can contribute to the development of R by optimizing our packages, developing new ones, and resolving issues.

2) Platform Independent

R is a platform-independent language or cross-platform programming language which means its code can run on all operating systems. R enables programmers to develop software for several competing platforms by writing a program only once. R can run quite easily on Windows, Linux, and Mac.

3) Machine Learning Operations

R allows us to do various machine learning operations such as classification and regression. For this purpose, R provides various packages and features for developing the artificial neural network. R is used by the best data scientists in the world.

4) Exemplary support for data wrangling

R allows us to perform data wrangling. R provides packages such as dplyr, readr which are capable of transforming messy data into a structured form.

5) Quality plotting and graphing

R simplifies quality plotting and graphing. R libraries such as ggplot2 and plotly advocates for visually appealing and aesthetic graphs which set R apart from other programming languages.

6) The array of packages

R has a rich set of packages. R has over 10,000 packages in the CRAN repository which are constantly growing. R provides packages for data science and machine learning operations.

7) Statistics

R is mainly known as the language of statistics. It is the main reason why R is predominant than other programming languages for the development of statistical tools.

8) Continuously Growing

R is a constantly evolving programming language. Constantly evolving means when something evolves, it changes or develops over time. R is a state of the art which provides updates whenever any new feature is added.

Disadvantages:

1) Data Handling

In R, objects are stored in physical memory. It is in contrast with other programming languages like Python. R utilizes more memory as compared to Python. It requires the entire data in one single place which is in the memory. It is not an ideal option when we deal with Big Data.

2) Basic Security

R lacks basic security. It is an essential part of most programming languages such as Python. Because of this, there are many restrictions with R as it cannot be embedded in a web-application.

3) Complicated Language

R is a very complicated language, and it has a steep learning curve. The people who don't have prior knowledge or programming experience may find it difficult to learn R.

4) Weak Origin

The main disadvantage of R is, it does not have support for dynamic or 3D graphics.

5) Lesser Speed

R programming language is much slower than other programming languages such as MATLAB and Python. In comparison to other programming language, R packages are much slower.

Loading data into R:

R is a programming language designed for data analysis. Therefore loading data is one of the core features of R.

R contains a set of functions that can be used to load data sets into memory. You can also load data into memory using R Studio - via the menu items and toolbars.

The most common ready-to-go data format is a family of tabular formats called structured values. Most of the data you find will be in (or nearly in) one of these formats. When you can read such files into R, you can analyze data from an incredible range of public and private data sources.

Data Formats

R can load data in two different formats:

- CSV files
- Text files

CSV means Comma Separated Values. You can export CSV files from many data carrying applications. For instance, you can export CSV files from data in an Excel spreadsheet. Here is an example of how a CSV file looks like inside:

```
name,id,salary
"John Doe",1,99999.00
"Joe Blocks",2,120000.00
"Cindy Loo",3,150000.00
```

As you can see, the values on each line are separated by commas. The first line contains a list of column names. These column names tell what the data in the following lines mean. These names only make sense to you. R does not care about these names. R just uses these names to identify data from the different columns.

A text file is typically similar to a CSV file, but instead of using commas as separators between values, text files often use other characters, like e.g. a Tab character. Here is an example of how a text file could look inside:

```
name          id      salary

"John Doe"      1       99999.00
"Joe Blocks"    2       120000.00
"Cindy Loo"     3       150000.00
```

As you can see, the data might be easier to read in text format - if you look at the data directly in the data file that is.

Actually, the name "text files" is a bit confusing. Both CSV files and text files contains data in textual form (as characters). One just uses commas as separator between the values, whereas the others use a tab character.

R contains a set of functions that can be used to load data sets into memory. We can also load data into memory using R Studio - via the menu items and toolbars. the following section describes both the ways:

R has **three** different functions which can import data. These are:

read.table()

read.csv()

read.delim()

These functions are very similar to each other, so if you master one of them you will soon master the others. In fact, you can probably just use the read.table() function for all of your data imports. These 3 functions will be covered in the following sections.

1. read.table()

The R function read.table() function loads data from a file into a tabular data set (table) in memory. A tabular data set consists of rows and columns, just like a spreadsheet. Sometimes rows are also referred to as "records" and columns referred to as "fields" or "properties".

The read.table() function takes three parameters:

- The file name of the file to load
- A flag telling if the file contains a header line
- The separator character used inside the file to separate the values of each row.

The parameters to read.table() are listed between the parentheses, separated with commas. Here is an example of loading a CSV file using read.table() in R:

read.table("data.csv", header=T, sep=";")

The first parameter is the path to the file to read. In the example above that is the "data.csv" part. This parameter should contain a path to the file to read. In the above example only the file name itself is shown. Then R expects to find the file in the same directory R is running from. If you want to specify the full path to the file, you can do so too. Here is an example of how that looks on Windows:

"d:\\data\\projects\\tutorial-projects\\r-programming\\data.csv"

Normally, Windows only uses a single backslash (the \ character) between directory names, but in programming languages it is normal to use the \ character as an escape character in strings (text variables). When a programming language sees a \ in a string it will normally look at the next character after the \ to determine what character to insert into the string. To actually insert a \ you will therefore often need two \ (\\) as shown above.

The same file path on a Mac or Linux machine could look like this:

"/data/projects/tutorial-projects/r-programming/data.csv"

Notice the use of / between directories instead of \, and notice that you only need a single / between the directories, because / is not an escape character.

The second parameter of read.table() is the header=T part. This tells the read.table() function whether the first line in the data file is a header line or not. A value of header=T or header=TRUE means that the first line is a header line. A value of header=F or header=FALSE means that the first line is not a header line.

By "header line" is meant whether the first line contains the column names, or if the first line already contains data. Look at this CSV file:

name;id;salary

John Doe;1;99999

Joe Blocks;2;120000

Cindy Loo;3;150000

Notice how the first row contains the column names for the data on the following rows.

The third parameter specifies what character inside the data file that is used to separate the different column values on each row. If you look at the CSV file contents above you can see that a semicolon (;) is used as separator. That is why the third parameter to the read.table() function call is sep=";" meaning that the separator character used in the data file is a semicolon.

To execute read.table() you type the commands shown in this section into the console part of R Studio and press the "Enter" key.

More read.table() Parameters

The read.table() function is very advanced and can take more parameters than I have shown above. To get a full list of the parameters, type

help("read.table")

into the R console in R Studio and press enter. In the lower right part of the R Studio window, R Studio will show you the help for the read.table() function.

Assigning the Data Set to a Variable

If you just type in this command:

read.table("data.csv", header=T, sep=";")

Then R Studio will load the data file and print its contents to the console. But the data set will not be kept in memory. Too keep the data set in memory so you can work with it, you have to assign it to a variable. You do so like this:

data = read.table("data.csv", header=T, sep=";")

Or like this:

data <- read.table("data.csv", header=T, sep=";")

The first word, data, is the name of the variable you want to assign the loaded data set to. You can freely choose the variable name (but not all characters are allowed). You can load multiple data sets into memory, and assign each data set to its own variable. Then you can access them separately during your analysis.

Whether you use the = or <- after the variable name doesn't matter. I prefer the = character because I am used to that from other programming languages. But some prefer the <- notation. The result is the same though.

2. read.csv()

The read.csv() function reads a CSV file into the memory. The read.csv() function takes 3 parameters, just like the read.table() function. Here is an example call to the read.csv() function:

```
data = read.csv("D:\\data\\data.csv", header=T, sep=";")
```

This example loads the CSV file located at D:\\data\\data.csv and assign it to the variable named data. The first line is a header line containing the names of the columns in the CSV file. This is specified by the second parameter header=T. The third parameter specifies that the separator character used inside the CSV file is ; (a semicolon).

3. read.delim()

The read.delim() function reads a CSV file into the memory, just like the read.csv() function. The read.delim() function takes 3 parameters, just like the read.table() function. Here is an example call to the read.delim() function:

```
data = read.delim("D:\\data\\data.csv", header=T, sep=";")
```

This example loads the CSV file located at D:\\data\\data.csv and assign it to the variable named data. The first line is a header line containing the names of the columns in the CSV file. This is specified by the second parameter header=T. The third parameter specifies that the separator character used inside the CSV file is ; (a semicolon).

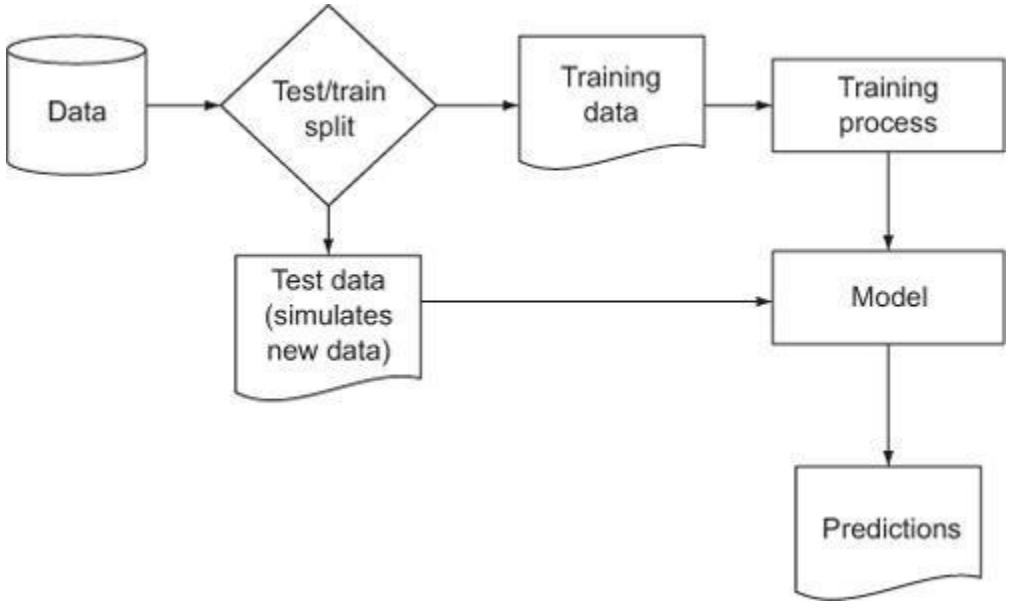
Modeling Methods:

1. choosing and evaluating models:

As a data scientist, your ultimate goal is to solve a concrete business problem: increase look-to-buy ratio, identify fraudulent transactions, predict and manage the losses of a loan portfolio, and so on.

Many different statistical modeling methods can be used to solve any given problem. Each statistical method will have its advantages and disadvantages for a given business goal and business constraints.

To make progress, you must be able to measure model quality during training and also ensure that your model will work as well in the production environment as it did on your training data. In general, we'll call these two tasks **model evaluation** and **model validation**. To prepare for these statistical tests, we always split our data into training data and test data, as illustrated in figure below:



We define model evaluation as quantifying the performance of a model. To do this we must find a measure of model performance that's appropriate to both the original business goal and the chosen modeling technique. For example, if we're predicting who would default on loans, we have a classification task, and measures like precision and recall are appropriate. If we instead are predicting revenue lost to defaulting loans, we have a scoring task, and measures like root mean square error (RMSE) are appropriate. The point is this: there are a number of measures the data scientist should be familiar with.

We define model validation as the generation of an assurance that the model will work in production as well as it worked during training. It is a disaster to build a model that works great on the original training data and then performs poorly when used in production. The biggest cause of model validation failures is not having enough training data to represent the variety of what may later be encountered in production. For example, training a loan default model only on people who repaid their loans might score well on a simple evaluation ("predicts no defaults and is 100% accurate!") but would obviously not be a good model to put into

production. Validation techniques attempt to quantify this type of risk before you put a model into production.

Mapping problems to machine learning tasks:

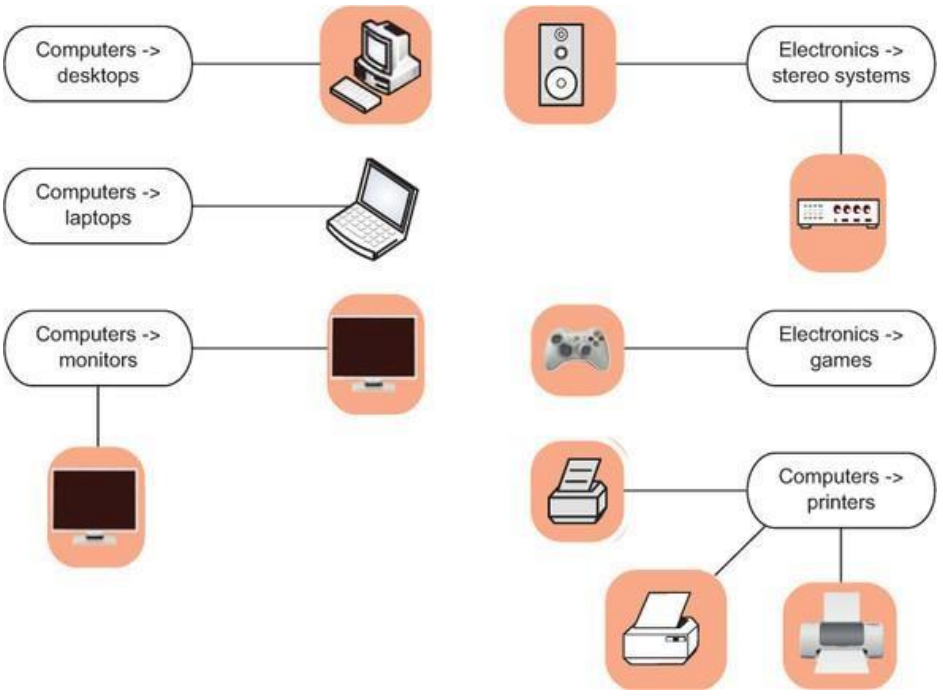
Your task is to map a business problem to a good machine learning method. To use a real-world situation, let's suppose that you're a data scientist at an online retail company. There are a number of business problems that your team might be called on to address:

- Predicting what customers might buy, based on past transactions
- Identifying fraudulent transactions
- Determining price elasticity (the rate at which a price increase will decrease sales, and vice versa) of various products or product classes
- Determining the best way to present product listings when a customer searches for an item
- Customer segmentation: grouping customers with similar purchasing behavior
- AdWord valuation: how much the company should spend to buy certain AdWords on search engines
- Evaluating marketing campaigns
- Organizing new products into a product catalog

Your intended uses of the model have a big influence on what methods you should use. If you want to know how small variations in input variables affect outcome, then you likely want to use a regression method. If you want to know what single variable drives most of a categorization, then decision trees might be a good choice. Also, each business problem suggests a statistical approach to try. If you're trying to predict scores, some sort of regression is likely a good choice; if you're trying to predict categories, then something like random forests is probably a good choice.

Solving classification problems:

Suppose your task is to automate the assignment of new products to your company's product categories, as shown in figure 5.2. This can be more complicated than it sounds. Products that come from different sources may have their own product classification that doesn't coincide with the one that you use on your retail site, or they may come without any classification at all. Many large online retailers use teams of human taggers to hand-categorize their products. This is not only labor-intensive, but inconsistent and error-prone. Automation is an attractive option; it's labor-saving, and can improve the quality of the retail site.



Product categorization based on product attributes and/or text descriptions of the product is an example of classification: deciding how to assign (known) labels to an object. Classification itself is an example of what is called supervised learning: in order to learn how to classify objects, you need a dataset of objects that have already been classified (called the training set). Building training data is the major expense for most classification tasks, especially text-related ones. Table 5.1 lists some of the more common effective classification methods.

Table 5.1 Some common classification methods

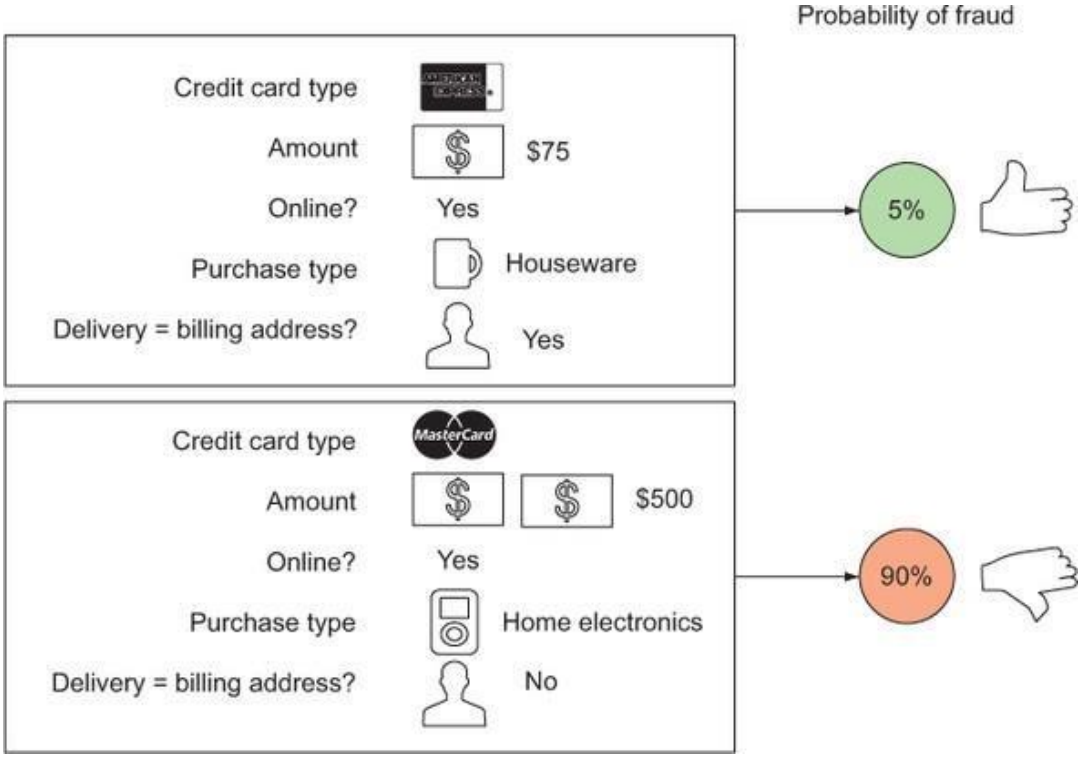
Method	Description
Naive Bayes	Naive Bayes classifiers are especially useful for problems with many input variables, categorical input variables with a very large number of possible values, and text classification. Naive Bayes would be a good first attempt at solving the product categorization problem.
Decision trees	Decision trees (discussed in section 6.3.2) are useful when input variables interact with the output in "if-then" kinds of ways (such as IF `age > 65`, THEN `has.health.insurance=T`). They are also suitable when inputs have an AND relationship to each other (such as IF `age < 25` AND `student=T`, THEN...) or when input variables are redundant or correlated. The decision rules that come from a decision tree are in principle easier for nontechnical users to understand than the decision processes that come from other classifiers. In section 6.3.2, we'll discuss an important extension of decision trees: random forests.
Logistic regression	Logistic regression is appropriate when you want to estimate class probabilities (the probability that an object is in a given class) in addition to class assignments.[a] An example use of a logistic regression–based classifier is estimating the probability of fraud in credit card purchases. Logistic regression is also a good choice when you want an idea of the relative impact of different input variables on the output. For example, you might find out that a $100 increase in transaction size increases the odds that the transaction is fraud by 2%, all else being equal.

Table 5.1 Some common classification methods *(continued)*

Method	Description
Support vector machines	Support vector machines (SVMs) are useful when there are very many input variables or when input variables interact with the outcome or with each other in complicated (nonlinear) ways. SVMs make fewer assumptions about variable distribution than do many other methods, which makes them especially useful when the training data isn't completely representative of the way the data is distributed in production.

Solving scoring problems:

For a scoring example, suppose that your task is to help evaluate how different marketing campaigns can increase valuable traffic to the website. The goal is not only to bring more people to the site, but to bring more people who buy. You're looking at a number of different factors: the communication channel (ads on websites, YouTube videos, print media, email, and so on); the traffic source (Facebook, Google, radio stations, and so on); the demographic targeted; the time of year; and so on. Predicting the increase in sales from a particular marketing campaign is an example of regression, or scoring. Fraud detection can be considered scoring, too, if you're trying to estimate the probability that a given transaction is a fraudulent one (rather than just returning a yes/no answer). This is shown in figure 5.3. Scoring is also an instance of supervised learning.



COMMON SCORING METHODS-

Linear regression- Linear regression builds a model such that the predicted numerical output is a linear additive function of the inputs. This can be a very effective approximation, even

when the underlying situation is in fact nonlinear. The resulting model also gives an indicationof the relative impact of each input variable on the output. Linear regression is often a good first model to try when trying to predict a numeric value.

Logistic regression- Logistic regression always predicts a value between 0 and 1, making it suitable for predicting probabilities (when the observed outcome is a categorical value) and rates (when the observed outcome is a rate or ratio). As we mentioned, logistic regression is an appropriate approach to the fraud detection problem, if what you want to estimate is the probability that a given transaction is fraudulent or legitimate.

Working without known targets:

The preceding methods require that you have a training dataset of situations with known outcomes. In some situations, there's not (yet) a specific outcome that you want to predict. Instead, you may be looking for patterns and relationships in the data that will help you understand your customers or your business better. These situations correspond to a class of approaches called unsupervised learning: rather than predicting outputs based on inputs, the objective of unsupervised learning is to discover similarities and relationships in the data. Some common clustering methods include these:

- K-means clustering
- Apriori algorithm for finding association rules
- Nearest neighbor

WHEN TO USE BASIC CLUSTERING-

Suppose you want to segment your customers into general categories of people with similar buying patterns. You might not know in advance what these groups should be. This problem is a good candidate for k-means clustering. K-means clustering is one way to sort the data into groups such that members of a cluster are more similar to each other than they are to members of other clusters. Suppose that you find that your customers cluster into those with young children, who make more family-oriented purchases, and those with no children or with adult children, who make more leisure- and social-activity-related purchases. Once you have assigned a customer into one of those clusters, you can make general statements about their behavior. For example, a customer in the with-young children cluster is likely to respond more favorably to a promotion on attractive but durable glassware than to a promotion on fine crystal wine glasses.

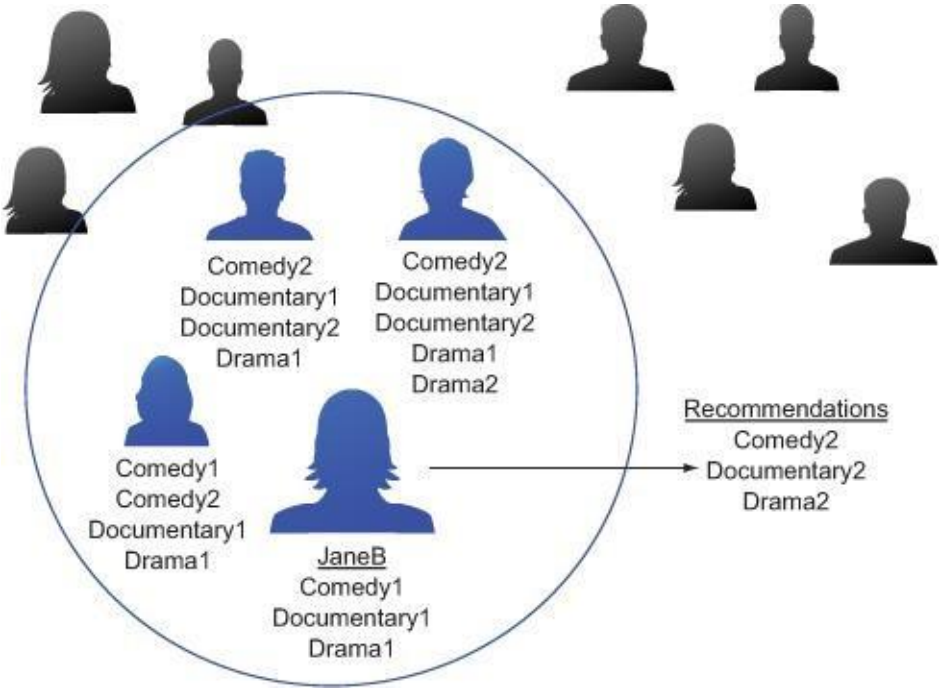
WHEN TO USE ASSOCIATION RULES

You might be interested in directly determining which products tend to be purchased together. For example, you might find that bathing suits and sunglasses are frequently purchased at the same time.

This is a good application for association rules (or even recommendation systems). You can mine useful product recommendations: whenever you observe that someone has put a bathing suit into their shopping cart, you can recommend suntan lotion, as well. We can use the Apriori algorithm for discovering association rules.

WHEN TO USE NEAREST NEIGHBOR METHODS

Another way to make product recommendations is to find similarities in people (figure 5.6). For example, to make a movie recommendation to customer JaneB, you might look for the three customers whose movie rental histories are the most like hers. Any movies that those three people rented, but JaneB has not, are potentially useful recommendations for her. This can be solved with nearest neighbor (or k-nearest neighbor methods, with K = 3). Nearest neighbor algorithms predict something about a data point p (like a customer's future purchases) based on the data point or points that are most similar to p.



Problem-to-method mapping

Table 5.2 maps some typical business problems to their corresponding machine learning task, and to some typical algorithms to tackle each task. Table 5.2 From problem to approach Example tasks Machine learning terminology .

Table 5.2 From problem to approach

Example tasks	Machine learning terminology	Typical algorithms
Identifying spam email Sorting products in a product catalog Identifying loans that are about to default Assigning customers to customer clusters	Classification: assigning known labels to objects	Decision trees Naive Bayes Logistic regression (with a threshold) Support vector machines

Table 5.2 From problem to approach *(continued)*

Example tasks	Machine learning terminology	Typical algorithms
Predicting the value of AdWords Estimating the probability that a loan will default Predicting how much a marketing campaign will increase traffic or sales	Regression: predicting or forecasting numerical values	Linear regression Logistic regression
Finding products that are purchased together Identifying web pages that are often visited in the same session Identifying successful (much-clicked) combinations of web pages and AdWords	Association rules: finding objects that tend to appear in the data together	Apriori
Identifying groups of customers with the same buying patterns Identifying groups of products that are popular in the same regions or with the same customer clusters Identifying news items that are all discussing similar events	Clustering: finding groups of objects that are more similar to each other than to objects in other groups	K-means
Making product recommendations for a customer based on the purchases of other similar customers Predicting the final price of an auction item based on the final prices of similar products that have been auctioned in the past	Nearest neighbor: predicting a property of a datum based on the datum or data that are most similar to it	Nearest neighbor

Evaluating models:

When building a model, the first thing to check is if the model even works on the data it was trained from.

From an evaluation point of view, we group model types this way:

- Classification
- Scoring
- Probability estimation
- Ranking
- Clustering

For most model evaluations, we just want to compute one or two summary scores that tell us if the model is effective.

Evaluating classification models:

A classification model places examples into two or more categories. The most common measure of classifier quality is **accuracy**. For measuring classifier performance, we'll first introduce the incredibly useful tool called the **confusion matrix** and show how it can be used to calculate many important evaluation scores. The first score we'll discuss is accuracy, and then we'll move on to better and more detailed measures such as **precision** and **recall.**

Let's use the example of classifying email into spam (email we in no way want) and non-spam (email we want). A ready-to-go example (with a good description) is the Spambase dataset . Each row of this dataset is a

set of features measured for a specific email and an additional column telling whether the mail was spam (unwanted) or non-spam (wanted).

THE CONFUSION MATRIX -The absolute most interesting summary of classifier performance is the confusion matrix. This matrix is just a table that summarizes the classifier's predictions against the actual known data categories. The confusion matrix is a table counting how often each combination of known outcomes (the truth) occurred in combination with each prediction type. For our email spam example, the confusion matrix is given by the following R command.

Listing 5.3 Spam confusion matrix

```
> cM <- table(truth=spamTest$spam,prediction=spamTest$pred>0.5)
> print(cM)
          prediction
truth       FALSE TRUE
  non-spam    264   14
  spam         22  158
```

Using this summary, we can now start to judge the performance of the model. In a two-by-two confusion matrix, every cell has a special name, as illustrated in table 5.4.

Table 5.4 Standard two-by-two confusion matrix

	Prediction=NEGATIVE	Prediction=POSITIVE
Truth mark=NOT IN CATEGORY	True negatives (TN) `cM[1,1]=264`	False positives (FP) `cM[1,2]=14`
Truth mark=IN CATEGORY	False negatives (FN) `cM[2,1]=22`	True positives (TP) `cM[2,2]=158`

ACCURACY- Accuracy is by far the most widely known measure of classifier performance. For a classifier, accuracy is defined as the number of items categorized correctly divided by the total number of items. It's simply what fraction of the time the classifier is correct. At the very least, you want a classifier to be accurate. In terms of our confusion matrix, accuracy is (TP+TN)/(TP+FP+TN+FN)=(cM[1,1]+cM[2,2])/sum(cM) or 92% accurate. The error of around 8% is unacceptably high for a spam filter, but good for illustrating different sorts of model evaluation criteria.

PRECISION AND RECALL- Another evaluation measure used by machine learning researchers is a pair of numbers called precision and recall. Precision is what fraction of the items the classifier flags as being in the class actually are in the class. So precision is TP/(TP+FP), which is cM[2,2]/(cM[2,2]+cM[1,2]).

Recall is what fraction of the things that are in the class are detected by the classifier, or TP/(TP+FN)=cM[2,2]/(cM[2,2]+cM[2,1]). For our email spam example this is 88%.

Evaluating scoring models

Evaluating models that assign scores can be a somewhat visual task. The main concept is looking at what is called the residuals or the difference between our predictions f(x[i,]) and actual outcomes y[i].

ROOT MEAN SQUARE ERROR- The most common goodness-of-fit measure is called root mean square error (RMSE). This is the square root of the average square of the difference between our prediction and actual

values.

R-SQUARED-Another important measure of fit is called R-squared (or R^2, or the coefficient of determination). It's defined as 1.0 minus how much unexplained variance your model leaves (measured relative to a null model of just using the average y as a prediction).

Evaluating probability models:

Probability models are useful for both classification and scoring tasks. Probability models are models that both decide if an item is in a given class and return an estimated probability (or confidence) of the item being in the class. The modeling techniques of logistic regression and decision trees are fairly famous for being able to return good probability estimates.

Evaluating ranking models :

Ranking models are models that, given a set of examples, sort the rows or (equivalently) assign ranks to the rows. Ranking models are often trained by converting groups of examples into many pair-wise decisions (statements like "a is before b"). You can then apply the criteria for evaluating classifiers to quantify the quality of your ranking function. Two other standard measures of a ranking model are Spearman's rank correlation coefficient (treating assigned rank as a numeric score) and the data mining concept of lift .

Evaluating clustering models :

Clustering models are hard to evaluate because they're unsupervised: the clusters that items are assigned to are generated by the modeling procedure, not supplied in a series of annotated examples. Evaluation is largely checking observable summaries about the clustering. Distance metrics are good for checking the performance of clustering algorithms, but they don't always translate to business needs.

Validating models:

We've discussed how to choose a modeling technique and evaluate the performance of the model on training data. We call the testing of a model on new data (or a simulation of new data from our test set)as model validation. The following sections discuss the main problems we try to identify.

Identifying common model problems:

Table 5.7 lists some common modeling problems you may encounter.

Table 5.7 Common model problems

Problem	Description
Bias	Systematic error in the model, such as always underpredicting.
Variance	Undesirable (but non-systematic) distance between predictions and actual values. Often due to oversensitivity of the model training procedure to small variations in the distribution of training data.
Overfit	Features of the model that arise from relations that are in the training data, but not representative of the general population. Overfit can usually be reduced by acquiring more training data and by techniques like regularization and bagging.
Nonsignificance	A model that appears to show an important relation when in fact the relation may not hold in the general population, or equally good predictions can be made without the relation.

OVERFITTING-

A lot of modeling problems are related to overfitting. Looking for signs of overfit is a good first step in diagnosing models.

An overfit model looks great on the training data and performs poorly on new data. A model's prediction error on the data that it trained from is called training error. A model's prediction error on new data is called generalization error. Usually, training error will be smaller than generalization error .

Ideally, though, the two error rates should be close. If generalization error is large, then your model has probably overfit.You want to avoid overfitting by preferring (as long as possible) simpler models, which do in fact tend to generalize better. Figure 5.11 shows the typical appearance of a reasonable model and an overfit model. overfit models tend to be less accurate in production than during training, which is embarrassing.

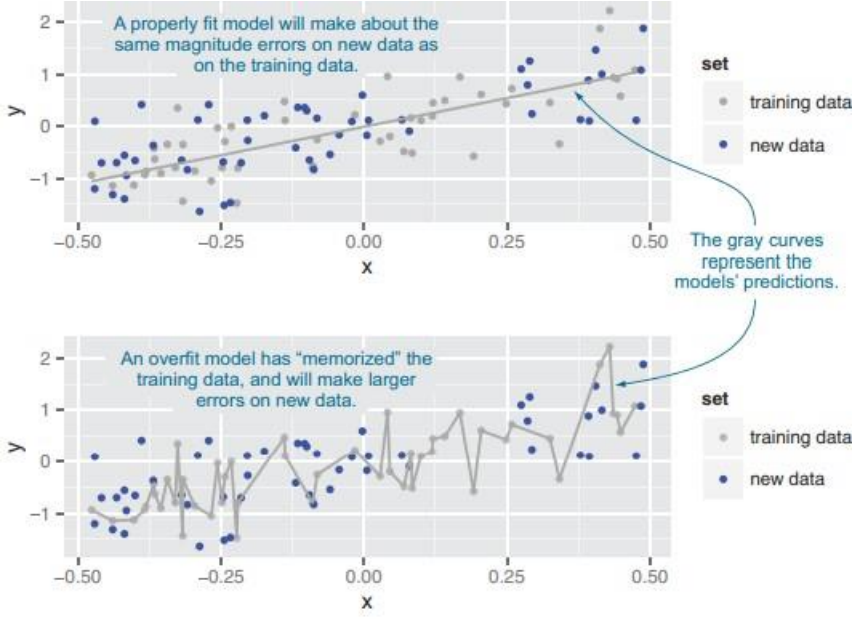


Figure 5.11 A notional illustration of overfitting

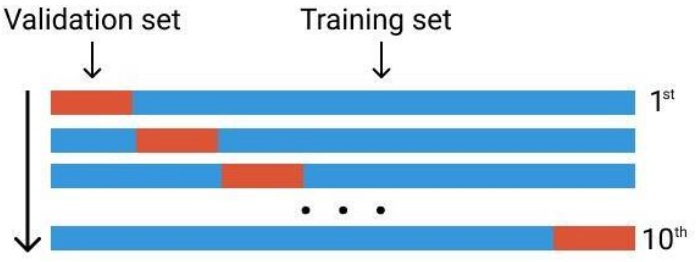
Ensuring model quality:

K-FOLD CROSS-VALIDATION-

The idea behind k-fold cross-validation is to repeat the construction of the model on different subsets of the available training data and then evaluate the model only on data not seen during construction. This is an attempt to simulate the performance of the model on unseen future data.

K-fold cross-validation is one of the most commonly used model evaluation methods.

Let's say that we have 100 rows of data. We randomly divide them into ten groups of folds. Each fold will consist of around 10 rows of data. The first fold is going to be used as the validation set, and the rest is for the training set. Then we train our model using this dataset and calculate the accuracy or loss. We then repeat this process but using a different fold for the validation set. See the image below.



SIGNIFICANCE TESTING-

The idea behind significance testing is that we can believe our model's performance is good if it's very unlikely that a naive model (a null hypothesis) could score as well as our model. The standard incantation in that case is "We can reject the null hypothesis." This means our model's measured performance is unlikely for the null model. Null models are always of a simple form: assuming two effects are independent when we're trying to model a relation, or assuming a variable has no effect when we're trying to measure an effect strength.

For example, suppose you've trained a model to predict how much a house will sell for, based on certain variables. You want to know if your model's predictions are better than simply guessing the average selling price of a house in the neighborhood (call this the null model). Your new model will mispredict a given house's selling price by a certain average amount, which we'll call err.model. The null model will also mispredict a given house's selling price by a different amount, err.null. The null hypothesis is that D = (err.null - err.model) == 0—on average, the new model performs the same as the null model. When you evaluate your model over a test set of houses, you will (hopefully) see that D = (err.null - err.model) > 0 (your model is more accurate). You want to make sure that this positive outcome is genuine, and not just something you observed by chance. The p-value is the probability that you'd see a D as large as you observed if the two models actually perform the same.

CONFIDENCE INTERVALS –

An important and very technical frequentist statistical concept is the confidence interval.

A confidence interval is the mean of your estimate plus and minus the variation in that estimate. This

is the range of values you expect your estimate to fall between if you redo your test, within a certain level of confidence.

Confidence, in statistics, is another way to describe probability. For example, if you construct a confidence interval with a 95% confidence level, you are confident that 95 out of 100 times the estimate will fall between the upper and lower values specified by the confidence interval.

Linear and Logistic regression:

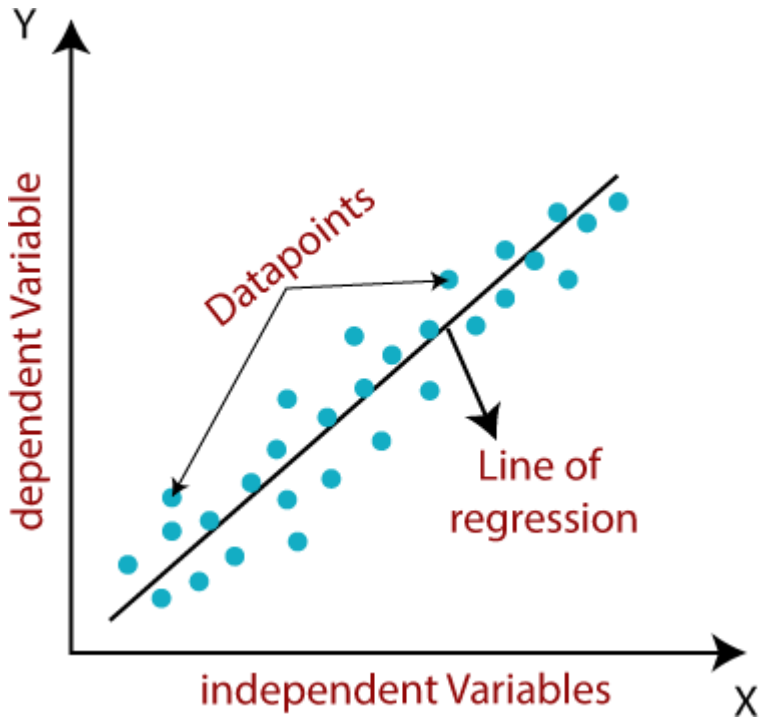
Linear regression:

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$y= a_0+a_1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

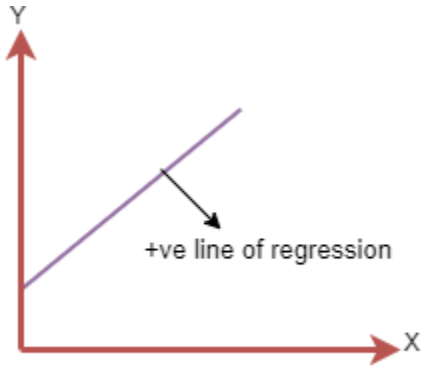
- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

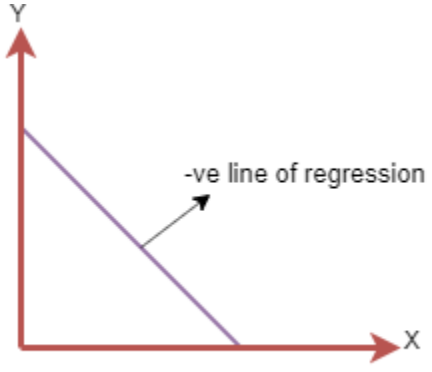
A linear line showing the relationship between the dependent and independent variables is called a regression line. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function:

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.

- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

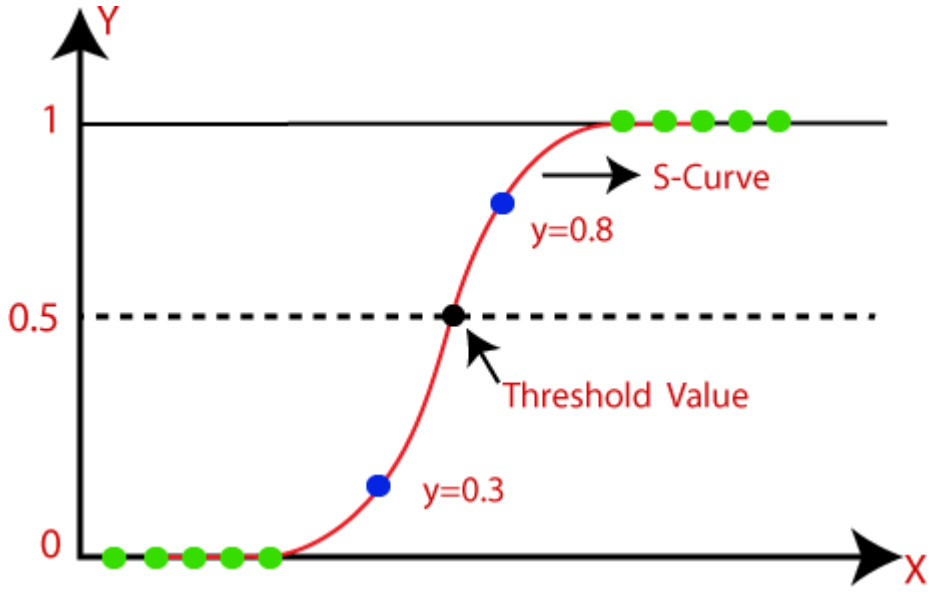
1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Logistic Regression:

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

UNIT-V

Data manipulation using Pandas- Installing and using Pandas-Introducing Pandas Objects-Data Indexing and selection- -Handling Missing data, Merge and joining Data sets, Aggregation and Grouping

Data visualization using Matplotlib-Simple Line Plots-Simple scatter plots, Multiple subplots, Visualization with seaborn

Installing and using Pandas:

How to Install Python Pandas on Windows?

Pandas in Python is a package that is written for data analysis and manipulation. Pandas offer various operations and data structures to perform numerical data manipulations and time series. Pandas is an open-source library that is built over Numpy libraries. Pandas library is known for its high productivity and high performance. Pandas is popular because it makes importing and analyzing data much easier.

Pandas programs can be written on any plain text editor like **notepad, notepad++,** or anything of that sort and saved with a **.py** extension. To begin with, writing Pandas Codes and performing various intriguing and useful operations, one must have Python installed on their System. This can be done by following the step by step instructions provided below:

Downloading and Installing Pandas

Pandas can be installed in multiple ways on Windows and on Linux. Various different ways are listed below:

Windows

Python Pandas can be installed on Windows in two ways:

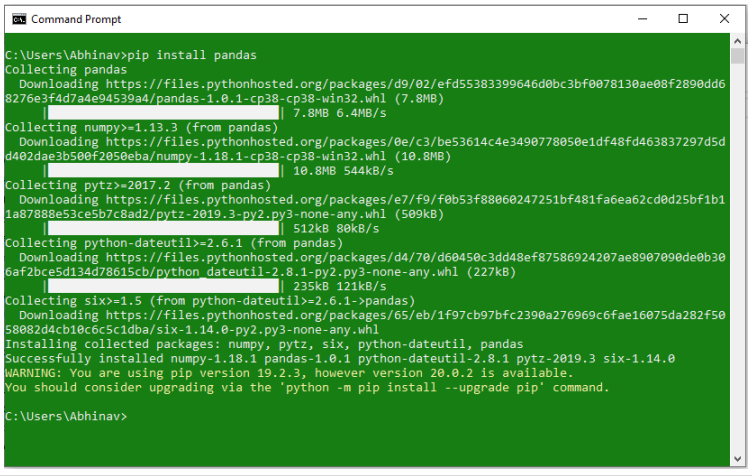
- Using pip
- Using Anaconda

Install Pandas using pip

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI).

Pandas can be installed using PIP by the use of the following command:

pip install pandas

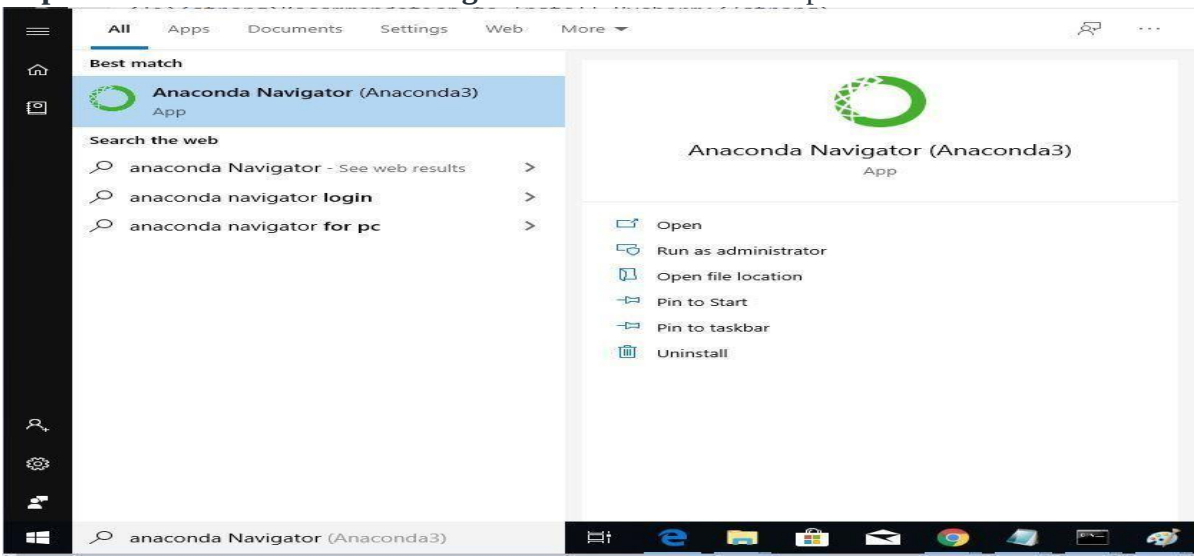


Install Pandas using Anaconda

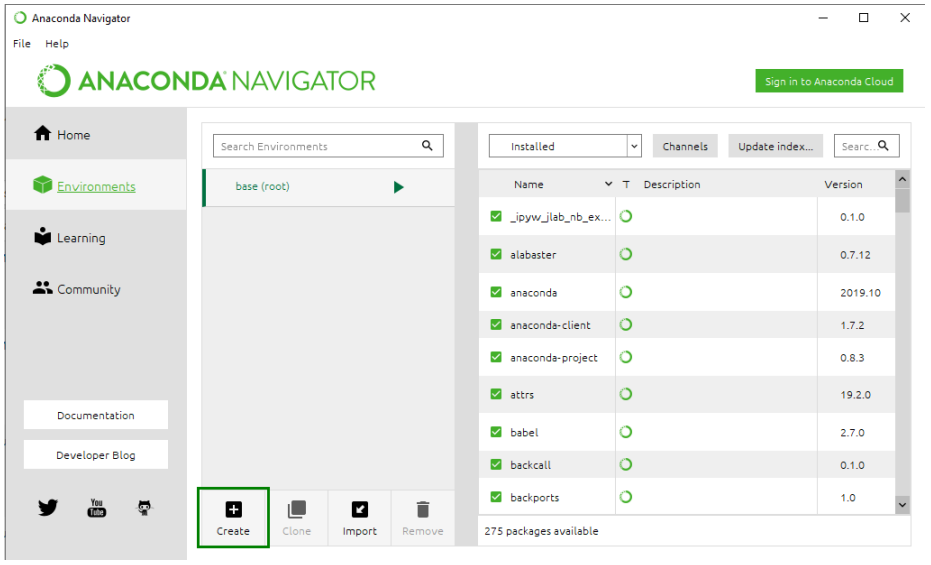
Anaconda is open-source software that contains Jupyter, spyder, etc that are used for large data processing, data analytics, heavy scientific computing. If your system is not pre-equipped with Anaconda Navigator, you can learn **how to install Anaconda Navigator on** Windows **or** Linux**?**

Steps to Install Pandas using Anaconda Navigator:

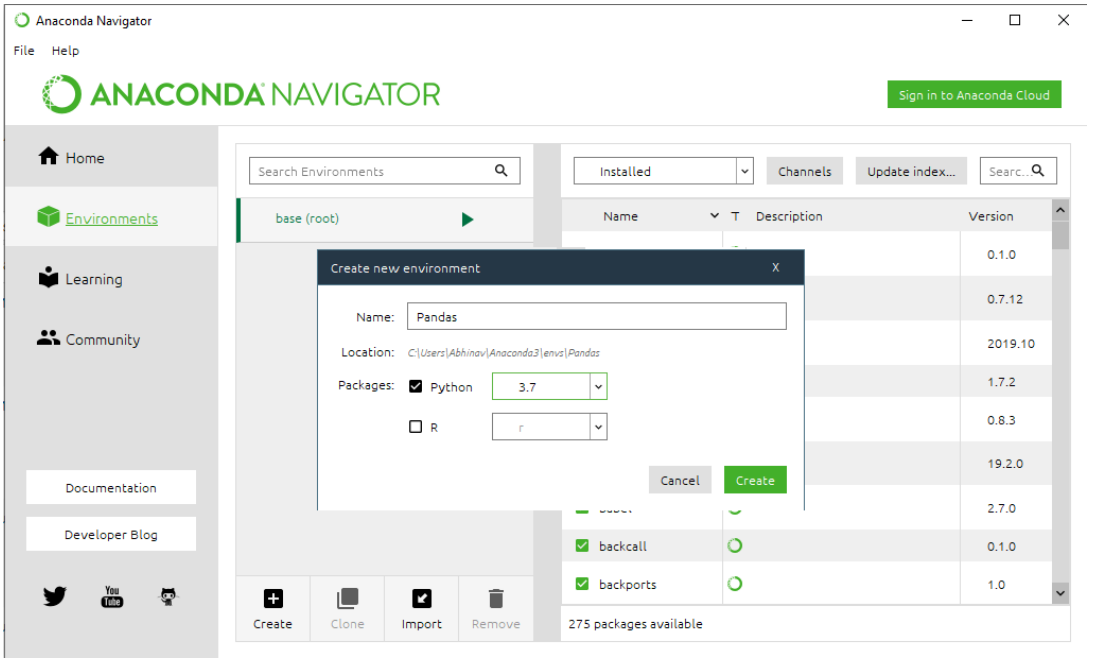
Step 1: Search for **Anaconda Navigator** in Start Menu and open it.



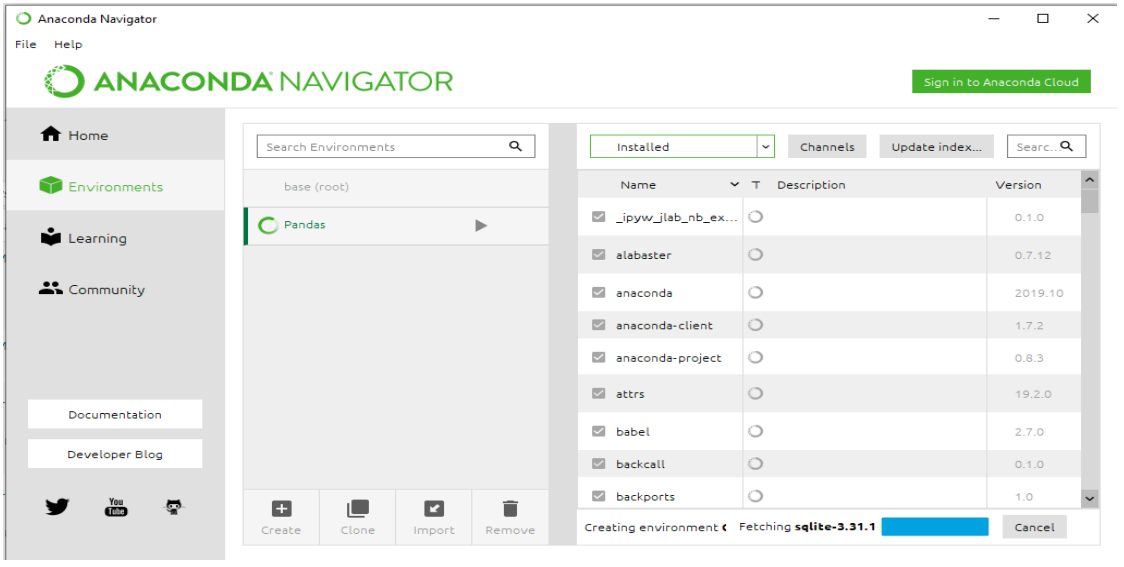
Step 2: Click on the **Environment tab** and then click on the **create** button to create a new Pandas Environment.



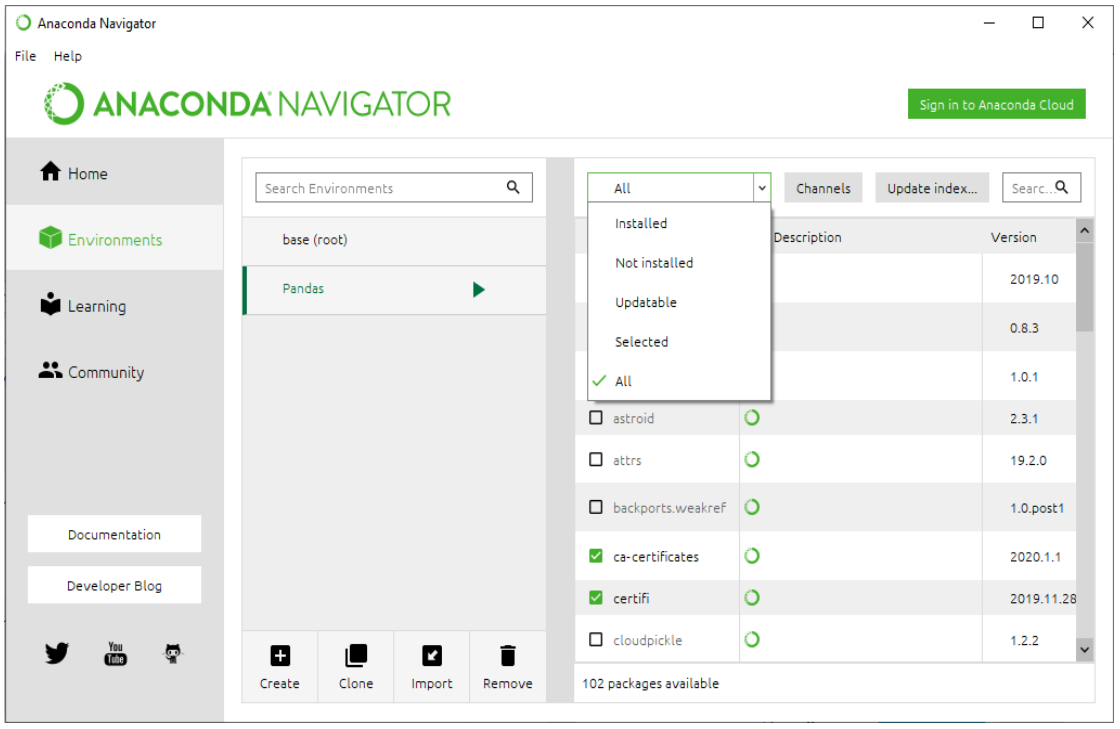
Step 3: Give a name to your Environment, e.g. Pandas and then choose a python version to run in the environment. Now click on the Create button to create Pandas Environment.



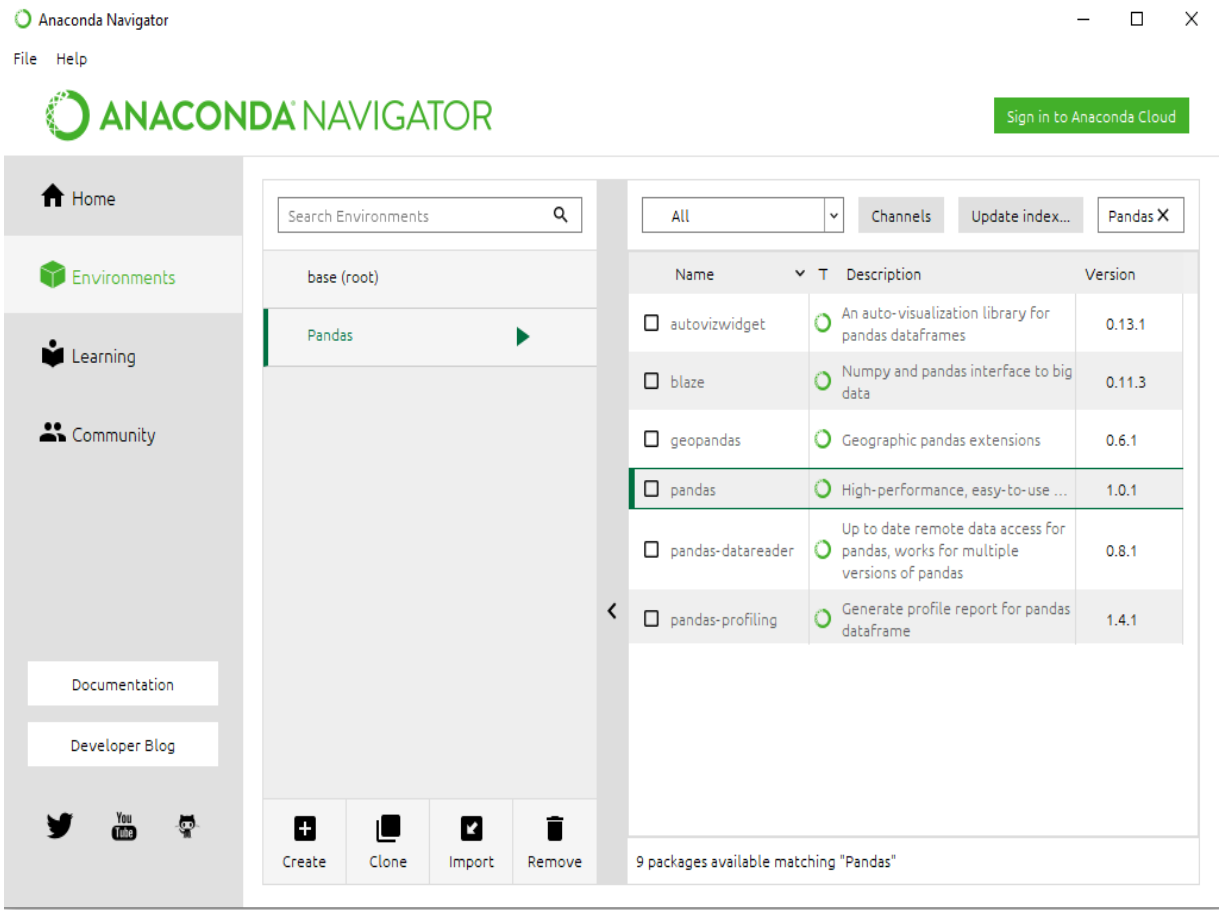
4: Now click on the **Pandas Environment** created to activate it.



Step 5: In the list above package names, select **All** to filter all the packages.

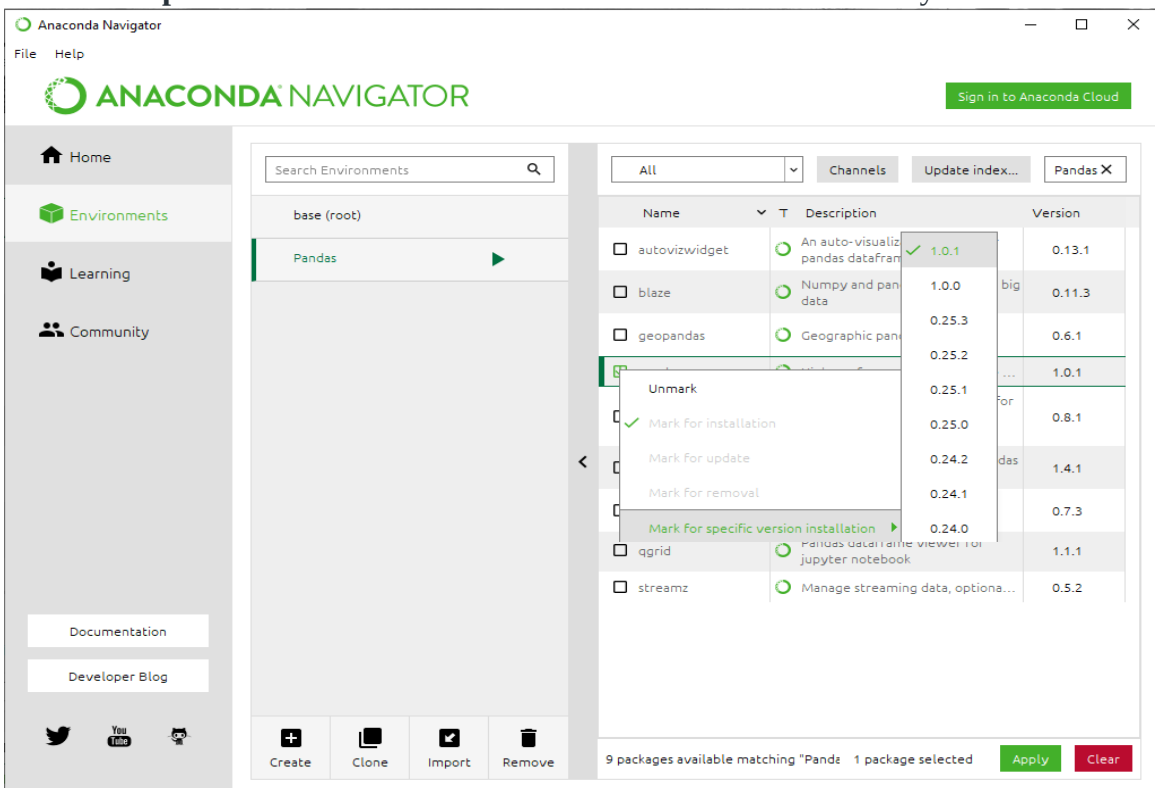


Step 6: Now in the Search Bar, look for '**Pandas**'. Select the **Pandas package** for Installation.

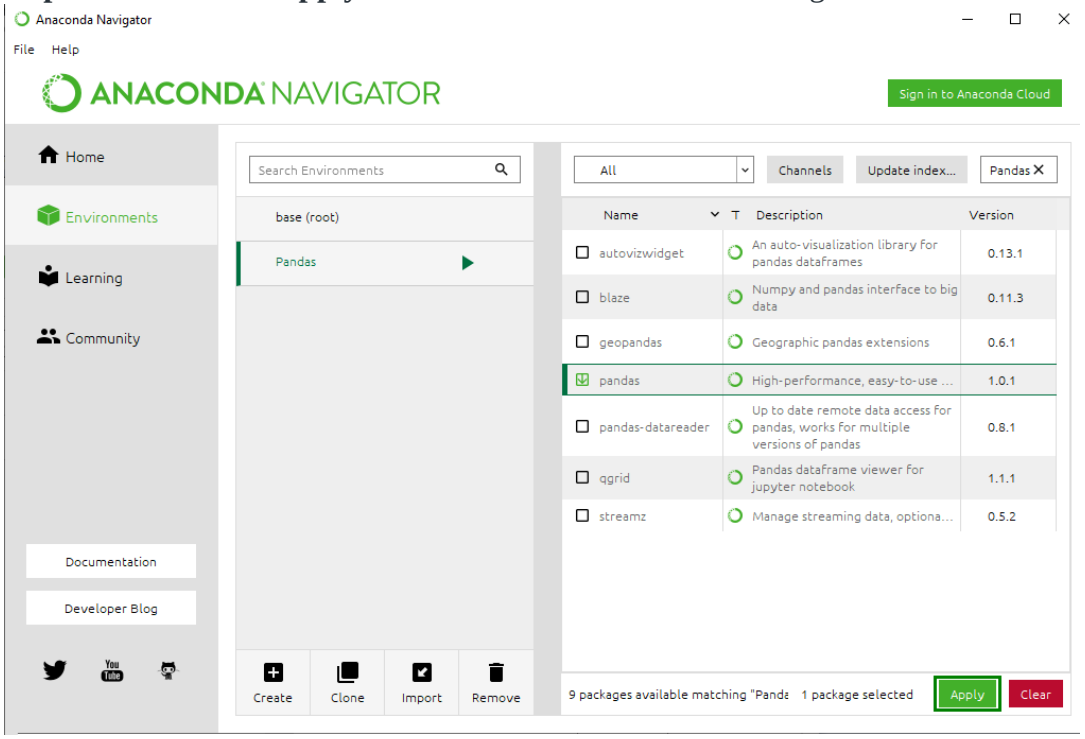


Step 7: Now Right Click on the checkbox given before the name of the package and then go to

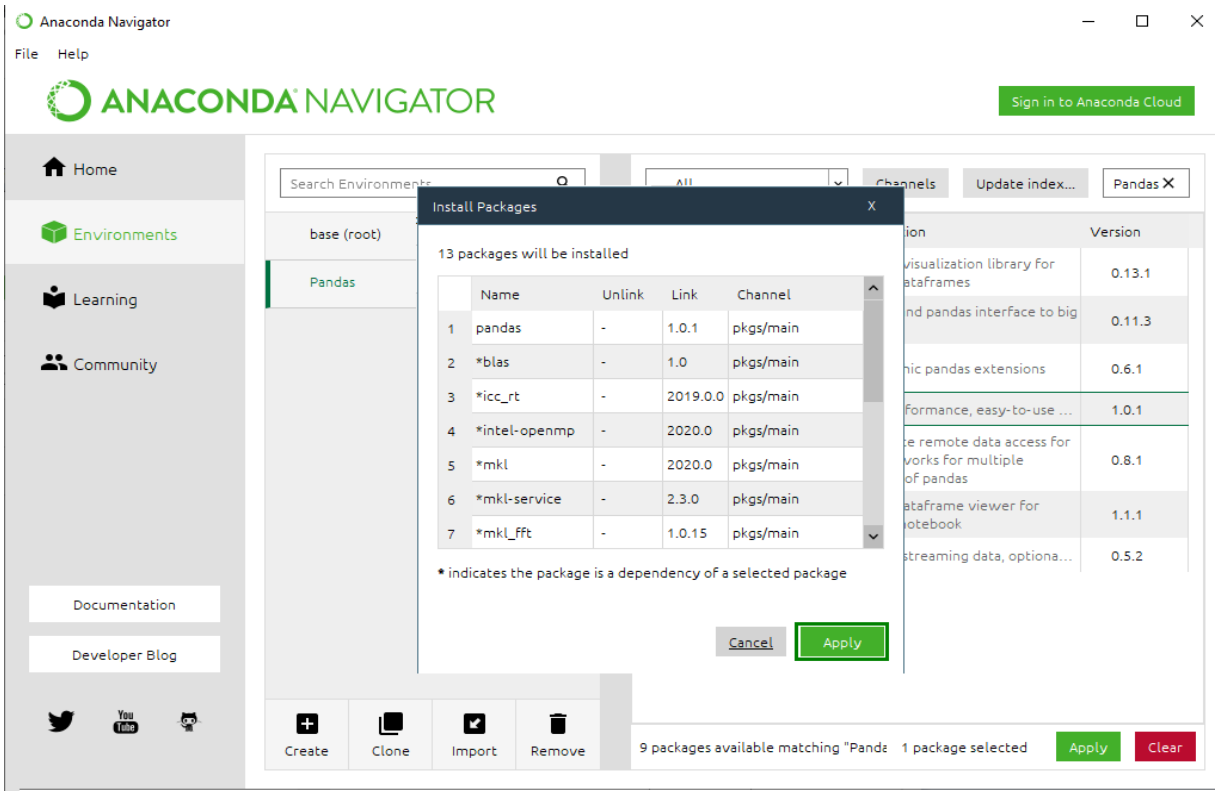
'**Mark for specific version installation**'. Now select the version that you want to install.



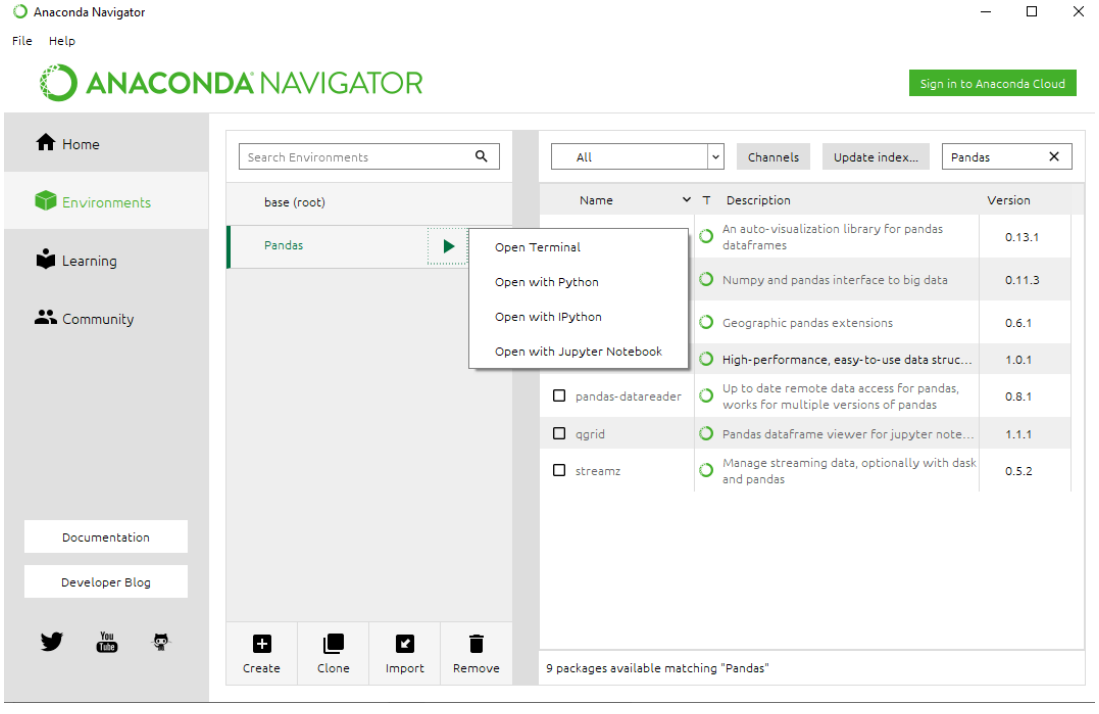
Step 8: Click on the **Apply** button to install the Pandas Package.



Step 9: Finish the Installation process by clicking on the **Apply** button.



Step 10: Now to open the Pandas Environment, click on the **Green Arrow** on the right of package name and select the Console with which you want to begin your Panda s programming.



Pandas Terminal Window:

INTRODUCING PANDAS OBJECTS

Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices.
three fundamental Pandas data structures:
the Series, DataFrame, and Index

The Pandas Series Object

A Pandas Series is a one-dimensional array of indexed data. It can be created from a list or array as follows:

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
data
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

As we see in the output, the Series wraps both a sequence of values and a sequence of indices, which we can access with the values and index attributes. The values are simply a familiar NumPy array:

```
data.values
```

```
array([ 0.25,  0.5 ,  0.75,  1.  ])
```

The index is an array-like object of type pd.Index, which we'll discuss in more detail momentarily.

```
data.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

Constructing Series objects

```
>>> pd.Series(data, index=index)
```

where index is an optional argument, and data can be one of many entities.

For example, data can be a list or NumPy array, in which case index defaults to an integer sequence:

```
pd.Series([2, 4, 6])
```

```
0    2
1    4
2    6
dtype: int64
```

data can be a scalar, which is repeated to fill the specified index:

```
pd.Series(5, index=[100, 200, 300])
```

```
100    5
200    5
300    5
dtype: int64
```

data can be a dictionary, in which index defaults to the sorted dictionary keys:

```
pd.Series({2:'a', 1:'b', 3:'c'})
```

```
1    b
2    a
3    c
dtype: object
```

In each case, the index can be explicitly set if a different result is preferred:

```
pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])
```

```
3    c
2    a
dtype: object
```

The Pandas DataFrame Object

The next fundamental structure in Pandas is the DataFrame. Like the Series object discussed in the p revious section, the DataFrame can be thought of either as a generalization of a NumPy array, or as a specialization of a Python dictionary. We'll now take a look at each of these perspectives.

DataFrame as a generalized NumPy array:

If a Series is an analog of a one-dimensional array with flexible indices, a DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names. Just as you might think of a two-dimensional array as an ordered sequence of aligned one-dimensional columns, you can think of a DataFrame as a sequence of aligned Series objects. Here, by "aligned" we mean that they share the same index.

To demonstrate this, let's first construct a new Series listing the area of each of the five states discussed in the previous section:

```
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
             'Florida': 170312, 'Illinois': 149995}
area = pd.Series(area_dict)
area
```

```
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662

dtype: int64
```

```
states = pd.DataFrame({'population': population,
                       'area': area})
states
```

Out[19]:

	area	population
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135

	area	population
New York	141297	19651127
Texas	695662	26448193

Data Indexing and selection:

Pandas now supports three types of Multi-axes indexing; the three types are mentioned in the following table −

Sr.No	Indexing & Description
1	**.loc()** Label based
2	**.iloc()** Integer based
3	**.ix()** Both Label and Integer based

loc()

Pandas provide various methods to have purely **label based indexing**. When slicing, the start bound is also included. Integers are valid labels, but they refer to the label and not the position.

.loc() has multiple access methods like −

- A single scalar label
- A list of labels
- A slice object
- A Boolean array

loc takes two single/list/range operator separated by ','. The first one indicates the row and the second one indicates columns.

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C',
'D'])

#select all rows for a specific column
print df.loc[:,'A']
```

Its **output** is as follows −

```
a   0.391548
b  -0.070649
c  -0.317212
d  -2.162406
e   2.202797
f   0.613709
g   1.050559
h   1.122680
Name: A, dtype: float64
```

Example 2

```
# import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C',
'D'])

# Select all rows for multiple columns, say list[]
print df.loc[:,['A','C']]
```

Its **output** is as follows −

```
          A         C
a  0.391548  0.745623
b -0.070649  1.620406
c -0.317212  1.448365
d -2.162406 -0.873557
e  2.202797  0.528067
f  0.613709  0.286414
g  1.050559  0.216526
h  1.122680 -1.621420
```

Example 3

```
# import the pandas library and aliasing as pd
import pandas as pd
```

```
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C',
'D'])

# Select few rows for multiple columns, say list[]
print df.loc[['a','b','f','h'],['A','C']]
```

Its **output** is as follows −

```
          A         C
a   0.391548  0.745623
b  -0.070649  1.620406
f   0.613709  0.286414
h   1.122680 -1.621420
```

Example 4

```
# import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C',
'D'])

# Select range of rows for all columns
print df.loc['a':'h']
```

Its **output** is as follows −

```
          A         B         C         D
a  0.391548 -0.224297  0.745623  0.054301
b -0.070649 -0.880130  1.620406  1.419743
c -0.317212 -1.929698  1.448365  0.616899
d -2.162406  0.614256 -0.873557  1.093958
e  2.202797 -2.315915  0.528067  0.612482
f  0.613709 -0.157674  0.286414 -0.500517
g  1.050559 -2.272099  0.216526  0.928449
h  1.122680  0.324368 -1.621420 -0.741470
```

Example 5

```
# import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C',
'D'])

# for getting values with a boolean array
print df.loc['a']>0
```

Its **output** is as follows −

```
A  False
B  True
C  False
D  False
Name: a, dtype: bool
```

.iloc()

Pandas provide various methods in order to get purely integer based indexing. Like python and numpy, these are **0-based** indexing.

The various access methods are as follows −

- An Integer
- A list of integers
- A range of values

Example 1

```
# import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C',
'D'])

# select all rows for a specific column
print df.iloc[:4]
```

Its **output** is as follows −

```
          A         B         C         D
0  0.699435  0.256239 -1.270702 -0.645195
1 -0.685354  0.890791 -0.813012  0.631615
2 -0.783192 -0.531378  0.025070  0.230806
3  0.539042 -1.284314  0.826977 -0.026251
```

Example 2

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C',
'D'])

# Integer slicing
print df.iloc[:4]
print df.iloc[1:5, 2:4]
```

Its **output** is as follows −

```
          A         B         C         D
0  0.699435  0.256239 -1.270702 -0.645195
1 -0.685354  0.890791 -0.813012  0.631615
2 -0.783192 -0.531378  0.025070  0.230806
3  0.539042 -1.284314  0.826977 -0.026251
```

```
           C           D
1  -0.813012    0.631615
2   0.025070    0.230806
3   0.826977   -0.026251
4   1.423332    1.130568
```

Example 3

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C',
'D'])

# Slicing through list of values
print df.iloc[[1, 3, 5], [1, 3]]
print df.iloc[1:3, :]
print df.iloc[:,1:3]
```

Its **output** is as follows −

```
           B           D
1   0.890791    0.631615
3  -1.284314   -0.026251
5  -0.512888   -0.518930

           A           B           C           D
1  -0.685354    0.890791   -0.813012    0.631615
2  -0.783192   -0.531378    0.025070    0.230806

           B           C
0   0.256239   -1.270702
1   0.890791   -0.813012
2  -0.531378    0.025070
3  -1.284314    0.826977
4  -0.460729    1.423332
5  -0.512888    0.581409
6  -1.204853    0.098060
7  -0.947857    0.641358
```

.ix()

Besides pure label based and integer based, Pandas provides a hybrid method for selections and subsetting the object using the .ix() operator.

Example 1

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C',
'D'])

# Integer slicing
```

```
print df.ix[:4]
```

Its **output** is as follows −

```
          A         B          C          D
0  0.699435  0.256239  -1.270702  -0.645195
1 -0.685354  0.890791  -0.813012   0.631615
2 -0.783192 -0.531378   0.025070   0.230806
3  0.539042 -1.284314   0.826977  -0.026251
```

Example 2

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C',
'D'])
# Index slicing
print df.ix[:,'A']
```

Its **output** is as follows −

```
0   0.699435
1  -0.685354
2  -0.783192
3   0.539042
4  -1.044209
5  -1.415411
6   1.062095
7   0.994204
Name: A, dtype: float64
```

Handling Missing Data :

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenario. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.
In Pandas missing data is represented by two value:

- None: None is a Python singleton object that is often used for missing data in Python code.
- NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()
- dropna()
- fillna()

- replace()
- interpolate()

In this article we are using CSV file, to download the CSV file used, Click Here.
Checking for missing values using isnull() and notnull()
In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

Checking for missing values using isnull()

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values.

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list
df = pd.DataFrame(dict)

# using isnull() function
df.isnull()
```

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.isnull(data["Gender"])

# filtering data
# displaying data only with Gender = NaN
data[bool_series]
```

As shown in the output image, only the rows having Gender = NULL are displayed.

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
20	Lois	NaN	4/22/1995	7:18 PM	64714	4.934	True	Legal
22	Joshua	NaN	3/8/2012	1:58 AM	90816	18.816	True	Client Services
27	Scott	NaN	7/11/1991	6:58 PM	122367	5.218	False	Legal
31	Joyce	NaN	2/20/2005	2:40 PM	88657	12.752	False	Product
41	Christine	NaN	6/28/2015	1:08 AM	66582	11.308	True	Business Development
49	Chris	NaN	1/24/1980	12:13 PM	113590	3.055	False	Sales
51	NaN	NaN	12/17/2011	8:29 AM	41126	14.009	NaN	Sales
53	Alan	NaN	3/3/2014	1:28 PM	40341	17.578	True	Finance
60	Paula	NaN	11/23/2005	2:01 PM	48866	4.271	False	Distribution
64	Kathleen	NaN	4/11/1990	6:46 PM	77834	18.771	False	Business Development
69	Irene	NaN	7/14/2015	4:31 PM	100863	4.382	True	Finance
70	Todd	NaN	6/10/2003	2:26 PM	84692	6.617	False	Client Services
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
939	Ralph	NaN	7/28/1995	6:53 PM	70635	2.147	False	Client Services
945	Gerald	NaN	4/15/1989	12:44 PM	93712	17.426	True	Distribution
961	Antonio	NaN	6/18/1989	9:37 PM	103050	3.050	False	Legal
972	Victor	NaN	7/28/2006	2:49 PM	76381	11.159	True	Sales
985	Stephen	NaN	7/10/1983	8:10 PM	85668	1.909	False	Legal
989	Justin	NaN	2/10/1991	4:58 PM	38344	3.794	False	Legal
995	Henry	NaN	11/23/2014	6:09 AM	132483	16.655	False	Distribution

145 rows × 8 columns

Checking for missing values using notnull()

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values.

```
# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, 45, 56, np.nan],

        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe using dictionary

df = pd.DataFrame(dict)

# using notnull() function
```

```
df.notnull()
```

Output:

	First Score	Second Score	Third Score
0	True	True	False
1	True	True	True
2	False	True	True
3	True	False	True

Filling missing values using fillna(), replace() and interpolate()
In order to fill null values in a datasets, we use fillna(), replace() and interpolate() function these function replace NaN values with some value of their own. All these function help in filling a null values in datasets of a DataFrame. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value.
Code #1: Filling null values with a single value

```
# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, 45, 56, np.nan],

        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)
```

```
# filling missing value using fillna()

df.fillna(0)
```

Output:

	First Score	Second Score	Third Score
0	100.0	30.0	0.0
1	90.0	45.0	40.0
2	0.0	56.0	80.0
3	95.0	0.0	98.0

Code #2: Filling null values with the previous ones

```
# importing pandas as pd

import pandas as pd



# importing numpy as np

import numpy as np



# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, 45, 56, np.nan],

        'Third Score':[np.nan, 40, 80, 98]}



# creating a dataframe from dictionary

df = pd.DataFrame(dict)
```

```
# filling a missing value with

# previous ones

df.fillna(method ='pad')
```

Output:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	90.0	56.0	80.0
3	95.0	56.0	98.0

Code #3: Filling null value with the next ones

```
# importing pandas as pd

import pandas as pd



# importing numpy as np

import numpy as np



# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, 45, 56, np.nan],

        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)
```

```
# filling  null value using fillna() function

df.fillna(method ='bfill')
```

Output:

	First Score	Second Score	Third Score
0	100.0	30.0	40.0
1	90.0	45.0	40.0
2	95.0	56.0	80.0
3	95.0	NaN	98.0

Filling null values in CSV File

```
# importing pandas package

import pandas as pd

# making data frame from csv file

data =  pd.read_csv("employees.csv")

# Printing the first 10 to 24 rows of

# the data frame for visualization

data[10:25]
```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
10	Louise	Female	8/12/1980	9:01 AM	63241	15.132	True	NaN
11	Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
12	Brandon	Male	12/1/1980	1:08 AM	112807	17.492	True	Human Resources
13	Gary	Male	1/27/2008	11:40 PM	109831	5.831	False	Sales
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14.543	True	Finance
15	Lillian	Female	6/5/2016	6:09 AM	59414	1.256	False	Product
16	Jeremy	Male	9/21/2010	5:56 AM	90370	7.369	False	Human Resources
17	Shawn	Male	12/7/1986	7:45 PM	111737	6.414	False	Product
18	Diana	Female	10/23/1981	10:27 AM	132940	19.082	False	Client Services
19	Donna	Female	7/22/2010	3:48 AM	81014	1.894	False	Product
20	Lois	NaN	4/22/1995	7:18 PM	64714	4.934	True	Legal
21	Matthew	Male	9/5/1995	2:12 AM	100612	13.645	False	Marketing
22	Joshua	NaN	3/8/2012	1:58 AM	90816	18.816	True	Client Services
23	NaN	Male	6/14/2012	4:19 PM	125792	5.042	NaN	NaN
24	John	Male	7/1/1992	10:08 PM	97950	13.873	False	Client Services

Now we are going to fill all the null values in Gender column with “No Gender”

```
# importing pandas package

import pandas as pd

# making data frame from csv file

data =  pd.read_csv("employees.csv")

# filling a null values using fillna()

data["Gender"].fillna("No Gender", inplace = True)

data
```

Output:

10	Louise	Female	8/12/1980	9:01 AM	63241	15.132	True	NaN
11	Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
12	Brandon	Male	12/1/1980	1:08 AM	112807	17.492	True	Human Resources
13	Gary	Male	1/27/2008	11:40 PM	109831	5.831	False	Sales
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14.543	True	Finance
15	Lillian	Female	6/5/2016	6:09 AM	59414	1.256	False	Product
16	Jeremy	Male	9/21/2010	5:56 AM	90370	7.369	False	Human Resources
17	Shawn	Male	12/7/1986	7:45 PM	111737	6.414	False	Product
18	Diana	Female	10/23/1981	10:27 AM	132940	19.082	False	Client Services
19	Donna	Female	7/22/2010	3:48 AM	81014	1.894	False	Product
20	Lois	No Gender	4/22/1995	7:18 PM	64714	4.934	True	Legal
21	Matthew	Male	9/5/1995	2:12 AM	100612	13.645	False	Marketing
22	Joshua	No Gender	3/8/2012	1:58 AM	90816	18.816	True	Client Services
23	NaN	Male	6/14/2012	4:19 PM	125792	5.042	NaN	NaN
24	John	Male	7/1/1992	10:08 PM	97950	13.873	False	Client Services

Code #5: Filling a null values using replace() method

```
# importing pandas package

import pandas as pd

# making data frame from csv file

data =  pd.read_csv("employees.csv")

# Printing the first 10 to 24 rows of

# the data frame for visualization

data[10:25]
```

Output:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
10	Louise	Female	8/12/1980	9:01 AM	63241	15.132	True	NaN
11	Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
12	Brandon	Male	12/1/1980	1:08 AM	112807	17.492	True	Human Resources
13	Gary	Male	1/27/2008	11:40 PM	109831	5.831	False	Sales
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14.543	True	Finance
15	Lillian	Female	6/5/2016	6:09 AM	59414	1.256	False	Product
16	Jeremy	Male	9/21/2010	5:56 AM	90370	7.369	False	Human Resources
17	Shawn	Male	12/7/1986	7:45 PM	111737	6.414	False	Product
18	Diana	Female	10/23/1981	10:27 AM	132940	19.082	False	Client Services
19	Donna	Female	7/22/2010	3:48 AM	81014	1.894	False	Product
20	Lois	NaN	4/22/1995	7:18 PM	64714	4.934	True	Legal
21	Matthew	Male	9/5/1995	2:12 AM	100612	13.645	False	Marketing
22	Joshua	NaN	3/8/2012	1:58 AM	90816	18.816	True	Client Services
23	NaN	Male	6/14/2012	4:19 PM	125792	5.042	NaN	NaN

Now we are going to replace the all Nan value in the data frame with -99 value.

```
# importing pandas package

import pandas as pd

# making data frame from csv file

data  = pd.read_csv("employees.csv")

# will replace  Nan value in dataframe with value -99

data.replace(to_replace = np.nan, value = -99)
```

Output:

10	Louise	Female	8/12/1980	9:01 AM	63241	15.132	True	-99
11	Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
12	Brandon	Male	12/1/1980	1:08 AM	112807	17.492	True	Human Resources
13	Gary	Male	1/27/2008	11:40 PM	109831	5.831	False	Sales
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14.543	True	Finance
15	Lillian	Female	6/5/2016	6:09 AM	59414	1.256	False	Product
16	Jeremy	Male	9/21/2010	5:56 AM	90370	7.369	False	Human Resources
17	Shawn	Male	12/7/1986	7:45 PM	111737	6.414	False	Product
18	Diana	Female	10/23/1981	10:27 AM	132940	19.082	False	Client Services
19	Donna	Female	7/22/2010	3:48 AM	81014	1.894	False	Product
20	Lois	-99	4/22/1995	7:18 PM	64714	4.934	True	Legal
21	Matthew	Male	9/5/1995	2:12 AM	100612	13.645	False	Marketing
22	Joshua	-99	3/8/2012	1:58 AM	90816	18.816	True	Client Services
23	-99	Male	6/14/2012	4:19 PM	125792	5.042	-99	-99
24	John	Male	7/1/1992	10:08 PM	97950	13.873	False	Client Services

Using interpolate() function to fill the missing values using linear method.

```
# importing pandas as pd

import pandas as pd

# Creating the dataframe

df = pd.DataFrame({"A":[12, 4, 5, None, 1],

                   "B":[None, 2, 54, 3, None],

                   "C":[20, 16, None, 3, 8],

                   "D":[14, 3, None, None, 6]})

# Print the dataframe

df
```

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

Let's interpolate the missing values using Linear method. Note that Linear method ignore the index and treat the values as equally spaced.

```
# to interpolate the missing values

df.interpolate(method ='linear', limit_direction ='forward')
```

Output:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	9.5	4.0
3	3.0	3.0	3.0	5.0
4	1.0	3.0	8.0	6.0

As we can see the output, values in the first row could not get filled as the direction of filling of values is forward and there is no previous value which could have been used in interpolation.

Dropping missing values using dropna()
In order to drop a null values from a dataframe, we used dropna() function this function drop Rows/Columns of datasets with Null values in different ways.
Dropping rows with at least 1 null value.

```
# importing pandas as pd

import pandas as pd

# importing numpy as np
```

```
import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, np.nan, 45, 56],

        'Third Score':[52, 40, 80, 98],

        'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

df
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52	NaN
1	90.0	NaN	40	NaN
2	NaN	45.0	80	NaN
3	95.0	56.0	98	65.0

Now we drop rows with at least one Nan value (Null value)

```
# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists
```

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
# creating a dataframe from dictionary
df = pd.DataFrame(dict)
# using dropna() function
df.dropna()
```

Output:

	First Score	Second Score	Third Score	Fourth Score
3	95.0	56.0	98	65.0

Dropping rows if all values in that row are missing.

```
# importing pandas as pd
import pandas as pd
# importing numpy as np
import numpy as np
# dictionary of lists
dict = {'First Score':[100, np.nan, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
```

```
        'Third Score':[52, np.nan, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)
df
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	NaN
1	NaN	NaN	NaN	NaN
2	NaN	45.0	80.0	NaN
3	95.0	56.0	98.0	65.0

Now we drop a rows whose all data is missing or contain null values(NaN)

```
# importing pandas as pd
import pandas as pd
# importing numpy as np
import numpy as np
# dictionary of lists
dict = {'First Score':[100, np.nan, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, np.nan, 80, 98],
```

```
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}

df = pd.DataFrame(dict)

# using dropna() function

df.dropna(how = 'all')
```

Output:

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	NaN
2	NaN	45.0	80.0	NaN
3	95.0	56.0	98.0	65.0

Dropping columns with at least 1 null value.

```
# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, np.nan, np.nan, 95],

        'Second Score': [30, np.nan, 45, 56],

        'Third Score':[52, np.nan, 80, 98],

        'Fourth Score':[60, 67, 68, 65]}

# creating a dataframe from dictionary
```

```
df =  pd.DataFrame(dict)

df
```

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	60
1	NaN	NaN	NaN	67
2	NaN	45.0	80.0	68
3	95.0	56.0	98.0	65

Now we drop a columns which have at least 1 missing values

```
# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, np.nan, np.nan, 95],

        'Second Score': [30, np.nan, 45, 56],

        'Third Score':[52, np.nan, 80, 98],

        'Fourth Score':[60, 67, 68, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

# using dropna() function
```

```
df.dropna(axis = 1)
```

Output :

	Fourth Score
0	60
1	67
2	68
3	65

Dropping Rows with at least 1 null value in CSV file

```
# importing pandas module

import pandas as pd

# making data frame from csv file

data  = pd.read_csv("employees.csv")

# making new data frame with dropped NA values

new_data = data.dropna(axis = 0, how ='any')

    new_data
```

Output:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
5	Dennis	Male	4/18/1987	1:35 AM	115163	10.125	False	Legal
6	Ruby	Female	8/17/1987	4:20 PM	65476	10.012	True	Product
8	Angela	Female	11/22/2005	6:29 AM	95570	18.523	True	Engineering
9	Frances	Female	8/8/2002	6:51 AM	139852	7.524	True	Business Development
11	Julie	Female	10/26/1997	3:19 PM	102508	12.637	True	Legal
12	Brandon	Male	12/1/1980	1:08 AM	112807	17.492	True	Human Resources
13	Gary	Male	1/27/2008	11:40 PM	109831	5.831	False	Sales
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14.543	True	Finance
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
993	Tina	Female	5/15/1997	3:53 PM	56450	19.040	True	Engineering
994	George	Male	6/21/2013	5:47 PM	98874	4.479	True	Marketing
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

764 rows × 8 columns

Now we compare sizes of data frames so that we can come to know how many rows had at least 1 Null value

```
print("Old data frame length:", len(data))

print("New data frame length:", len(new_data))

print("Number of rows with at least 1 NA value: ", (len(data)-len(new_data)))
```

Output :

Old data frame length: 1000

New data frame length: 764

Number of rows with at least 1 NA value: 236

Since the difference is 236, there were 236 rows which had at least 1 Null value in any column.

Merge and joining Data sets:

Pandas **merge()** is defined as the process of bringing the two datasets together into one and aligning the rows based on the common attributes or columns. It is an entry point for all standard database join operations between DataFrame objects:

Syntax:

1. pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
2. left_index=False, right_index=False, sort=True)

Parameters:

- **right:** *DataFrame or named Series*
  It is an object which merges with the DataFrame.
- **how:** *{'left', 'right', 'outer', 'inner'}, default 'inner'*
  Type of merge to be performed.
  - **left:** It use only keys from the left frame, similar to a SQL left outer join; preserve key order.
  - **right:** It use only keys from the right frame, similar to a SQL right outer join; preserve key order.
  - **outer:** It used the union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically.
  - **inner:** It use the intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.
- **on:** *label or list*
  It is a column or index level names to join on. It must be found in both the left and right DataFrames. If on is None and not merging on indexes, then this defaults to the intersection of the columns in both DataFrames.
  left_on: *label or list, or array-like*
  It is a column or index level names from the left DataFrame to use as a key. It can be an array with length equal to the length of the DataFrame.
- **right_on:** *label or list, or array-like*
  It is a column or index level names from the right DataFrame to use as keys. It can be an array with length equal to the length of the DataFrame.
- **left_index :** *bool, default False*
  It uses the index from the left DataFrame as the join key(s), If true. In the case of MultiIndex (hierarchical), many keys in the other DataFrame (either the index or some columns) should match the number of levels.

- **right_index :** *bool, default False*
  It uses the index from the right DataFrame as the join key. It has the same usage as the left_index.
- **sort:** *bool, default False*
  If True, it sorts the join keys in lexicographical order in the result DataFrame. Otherwise, the order of the join keys depends on the join type (how keyword).
- **suffixes:** *tuple of the (str, str), default ('_x', '_y')*
  It suffixes to apply to overlap the column names in the left and right DataFrame, respectively. The columns use (False, False) values to raise an exception on overlapping.
- **copy:** *bool, default True*
  If True, it returns a copy of the DataFrame.
  Otherwise, It can avoid the copy.
- **indicator:** *bool or str, default False*
  If True, It adds a column to output DataFrame "**_merge**" with information on the source of each row. If it is a string, a column with information on the source of each row will be added to output DataFrame, and the column will be named value of a string. The information column is defined as a categorical-type and it takes value of:
  - **"left_only"** for the observations whose merge key appears only in 'left' of the DataFrame, whereas,
  - **"right_only"** is defined for observations in which merge key appears only in 'right' of the DataFrame,
  - **"both"** if the observation's merge key is found in both of them.
- **validate:** *str, optional*
  If it is specified, it checks the merge type that is given below:
  - "one_to_one" or "1:1": It checks if merge keys are unique in both the left and right datasets.
  - "one_to_many" or "1:m": It checks if merge keys are unique in only the left dataset.
  - "many_to_one" or "m:1": It checks if merge keys are unique in only the right dataset.
  - "many_to_many" or "m:m": It is allowed, but does not result in checks.

Example1: Merge two DataFrames on a key

```
# import the pandas library
import pandas as pd
left = pd.DataFrame({
```

```
    'id':[1,2,3,4],
    'Name': ['John', 'Parker', 'Smith', 'Parker'],
    'subject_id':['sub1','sub2','sub4','sub6']})
right = pd.DataFrame({
     'id':[1,2,3,4],
    'Name': ['William', 'Albert', 'Tony', 'Allen'],
    'subject_id':['sub2','sub4','sub3','sub6']})
print (left)
print (right)
```

Output

```
    id      Name     subject_id

0   1       John        sub1
1   2       Parker      sub2
2   3       Smith       sub4
3   4       Parker      sub6

    id      Name     subject_id

0   1       William     sub2
1   2       Albert      sub4
2   3       Tony        sub3
3   4       Allen       sub6
```

Example2: Merge two DataFrames on multiple keys:

```
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
     'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left,right,on='id')
```

Output

```
    id   Name x    subject id x   Name y     subject id y
0   1     John       sub1          William     sub2
1   2     Parker     sub2          Albert      sub4
2   3     Smith      sub4          Tony        sub3
3   4     Parker     sub6          Allen       sub6
```

Aggregation and Grouping:

Pandas DataFrame.groupby()

In Pandas, **groupby()** function allows us to rearrange the data by utilizing them on real-world data sets. Its primary task is to split the data into various groups. These groups are categorized based on some criteria. The objects can be divided from any of their axes.

Syntax:

1. DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)

This operation consists of the following steps for aggregating/grouping the data:

- **Splitting datasets**
- **Analyzing data**
- **Aggregating or combining data**

Split data into groups

There are multiple ways to split any object into the group which are as follows:

- obj.groupby('key')
- obj.groupby(['key1','key2'])
- obj.groupby(key,axis=1)

Parameters of Groupby:

- **by:** *mapping, function, str, or iterable*

Its main task is to determine the groups in the groupby. If we use **by** as a function, it is called on each value of the object's index. If in case a dict or Series is passed, then the Series or dict VALUES will be used to determine the groups.

If a **ndarray** is passed, then the values are used as-is determine the groups.

We can also pass the label or list of labels to group by the columns in the **self**.

- **axis:** *{0 or 'index', 1 or 'columns'}, default value 0*
- **level:** *int, level name, or sequence of such, default value None.*

It is used when the axis is a MultiIndex (hierarchical), so, it will group by a particular level or levels.

- **as_index:** *bool, default True*
  It returns the object with group labels as the index for the aggregated output.
- **sort:** *bool, default True*
  It is used to sort the group keys. Get better performance by turning this off.

Pandas DataFrame.aggregate()

The main task of DataFrame.aggregate() function is to apply some aggregation to one or more column. Most frequently used aggregations are:

sum: It is used to return the sum of the values for the requested axis.

min: It is used to return the minimum of the values for the requested axis.

max: It is used to return the maximum values for the requested axis.

Syntax:

1. DataFrame.aggregate(func, axis=0, *args, **kwargs)

Parameters:

func: It refers callable, string, dictionary, or list of string/callables.

It is used for aggregating the data. For a function, it must either work when passed to a DataFrame or DataFrame.apply(). For a DataFrame, it can pass a dict, if the keys are the column names.

axis: (default 0): It refers to 0 or 'index', 1 or 'columns'

0 or 'index': It is an apply function for each column.

1 or 'columns': It is an apply function for each row.

***args:** It is a positional argument that is to be passed to **func**.

****kwargs:** It is a keyword argument that is to be passed to the **func**.

Returns:

It returns the scalar, Series or DataFrame.

scalar: It is being used when **Series.agg** is called with the single function.

Series: It is being used when DataFrame.agg is called for the single function.

DataFrame: It is being used when DataFrame.agg is called for the several functions.

Example:

```
import pandas as pd
import numpy as np
info=pd.DataFrame([[1,5,7],[10,12,15],[18,21,24],[np.nan,np.nan,np.nan]],columns=['X','Y','Z'])


info.agg(['sum','min'])
```

Output:

```
X      Y      Z
sum  29.0  38.0  46.0
min   1.0   5.0   7.0
```

Data Visualization using Matplotlib

Data Visualization is the process of presenting data in the form of graphs or charts. It helps to understand large and complex amounts of data very easily. It allows the decision-makers to make decisions very efficiently and also allows them in identifying new trends and patterns very easily. It is also used in high-level data analysis for Machine Learning and Exploratory Data Analysis (EDA). Data visualization can be done with various tools like Tableau, Power BI, Python.

Matplotlib

Matploptib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consist of several plots like line chart, bar chart, histogram, etc. It provides a lot of flexibility but at the cost of writing more code.

Pyplot

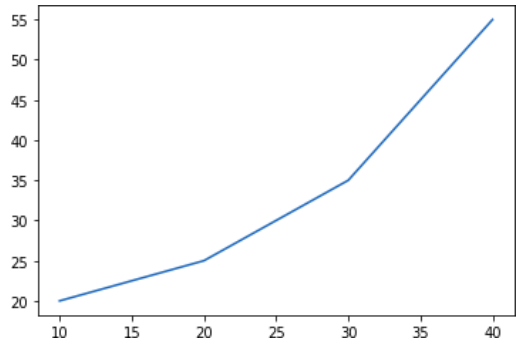
Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. The various plots we can utilize using Pyplot are Line Plot, Histogram, Scatter, 3D Plot, Image, Contour, and Polar.

After knowing a brief about Matplotlib and pyplot let's see how to create a simple plot.

Example:

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
plt.show()
```

Output:



Adding Title

The title() method in matplotlib module is used to specify the title of the visualization depicted and displays the title using various attributes.

Syntax:

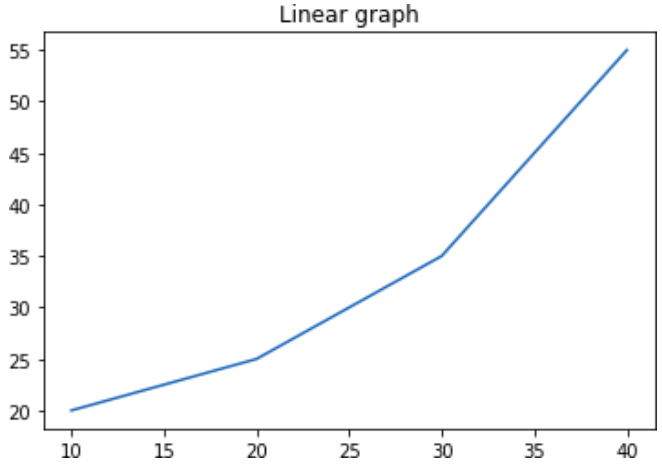
*matplotlib.pyplot.title(label, fontdict=None, loc='center', pad=None, **kwargs)*

Example:

```
import matplotlib.pyplot as plt

 # initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

 # plotting the data

plt.plot(x, y)

# Adding title to the plot

plt.title("Linear graph")
```

```
plt.show(}
```

Output:



Adding X Label and Y Label

In layman's terms, the X label and the Y label are the titles given to X-axis and Y-axis respectively. These can be added to the graph by using the **xlabel()** and **ylabel()** methods.

Syntax:

*matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, **kwargs)*

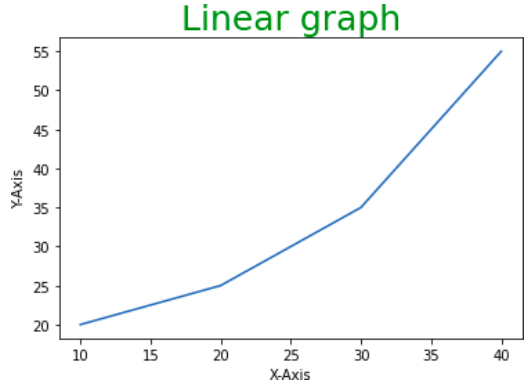
*matplotlib.pyplot.ylabel(ylabel, fontdict=None, labelpad=None, **kwargs)*

Example:

```
import matplotlib.pyplot as plt

 # initializing the data

x = [10, 20, 30, 40]
```

```
y = [20, 25, 35, 55]

# plotting the data

plt.plot(x, y)

# Adding title to the plot

plt.title("Linear graph", fontsize=25, color="green")

# Adding label on the y-axis

plt.ylabel('Y-Axis')

# Adding label on the x-axis

plt.xlabel('X-Axis')

plt.show()
```

Output:



Adding Legends

A legend is an area describing the elements of the graph. In simple terms, it reflects the data displayed in the graph's Y-axis. It generally appears as the box containing a small sample of each color on the graph and a small description of what this data means.

The attribute bbox_to_anchor=(x, y) of legend() function is used to specify the coordinates of the legend, and the attribute ncol represents the number of columns that the legend has. Its default value is 1.

Syntax:

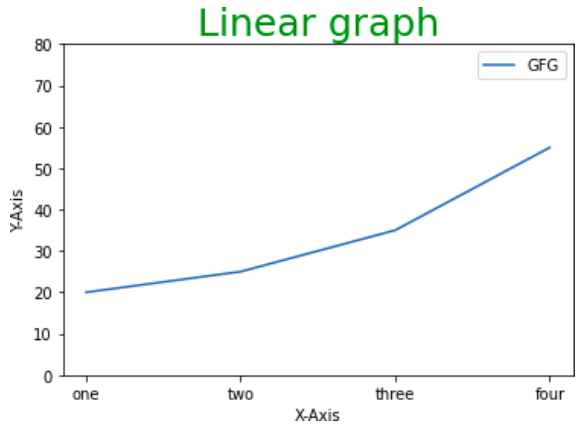
matplotlib.pyplot.legend([“name1”, “name2”], bbox_to_anchor=(x, y), ncol=1)

Example:

```
import matplotlib.pyplot as plt

 # initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

# plotting the data

plt.plot(x, y)

# Adding title to the plot

plt.title("Linear graph", fontsize=25, color="green")

# Adding label on the y-axisplt.ylabel('Y-Axis')

# Adding label on the x-axis

plt.xlabel('X-Axis')

 # Setting the limit of y-axis

plt.ylim(0, 80)

 # setting the labels of x-axis

plt.xticks(x, labels=["one", "two", "three", "four"])

# Adding legends
```

```
plt.legend(["GFG"])

plt.show()
```

Output:

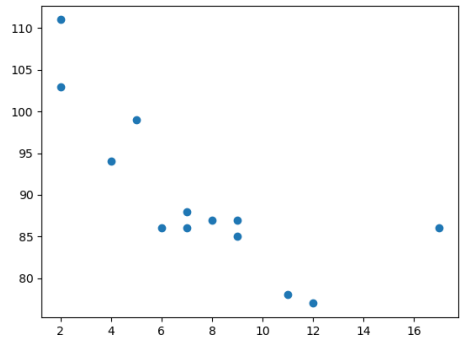


Simple Scatter plots:

Creating Scatter Plots

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

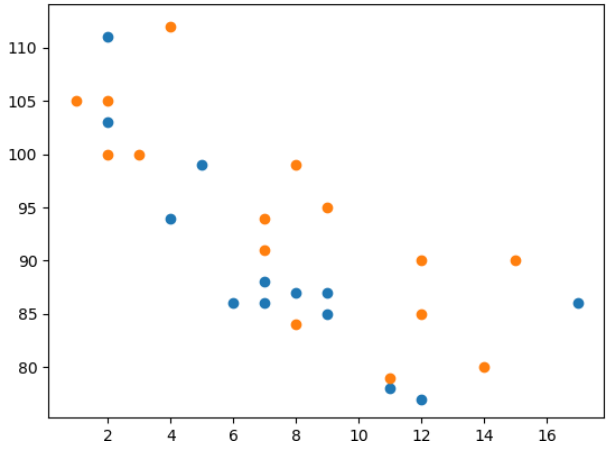
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



Compare Plots

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

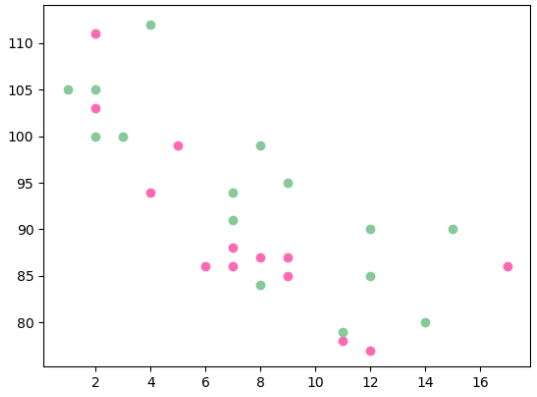
```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)
plt.show()
```



Colors

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')
plt.show()
```

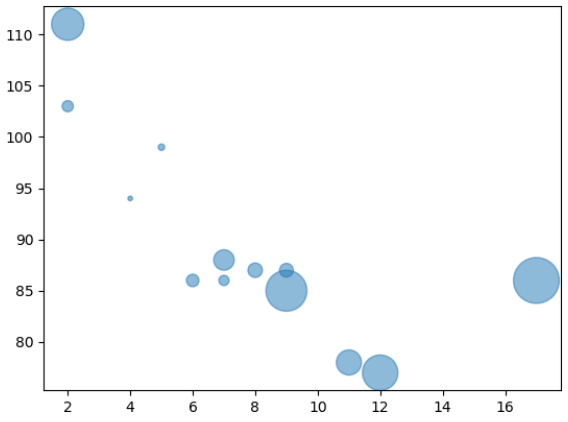


Alpha

You can adjust the transparency of the dots with the alpha argument.

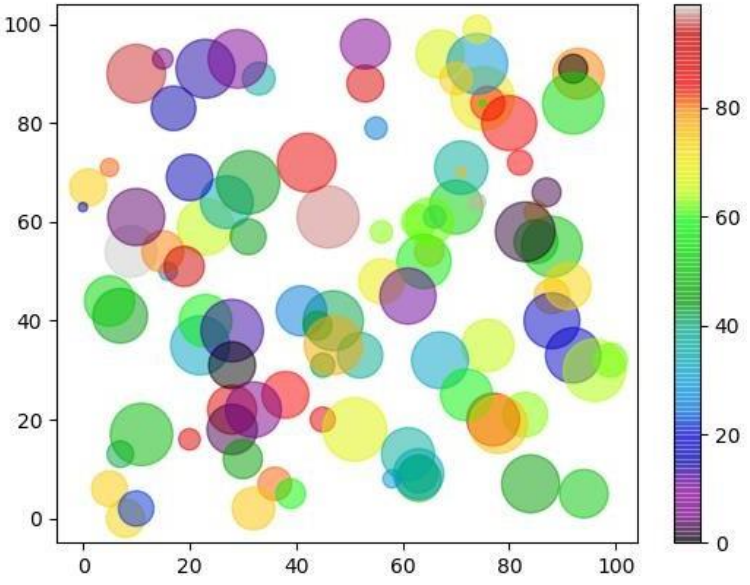
Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```



Combine Color Size and Alpha

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```

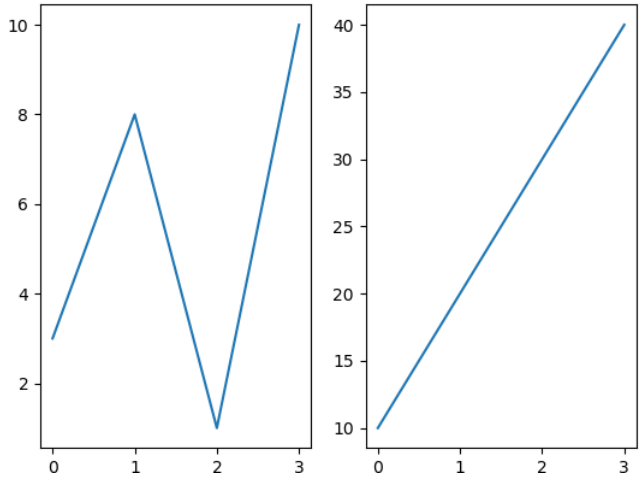


Matplotlib Subplot:

Display Multiple Plots

With the subplot() function you can draw multiple plots in one figure:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



The subplot() Function

The subplot() function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the *first* and *second* argument.

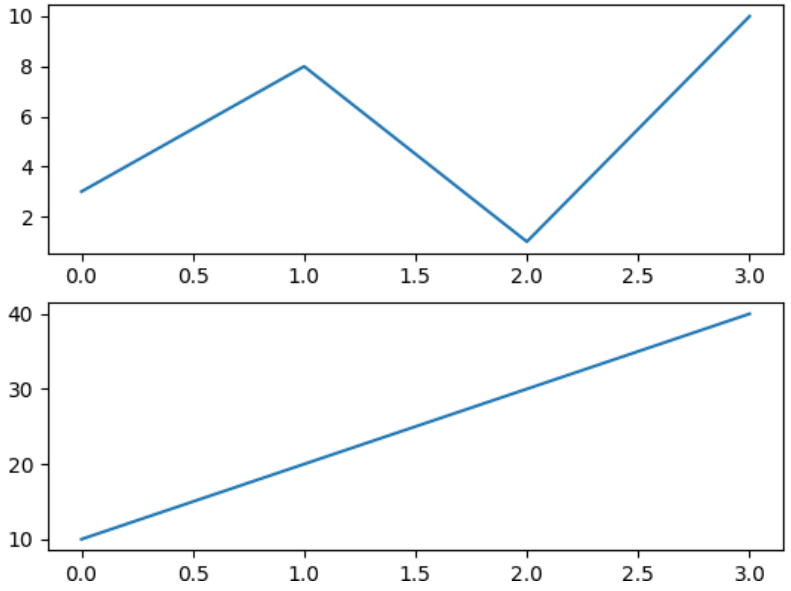
The third argument represents the index of the current plot.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```

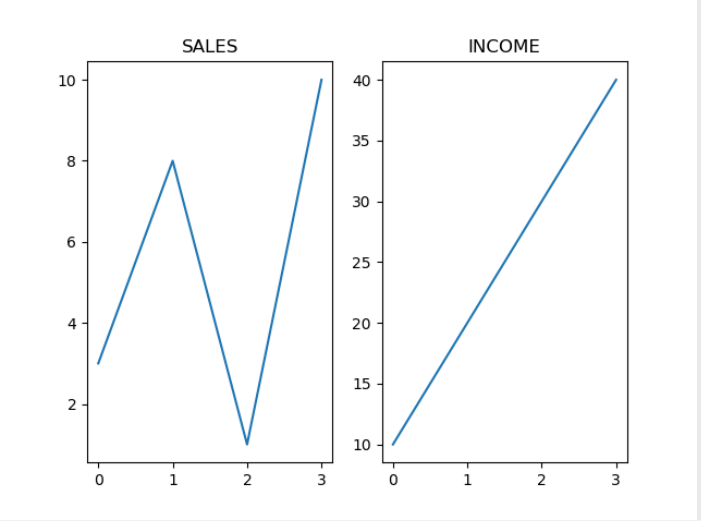


Title

You can add a title to each plot with the `title()` function:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.show()
```

Result:



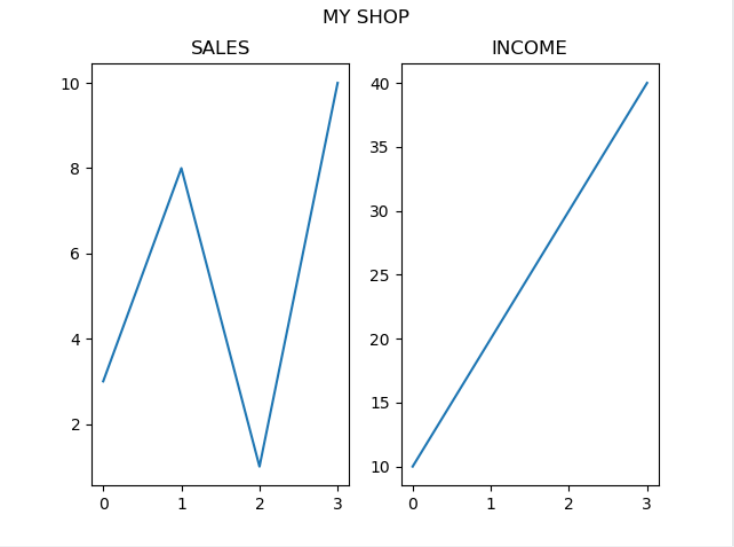
Super Title

You can add a title to the entire figure with the `suptitle()` function:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
```

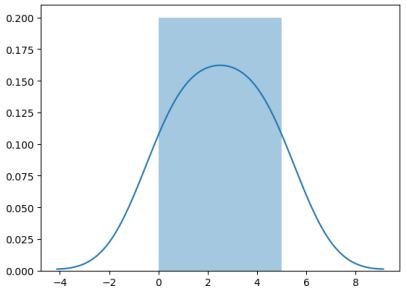
```
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

Result:



Visualize Distributions with Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.



Install Seaborn.

If you have Python and PIP already installed on a system, install it using this command:

```
C:\Users\Your Name>pip install seaborn
```

If you use Jupyter, install Seaborn using this command:

```
C:\Users\Your Name>!pip install seaborn
```

Distplots

Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

Import Matplotlib

Import the pyplot object of the Matplotlib module in your code using the following statement:

```
import matplotlib.pyplot as plt
```

Import Seaborn

Import the Seaborn module in your code using the following statement:

```
import seaborn as sns
```

Plotting a Distplot

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5])
plt.show()
```

Plotting a Distplot Without the Histogram

Example

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5], hist=False)
plt.show()
```