Dr. SNS RAJALAKSHMI COLLEGE OF ARTS & SCIENCE

(AUTONOMOUS)

Accredited by NAAC (Cycle III) with 'A+' Grade

Affiliated to Bharathiar University

Coimbatore-641049

DEPARTMENT OF COMPUTER APPLICATIONS

II BCA

ELECTIVE TRACK 2 - HIGHER EDUCATION

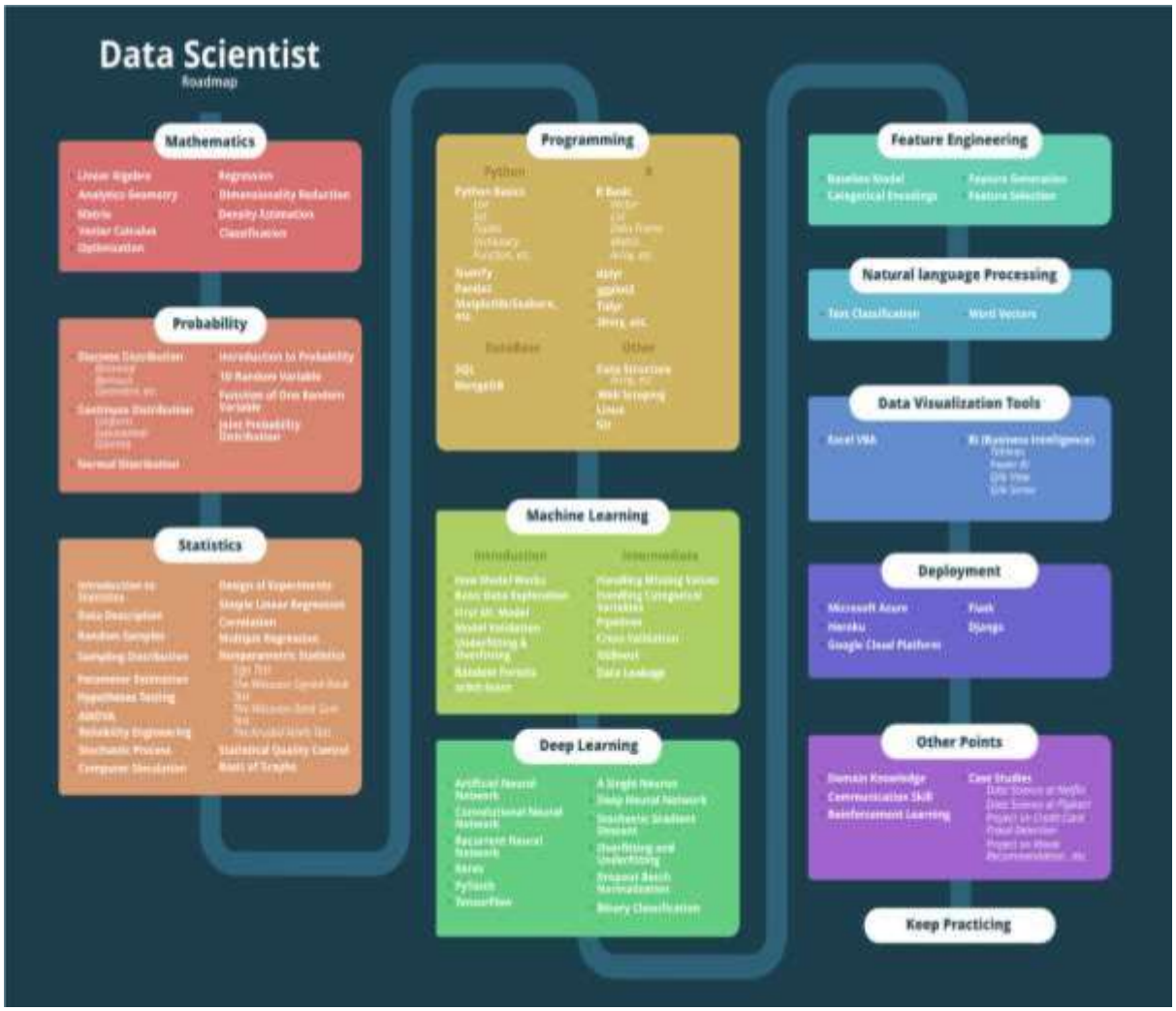
21UCU807 : DATA SCIENCE

UNIT-I

INTRODUCTION:

Data science is a field that involves using statistical and computational techniques to extract insights and knowledge from data. It encompasses a wide range of tasks, including data cleaning and preparation, data visualization, statistical modelling, machine learning, and more. Data scientists use these techniques to discover patterns and trends in data, make predictions, and support decision-making. They may work with a variety of data types, including structured data (such as numbers and dates in a spreadsheet) and unstructured data (such as text, images, or audio). Data science is used in a wide range of industries, including finance, healthcare, retail, and more.

DATA SCIENCE ROAD MAP:



FRAME THE PROBLEM:

Problem framing is setting up your business problem in a way that can be addressed with data. It's the process of taking an abstract goal like "we want to know which customers are likely to churn" and translating it into what data will be used, how the data will look, and what modelling approaches might be applicable. It's a combination of understanding the underlying mechanics of your problem and matching your problem to techniques and approaches that fit.

Problem framing is enacted in the way that you compile your data for analysis (what are your potential features? How are you defining your target? What is the appropriate level of granularity of your analysis?) And the techniques used to address your problem

Problem framing should be the first thing a data scientist does when working on a new project. The process of problem framing involves asking questions about the system you're trying to model. It typically includes:

Defining the dependent variable
Defining the level of granularity of your analysis
Assessing data availability
Defining potential features
Defining potential modelling approaches

SIX ADDITIONAL TIPS FOR PROBLEM FRAMING:

Don't be afraid to ask simple or "dumb" questions. These are necessary to get a good understand of the system you're modelling.

Research the problem! There are typically blog posts, research papers, or instructional videos about what you're trying to model.

Use them and pull from existing knowledge bases.
Reach out to others you think can help and collaborate. Problem framing can be a great project step for data scientists to collaborate and learn from one another.

Consider the timing of your data. Know when it's available and what will be known at the time of prediction when you're creating your data set.

Simplify, simplify, simplify. We often see data scientists relying on overly complex models when a simpler option might be the better choice. Don't assume complex is better. The simple solution might be the right one.

Know what information is important for your problem and define what success looks like before you move to problem framing.

To use the customer churn example, do you want an accurate prediction of whether each customer will churn or do you want a list of the 100 customers most likely to churn so you can target them with some sort of promotion? This will inform the output you're looking for from your model.

UNDERSTAND THE DATA:

Data understanding involves accessing the data and exploring it using tables and graphics that can be organized in IBM, SPSS, and Modeller using the CRISP-DM project tool.

This enables you to determine the quality of the data and describe the results of these steps in the project documentation.

COLLECTING DATA:

- **Existing data:**

  This includes a wide variety of data, such as transactional data, survey data, Web logs, etc. Consider whether the existing data are enough to meet your needs.

- **Purchased data:**

  Does your organization use supplemental data, such as demographics? If not, consider whether it may be needed.

- **Additional data:**

  If the above sources don't meet your needs, you may need to conduct surveys or begin additional tracking to supplement the existing data stores.

DESCRIBING DATA:

- **Amount of data:**

  For most modelling techniques, there are trade-offs associated with data size. Large data sets can produce more accurate models, but they can also lengthen the processing time. Consider whether using a subset of data is a possibility. When taking notes for the final report, be sure to include size statistics for all data sets, and remember to consider both the number of records as well as fields (attributes) when describing data.

- **Value types:**

  Data can take a variety of formats, such as numeric, categorical (string), or Boolean (true/false). Paying attention to value type can head off problems during later modelling.

- **Coding schemes:**

  Frequently, values in the database are representations of characteristics such as gender or product type. For example, one data set may use *M* and *F* to represent *male* and *female*, while another may use the numeric values *1* and *2*. Note any conflicting schemes in the data report.

VERIFYING DATA:

- **Missing data:**

  It include values that are blank or coded as a non-response (such as *$null$*, or *999*).

- **Data errors:**

  Data errors are usually typographical errors made in entering the data.

- **Measurement errors:**

  It include data that are entered correctly but are based on an incorrect measurement scheme.

- **Coding inconsistencies:**

  It typically involve nonstandard units of measurement or value inconsistencies, such as the use of both *M* and *male* for gender.

- **Bad metadata**:

  It include mismatches between the apparent meaning of a field and the meaning stated in a field name or definition.

EXPLORING DATA:

Data exploration is the first step of data analysis used to explore and visualize data to uncover insights from the start or identify areas or patterns to dig into more.

Using interactive dashboards and point-and-click data exploration, users can better understand the bigger picture and get to insights faster.

MODEL:

There are three types of data models: dimensional, relational, and entity relational. These models follow three approaches: conceptual, logical, and physical. Other data models are also there; however, they are obsolete, such as network, hierarchical, object-oriented, and multi-value.

DATA MODELING EXAMPLES:

The best way to picture a data model is to think about a building plan of an architect. An architectural building plan assists in putting up all subsequent conceptual models, and so does a data model.

These data modelling examples will clarify how data models and the process of data modelling highlights essential data and the way to arrange it.

1. ER (ENTITY-RELATIONSHIP) MODEL:

This model is based on the notion of real-world entities and relationships among them. It creates an entity set, relationship set, general attributes, and constraints.

Here, an entity is a real-world object; for instance, an employee is an entity in an employee database. An attribute is a property with value, and entity sets share attributes of identical value. Finally, there is the relationship between entities.

2. HIERARCHICAL MODEL:

This data model arranges the data in the form of a tree with one root, to which other data is connected. The hierarchy begins with the root and extends like a tree. This model effectively explains several real-time relationships with a single one-to-many relationship between two different kinds of data.

For example, one supermarket can have different departments and many aisles. Thus, the 'root' node supermarket will have two 'child' nodes of (1) Pantry, (2) Packaged Food.

3. NETWORK MODEL:

This database model enables many-to-many relationships among the connected nodes. The data is arranged in a graph-like structure, and here 'child' nodes can have multiple 'parent' nodes. The parent nodes are known as owners, and the child nodes are called members.

4. RELATIONAL MODEL:

This popular data model example arranges the data into tables. The tables have columns and rows, each cataloguing an attribute present in the entity. It makes relationships between data points easy to identify.

For example, e-commerce websites can process purchases and track inventory using the relational model.

5. OBJECT-ORIENTED DATABASE MODEL:

This data model defines a database as an object collection, or recyclable software components, with related methods and features.

For instance, architectural and engineering real-time systems used in 3D modelling use this data modelling process.

6. OBJECT-RELATIONAL MODEL:

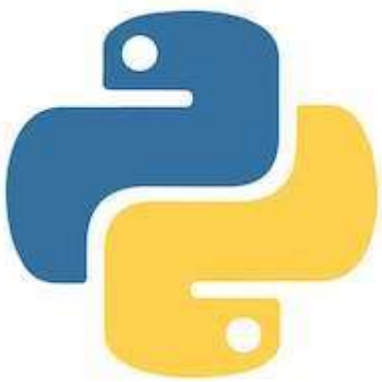
This model is a combination of an object-oriented database model and a relational database model. Therefore, it blends the advanced functionalities of the object-oriented model with the ease of the relational data model.

The data modelling process helps organizations to become more data-driven. This starts with cleaning and modelling data. Let us look at how data modelling occurs at different levels.

These were the important types we discussed in what is data modelling. Next, let's have a look at the techniques.

A SURVEY OF PROGRAMMING LANGUAGES FOR DATA SCIENCE:

PYTHON:



Whether it's the most popular programming language in the world or simply in the top three, Python has undoubtedly boomed in recent years. Even though it's been around for 30 years, the language finally began to catch on more broadly around 2007, when Python-heavy Dropbox launched, providing extensive real-world proof of concept for its strengths. (And the fact that Google signed on around the same time was also huge.) But nowhere is Python's popularity more evident than in data science circles, where it's often considered the go-to.

Python has become widely used for several reasons. It prioritizes readability, it's dynamically typed and it sports intuitive syntax, which makes it relatively easy to learn and use. Also, it has an incredibly rich ecosystem of libraries for data pre-processing and analysis (NumPy, SciPy, and Pandas), visualization (Matplotlib, Seaborne, Bokeh) and more. Perhaps most notably, as machine learning and deep learning matured and grew mainstream, so too did Python, which added watershed platforms and libraries like scikit-learn, Keras, the Facebook developed

And as Python's traction has cemented, so too has its support resources. The language has a large, dedicated support community and a plethora of Python-specific books and boot camps and courses — all reasons, perhaps, the language regularly lands among the "most loved" in Stack Overflow's annual developer surveys.

R:

Even though its reputation as less-than-desirable for production is something of a misnomer, R is still considered best-suited for data mining and statistical analysis, of which it offers a wide range of options. (Indeed, R's favoured status among statisticians is something of a chronic meme-generator.) It's also regarded as more approachable than Python for no developers, since it's possible to whip up a statistical model — and a sharp-looking visualization — with just a few lines of code.

R also sports a robust constellation of packages. Most notable is the tidy verse family, created by R-community icon Hadley Wickham. It features popular packages for organizing data (tidyr), mugging data (dplyr) and visualization (the ground-breaking ggplot2).

You can see it in action via the much-loved #Tidy Tuesday project. Each week, a new data set is released for data scientists to practice and demonstrate their data wrangling and visualization skills. It's the kind of learning (and honing) in public that's also emblematic of R's famously supportive — and inclusive — community.

JULIA:

Much has been made of Julia's swift ascendancy in data science circles over the last few years — and for good reason. It's now a top 20 language in the IEEE Spectrum rankings, and it also cracked the top 20 of the influential TIOBE index last year. (As of writing, it stands at #28).

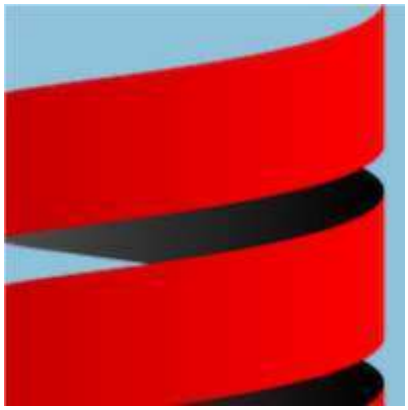
As developer Anupam Chugh noted in Built In, Julia is faster than Python; dynamic, yet more type-safe (thanks to its just-in-time compiler and multiple dispatch); and better equipped for distributed and parallel computing.

That power has made it an emergent option for big-data analysis in the private sector (BlackRock, Apple, Oracle and Google are all users) and, especially, for scientific research, where it's being used in noteworthy projects for climate modelling, weather forecasting and astronomical surveys, among others. Its wide-ranging interoperability — compatible with everything from Python to old-school Fortran — and its continued boost from MIT, where it has roots, also make Julia a likely long-term contender. And like Python, it too garners a lot of love.

JAVA:

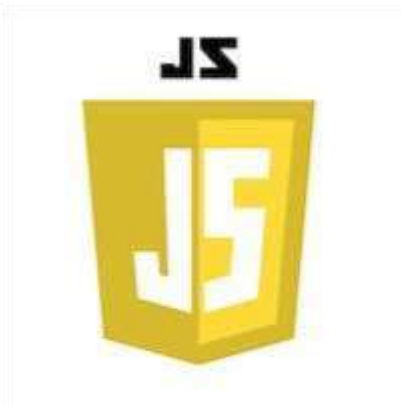
Another object-oriented language, the "write once, run anywhere" Java is most associated with web development, but it has a prominent role in data contexts, in large part because Java Virtual Machines (JVMs) are a common component in big-data frameworks. The Apache Hadoop ecosystem (including Hive, Spark and Map Reduce) relies on JVMs, which means at least a passing familiarity will be helpful for efficiently running data jobs that require large storage and processing requirements.

SCALA:



Speaking of Java, Scala was explicitly designed to be a cleaner, less wordy alternative to that popular language. "You can write hundreds of lines of confusing-looking Java code in less than 15 lines in Scala," wrote Packt. Designed in Switzerland and released in 2004, Scala runs on the same JVMs mentioned above, which makes it a hand-in-glove fit for distributed big-data projects and data pipelines. A principal software engineer at Seattle-based Hiya told Built In in 2019 that he's built petabyte-scale data projects on Scala.

JAVASCRIPT:



The same reason that JavaScript remains a web-programming workhorse — it's ability to specify page behaviour — is also what makes it great for data visualization. For example, D3.js, one of the most versatile visualization libraries for building dynamic, interactive viz, is a JavaScript library. So the more conversant you are, the more granular you can go on Observable-style presentations.

SQL:

SQL sometimes isn't even considered a proper programming language, since it's domain-specific. That's silly of course, and just because you can't, say, build an app with SQL doesn't mean it's not a must-know. SQL extensions can be added to allow more complex processes to run alongside a database, but it's primarily used as a way to communicate with relational databases. Most data scientists won't be futzing with actual database administration, but querying the data — and being able to investigate the nuances of how data might be manipulated by the database — are crucial skills to have.

STRINGS:

A string is generally considered as a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. String may also denote more general arrays or other sequence (or list) data types and structures.

Most Useful String Methods for a Data Scientist:

In simple words, Machine Learning is training/teaching algorithms with historical data to predict output on unseen data. Most of the times, the type of data is in the form of text. When working with text data, one must be familiar with python's available string methods to make life easier.

In this post, I'll talk about some of the string methods that I personally found very useful while handling text data.

1. split()

split() separates the string into words based on the pre-defined **separator.** It returns a list of the words in the string.
similar functions : rsplit()

Syntax: `str.` **split**(*sep=None*,

maxsplit=-1)

- **sep -** separator used to break the string into words, uses ***white space*** as the default separator
- ***maxsplit*** - *max* number of splits to be done(the list will have at most `maxsplit+1` elements), uses `-1`(no limit on the number of splits) *as maxsplit* if not specified.

2. strip() strip() removes the leading(beginning) and trailing(ending) spaces of the

string. **similar functions** : rstrip() & lstrip()

Syntax `str.`**strip**([*chars*])

- ***chars*** - set of characters to be removed, default it removes whitespace.

3. replace() replace() is used to replace all old substring of the string with *new*.

Syntax `str.`**replace**(*old*, *new*[, *count*])

old - old substring to look for

- **new** - new substring to replace the old substring with
- **count** - number of times to place old substring with a new substring.

4. join() join() is used to concatenate/join the strings in an *iterable* with a string separator.

Syntax

`str.`**join**(*iterable*)

- ***iterable*** - like list, tuple, string etc.

5. lower() lower() converts all the characters of a string to lowercase. **similar functions** : upper()

Syntax

`str.`**lower**()

6. count()

count() returns the number of times a substring appeared in a string.

Syntax `str.`**count**(*sub*[,

start[, *end*]])

- **sub** - substring to search for
- **start** - starting index to search the substring in the given string, default index is 0.
- **end** - ending index of the string, default is the end of the string.

7. isdigit()

isdigit() returns `True` if all the characters in the given string are digits, returns `False` if at least one character is other than a digit.

NOTE: isdigit() is very useful in ML preprocessing to check if any value in the columns of a Dataframe is a digit. Sometimes, you may find special characters(' ', ? etc) in place of values.

Syntax

`str.`**isdigit**()

8. casefold()

casefold() is used for caseless matching. It is similar to lower(), but more aggressive because it is intended to remove all case distinctions in a string.

Syntax `str.`**casefold**()

9.find()

find() returns the position/index of the first occurrence of the specified substring in the given string, returns `-1` if the substring is not found.

similar functions : rfind()

Syntax `str.`**find**(*sub*[,

start[, *end*]])

sub - substring to search in the given string

DEFINING FUNCTION:

The purpose of defining a function is to give a name to a computational process that may be applied multiple times. There are many situations in computing that require repeated computation. For example, it is often the case that we want to perform the same manipulation on every value in a column of a table.

A data scientist's main role is to organise and analyse large portions of data through custom-designed analytical software, in order to provide stakeholders with findings that they can use to make informed business decisions.

PYTHON:

This data science with Python tutorial will help you learn the basics of Python along with different steps of data science such as data pre-processing, data visualization, statistics, making machine learning models, and much more with the help of detailed and well-explained examples. This tutorial will help both beginners as well as some trained professionals in mastering data science with Python**.**

Python is a programming language widely used by Data Scientists.

Python has in-built mathematical libraries and functions, making it easier to calculate mathematical problems and to perform data analysis.

Data science" is just about as broad of a term as they come. It may be easiest to describe what it is by listing its more concrete components:

DATA EXPLORATION & ANALYSIS:

- Included here: Pandas; NumPy; SciPy; a helping hand from Python's Standard Library.

DATA VISUALIZATION:

A pretty self-explanatory name. Taking data and turning it into something colorful.

- Included here: Matplotlib; Seaborn; Datashader; others.

CLASSICAL MACHINE LEARNING:

Conceptually, we could define this as any supervised or unsupervised learning task that is not deep learning (see below). Scikit-learn is far-and-away the go-to tool for implementing classification, regression, clustering, and dimensionality reduction, while StatsModels is less actively developed but still has a number of useful features.

- Included here: Scikit-Learn, StatsModels.

DEEP LEARNING:

This is a subset of machine learning that is seeing a renaissance, and is commonly implemented with Keras, among other libraries. It has seen monumental improvements over the last ~5 years, such as Alex Net in 2012, which was the first design to incorporate consecutive convolutional layers.

- Included here: Keras, TensorFlow, and a whole host of others.

DATA STORAGE AND BIG DATA FRAMEWORKS:

Big data is best defined as data that is either literally too large to reside on a single machine, or can't be processed in the absence of a distributed environment. The Python bindings to Apache technologies play heavily here.

- Apache Spark; Apache Hadoop; HDFS; Dask; h5py/pytables.

ODDS AND ENDS:

Includes subtopics such as natural language processing, and image manipulation with libraries such as OpenCV.

- Included here: nltk; Spacy; OpenCV/cv2; scikit-image; Cython.

DATA MUNGING:

Data munging is the general procedure for transforming data from erroneous or unusable forms, into useful and use-case-specific ones. Without some degree of munging, whether performed by automated systems or specialized users, data cannot be ready for any kind of downstream consumption**.**

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. The goal of data wrangling is to assure quality and useful data. Data analysts typically spend the

majority of their time in the process of data wrangling compared to the actual analysis of the data.

The process of data wrangling may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses. Data wrangling typically follows a set of general steps which begin with extracting the data in a raw form from the data source, "munging" the raw data (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use.

STEPS FOR DATA MUNGING:

According to Trifecta, one of the established leaders of the global market for data preparation technology, data wrangling involves mainly six core activities. They are mentioned below.

1. **Discovering:**

   In this process, you understand and learn what is there in your data and to find the best way for some productive analytic explorations.
2. **Structuring:**

   Data is usually in the raw form. While analyzing the data, it needs to make sure that the data is restructured in the way which suits better during the analytical procedures.
3. **Cleaning:**

   Inconsistent and noisy data cannot be used to gain meaningful insights in an organization. The noisy data needs to be cleaned before it is used for analytical approaches.

4. **Enriching:**

   In this process, the cleaned data is enriched by analyzing what new data can be derived from the existed data. This new information is sometimes available in in-house databases, but, and increasingly so, may be sourced from marketplaces for third-party data.
5. **Validating:**

   Validating is the activity that surfaces data quality and consistency issues, or verifies that they have been properly addressed by applied transformations. Validations should be conducted along multiple dimensions.

6. **Publishing**:

Publishing refers to planning for and delivering the output of your data wrangling efforts for downstream project needs (like loading the data in a particular analysis package) or for future project needs (like documenting and archiving transformation logic).

PROBLEMS WITH DATA:

In this blog post, let's discuss some of the most common data quality issues and how we can tackle them.

- Duplicate data. ...
- Inaccurate data. ...
- Ambiguous data. ...
- Hidden data. ...
- Inconsistent data. ... Too much data. ...
- Data Downtime…

REGULAR EXPRESSION:

Regular expression or called fancily Regex is one of the most important topics one needs to be aware of to be a data scientist. The knowledge of Regular expression pays its way while programming. Data scientist use regular expressions in the field of Natural language processing such as text mining, computer vision to extract the part of sentence or strip away the words desired. Regular expression forms the basis of text mining which is now deemed mostly as NLP.

An example of where regular expression is used is in "Data science applied to Ad industry". With each advertisement we see online, there are texts and images embedded with ads. Say we need to optimize our advertisements as advertisers to get maximum click-through rate and conversions. So how do we analyse and optimize? The first strategy is to analyse as which texts impact the most audience and brings most click-through rate, which verbs, and which sentence type a community/region/country responds more. So for performing such analysis and perform the job of data science, the basic understanding of Regular expression is very important in the data science field.