**UNIT II:**

**The Data Science Process: Overview of the Data Science Process-Setting the research goal, Retrieving Data, Data Preparation, Exploration, Modeling, data Presentation and Automation. GettingData in and out of R, Using readr package, Interfaces to the outside world.**

Unit-2

short answer questions

1. Explain about using readr package
2. Explain about interfaces to outside world
3. Explain about data transformation.
4. Explain about data integration

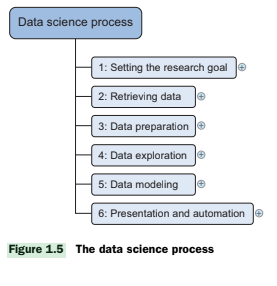
Long answer questions

1. Explain about the data science process
2. What functions can you use to read and write data in R?
3. Explain about exploratory data analysis
4. What is the importance of data preparation?

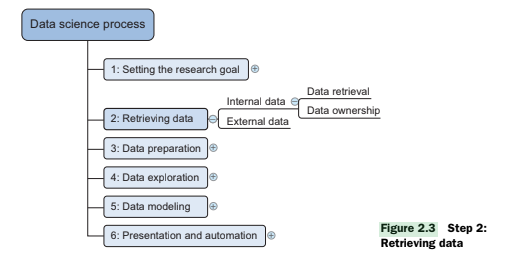
**Q) What are the six steps in the data science process?**

The **data science process** generally involves **six key steps** that help to make sure a data project is successful and provides useful results. These steps are often followed in order, but in reality, you might need to go back to earlier steps if you learn something new along the way.

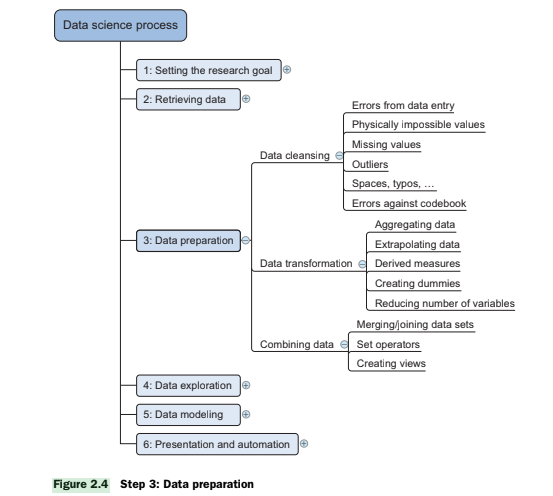
The first step is to **Set the Goal**. This means you need to **decide exactly what you want to find out or achieve with your data project**. It's important to **understand the problem clearly** and make sure everyone involved agrees on what the project is trying to do. You will often create a **project plan** that explains what you will research, why it's important, what data and tools you will need, and when you expect to finish. **Figure 1.5** shows a mind map of the data science process.



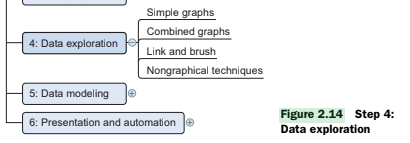
The second step is to **Get the Data**. Once you know what data you need, you have to **find it and get access to it**. This might involve looking at data that your company already has or finding data from other sources. It’s also important to make sure you have **permission to use the data** you find. You will then **collect this raw data** so it can be used in the next steps. **Figure 2.3** shows this step in the process.



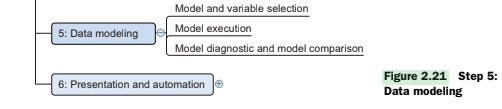
The third step is to **Prepare the Data**. Raw data is often not perfect, so you need to **clean it by fixing mistakes, removing bad values, and handling missing information**. You might also need to **combine data from different places** so you have all the information you need together. Finally, you will **transform the data** into a format that is suitable for the analysis and models you plan to use. **Figure 2.4** shows common actions in this step.



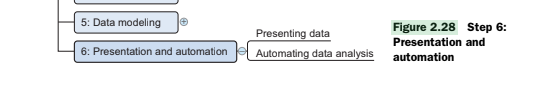
The fourth step is to **Explore the Data**. In this step, you **look closely at the data to understand it better**. You try to **find patterns, relationships, and any unusual things** in the data. Using **graphs, charts, and simple calculations** can help you see these things more clearly and give you **insights** that will be useful for building your models. **Figure 2.14** illustrates this stage.



The fifth step is to **Build Models**. Here, you **create models using techniques like machine learning and statistics** to answer your research questions or make predictions. You might **try different types of models** and **test them** to see which one works best. You will also **refine your models** to make them as accurate as possible. **Figure 2.21** shows the components of model building.



The sixth and final step is to **Present and Automate**. You need to **show your results and findings to the people who need to know**. This might be through reports or presentations. If the process needs to be done again with new data, you might **automate** some or all of the steps to save time. The goal is to make sure your findings are **clear and can be used to make good decisions**. **Figure 2.28** shows this final stage.

****

==================================================================**Q1) Defining Research Goals and Creating a Project Charter**

**Set Clear Goals**:

* Clearly state the purpose of your project.
* Ask questions until you understand the business goals.
* Know how your project will help the business.
* Avoid Misunderstanding:
* Make sure you fully understand the project before starting.

**Start by Understanding the Project**:

* Understand what the project is about (What).
* Understand why the project is important (Why).
* Understand how the project will be done (How).
* Make sure everyone knows what to do.
* Agree on the best way to do it.

**Create a Project Plan (Charter)**:

* Write down the research goal.
* Understand the project's context.

Clients Want to Know:

* Clients need to know what they are paying for.
* Get formal agreement on what you will deliver.

What to Include in a Project Charter:

* Clear research goal.
* Project mission and context.
* Analysis plan.
* Resources needed.
* Proof that the project is achievable.
* List the deliverables (what you will produce) and success measures
* Make a plan with a timeline.

**Importance of People Skills**:

* People skills are more important than technical skills at this stage.
* Senior team members usually guide this part.

**(OR)**

Defining research goals and creating a project charter is the **very first and a very important step** in the data science process. As shown in **Figure**, this step is about **understanding what you want to achieve** with your data project right from the beginning.

Firstly, it's crucial to **set clear goals**. You need to **clearly state the purpose of your project** so that everyone involved knows what you are trying to do. It's important to **ask questions** and keep asking until you **fully understand the business goals** behind the project. You also need to **know how your project will help the business** – what decisions will it inform or what improvements will it lead to? This helps to avoid any misunderstandings later on.

Before you even start looking at the data, you should **start by understanding the project** from all angles. You need to know **what** the project is about, **why** it is important for the business, and **how** you plan to do it. When everyone understands these three things, you can all **agree on the best way to move forward** with the project.

The next important part of this first step is to **create a project plan**, also often called a **project charter**. This is a **written document** that outlines the details of your project. Clients or managers often **want to know what they are paying for**, so getting a **formal agreement** on what you will deliver is essential.

A good **project charter** should include several key things. It should clearly state the **research goal** and explain the **project's context** – why is this project being done now and what background information is important? It should also outline your **plan for analysis**, what **resources** you expect to use (like data, software, or people), and some **proof that the project is achievable**, maybe some initial small tests. Importantly, it needs to list the **deliverables** – what exactly you will produce at the end of the project, such as reports or models – and how you will measure if the project has been a **success**. Finally, the charter should include a **plan with a timeline**, showing when different parts of the project are expected to be completed.

At this early stage, **people skills** are often **more important than strong technical skills**. Being able to communicate clearly, understand the business needs, and build agreement is key. This is why more **senior team members** often guide this part of the data science process. As mentioned in **Section 2.2**, skills in understanding business and working with people are more important here than technical abilities.

### **=========================================================== Step 2: Retrieving Data**

**Finding and Collecting Data**:

* Gather the data you need for your project.
* Check within your company for available data.
* If needed, look outside the company for additional data.

**Sources of Data**:If there is no data available, then, you need to collect new data. This method is called primary data collection.

Another method is to use the data which is readily available. This is called secondary data collection. Different sources in which secondary data is collected is

* Internal sources: Databases, data warehouses, data marts, and data lakes.
* External sources: Public datasets, data from other companies, online sources like Twitter, LinkedIn, etc.

**Challenges in Retrieving Data**:

* Finding all the relevant data can be challenging.
* Gaining access to the data might involve dealing with company policies and permissions.
* Data might be scattered in different places and formats.

**Quality Checks**:

* Check the quality of the data to avoid problems later.
* Make sure the data is accurate and suitable for your analysis.
* Detect and correct any errors early in the process.

=========================================================

Q) **What is Data Cleansing?**

**Data cleansing** is like cleaning up messy information. When you collect data, it can have mistakes, missing parts, or things that don't fit properly. Data cleansing is the job of finding and fixing these problems so that the data is correct and useful. It's a very important step in any data science project.

**Why is Data Cleansing Important?**

* **Better results:** If you use dirty data, the results of your analysis or models won't be good. It's like cooking with bad ingredients – the final dish won't taste right. **Garbage in equals garbage out**.
* **Saves time:** Cleaning data early on can save you a lot of time later when you are trying to understand strange results caused by bad data.

**Steps in Data Cleansing :**

1. **Removing unwanted data:** This means getting rid of information that you don't need or that is just wrong.  
   * **Duplicate values:** Imagine a list of customers where one person's name appears twice with the exact same information. You would remove one of these duplicates.
   * **Irrelevant observations:** If you are looking at data about shop sales in the UK, but one entry is for a sale in France, you would remove that irrelevant entry.
2. **Fixing Structural Errors:** This involves correcting mistakes in how the data is organised.  
   * **Inconsistent data types:** If one column that should contain numbers has some entries as text (e.g., "ten" instead of "10"), you would fix these to be consistent (change "ten" to "10").
   * **Misspelt attributes:** If a column is supposed to be "Colour" but it's sometimes written as "Color", you would make them all the same (e.g., "Colour").
3. **Managing unwanted outliers:** **Outliers** are data points that are very different from the rest of the data. They can sometimes be real but often they are errors.  
   * **Impossible values:** If you have a column for "Age" and someone's age is listed as "300", this is likely an error and you would need to handle it (maybe remove it or correct it if you can find the right age).
   * **Values outside a normal range:** If most people's heights in your data are between 150 cm and 200 cm, but you see one entry of "500cm", this is an outlier that might be a mistake.
4. **Handling missing data:** Sometimes, some pieces of information are missing in your data. You need to decide what to do with these gaps.  
   * **Missing values:** If you have a customer's record but their email address is empty, this is a missing value. You might choose to ignore this record if the email isn't important for your analysis, or you might try to find the email from another source.

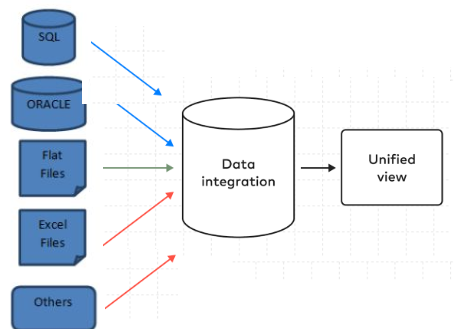
**More Examples of Common Errors and How to Fix Them:**

* **Redundant whitespace:** Imagine a name field with " John Smith ". The extra spaces at the beginning and end can cause problems when comparing names. You would remove these extra spaces so it becomes "John Smith".
* **Capital letter mismatches:** If you have categories like "apple", "Apple", and "APPLE", these should probably be treated as the same. You might convert all of them to lowercase (e.g., "apple") to make them consistent.
* **Different units of measurement:** If you have sales data in both "Pounds" and "Dollars", you would need to convert them to the same unit (e.g., all to Pounds) before you can compare them properly.
* **Deviations from a codebook:** If your data uses codes to represent categories (e.g., "F" for Female, "M" for Male), but you find an entry with "Woman", you would change it to the correct code ("F") based on your codebook or agreed standard.

Data cleansing is a detailed process, but these simple examples should give you a basic idea of what it involves. It's all about making your data clean, consistent, and reliable so you can get useful insights from it.

#### **Q) Introduction to Data Integration**

Data comes from multiple sources such as databases, Excel files, and text documents. Data integration means merging the data that comes from different sources. When all the related information is in one place, it becomes simpler to find patterns, make reports, and make better decisions.While integrating, different Measurement Units must be Converted to a common unit (e.g., liters to gallons).



**Ways to integrate data**

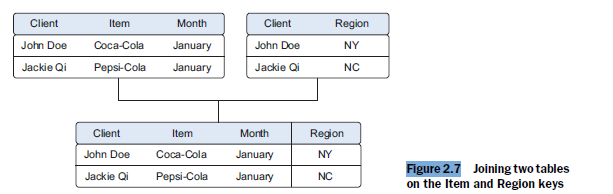
1. Joining (Merging): This is like linking two tables based on some common information.

For example, we have two tables,

One is a purchase table that contains the client, item, month columns

Another table is customer\_region table that contains client and region columns as shown in below figure

When we combine purchase table with customer\_region table based on a common column client, we get a table with columns client, item, month and region.



#### 

#### Appending (Stacking): This is like adding one table on top of another, usually when the have the same columns but different sets of information. For example,

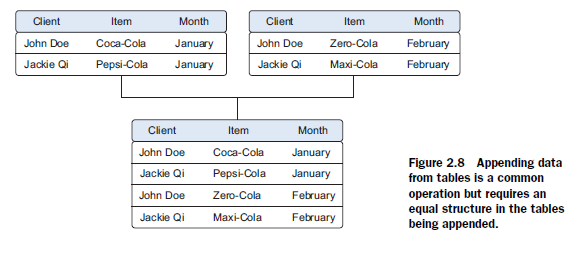
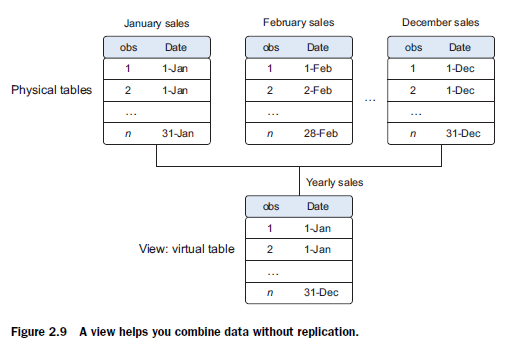


Figure 2.8 shows an example of appending two tables containing sales data for january and february months to create a combined table.

1. **Views:** Views combine data virtually without duplication.

Using Views: Instead of creating new physical tables by joining or appending, you can create a 'view' which is like a virtual table. It combines the data from different sources without actually storing a duplicate copy, saving space. Figure 2.9 shows how monthly sales data can be virtually combined into a yearly sales view.



Different Levels of Aggregation: Align data to the same time period (e.g., daily vs weekly data). Aggregations: Help improve data insights and model performance. Aggregated measures add more insights, like total sales or percentage of sales per region.

Example: Instead of using raw sales numbers, a model might perform better using percentage of total sales as input.

#### **Q) Why Transform Data?**

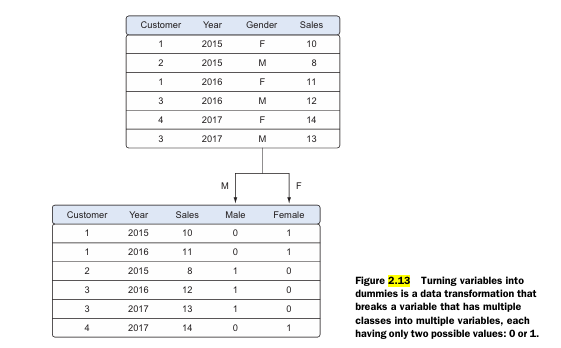
Think of it like preparing ingredients before you can cook a meal – the raw ingredients need to be cleaned, chopped, and sometimes changed in form before they can be used in a recipe.

You need to transform data to make it useful for analysis and building models

Here are the main reasons **why you need to transform data**:

1. **To make the data suitable for your models**

For example, some models work best with numbers, so you might need to convert text data into numerical form



shows an example of turning a variable like "Gender" (with values "F" and "M") into two new variables ("Female" and "Male") with values 0 or 1. This is called **creating dummy variables**

#### **2. To simplify relationships between variables**

Sometimes, the relationship between the output and the input variables is not linear. For example:

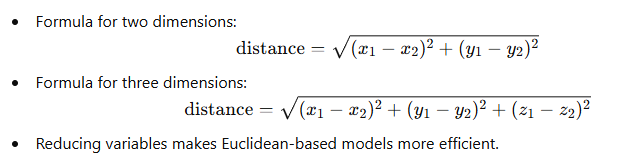
#### 

#### **3. To create new, more useful variables**

You can **combine existing variables** to create new ones that might provide better insights. For example, you might have "sales quantity" and "price" and you can create a new variable "total revenue" by multiplying them .Example creating a new age column from date of birth column

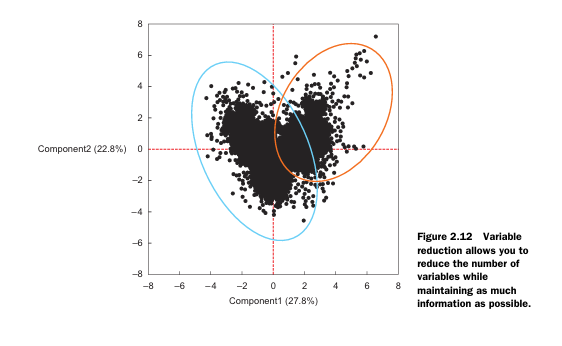
#### **4. Reducing the Number of Variables**

* Too many variables can make models complex and harder to handle.
* Some models (e.g., those using **Euclidean distance**) work well only with up to 10 variables.



Reducing variables makes Euclidean-based models more efficient.Variable reduction keeps important information while removing unnecessary variables.

* Example: **Principal Component Analysis (PCA)** combines multiple variables into fewer **principal components** that explain most of the variation in the data.



=======================================================

#### **Step 4: Exploratory Data Analysis**

In Exploratory Data Analysis (EDA), you will get to know your data

The main goal of EDA is to understand what your data is telling you

1. What kind of information do you have? It involves analyzing data to understand its structure and relationships.
2. If there are any patterns: Are there things that happen regularly?
3. If there are any strange things in your data (outliers): Are there any numbers that are very high or very low compared to the others? These might be mistakes or something interesting.The goal is not to clean data, but errors may be found and corrected.
4. How different pieces of information relate to each other: If one thing goes up, does something else also tend to go up or down? (correlation)

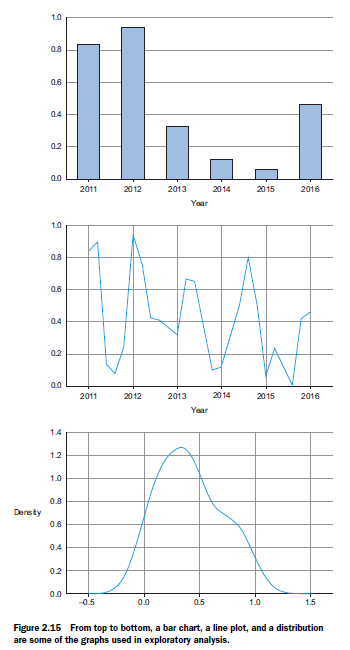
You do EDA by looking at your data in different ways. This can be through:

1. Pictures (visual techniques): Data is easier to understand when shown as graphs.

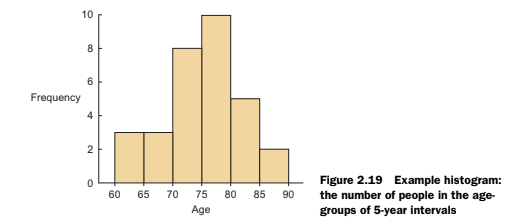
For example, you might draw simple charts like bar charts to see how many sales done per year Or you might use histograms to see how the numbers in your data are spread out

Common visual tools:

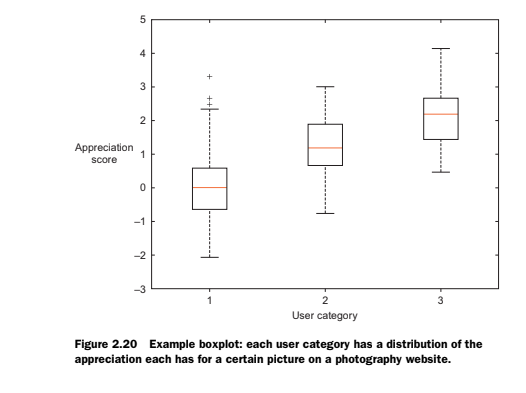
* 1. **Line graphs**
  2. **Bar charts**
  3. **Histograms**
  4. **Boxplots**
  5. **PARETO DIAGRAM**
  6. **LINK AND BRUSH TECHNIQUE**



* **Histogram** (Figure 2.19) Breaks data into **categories** (bins).Shows **how many observations** fall in each category. Example: Age groups of people in 5-year intervals.



* **Boxplot** (Figure 2.20): Shows **minimum, maximum, median**, and distribution of data. Helps **detect outliers**. Example: User appreciation scores for a picture.



Sometimes, single graphs are not enough to see patterns. Multiple graphs can be combined to give better insights.Example: i) Overlaying line charts and bar charts. ii) Creating interactive graphs for dynamic analysis.

1. Non-Graphical Techniques:

i) Simple summaries (descriptive techniques) This might involve calculating simple things like the average value or the range of value

ii) Tabulation: Presenting data in tables.

iii) Clustering: Grouping similar data points together.

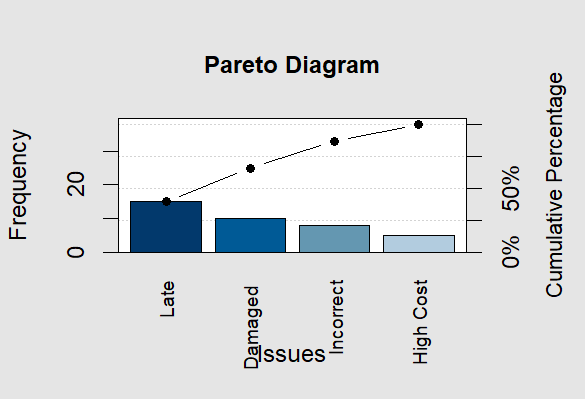
Why is EDA important?

1. It helps you really understand your data before you try to build models or answer big questions . You need to know what you're working with first!
2. The things you learn in EDA can help you decide the best way to build your models. If you see certain patterns, you can choose a model that is good at finding those kinds of patterns
3. Even though the main goal isn't to clean the data, you often find mistakes or strange values (outliers) during EDA. If you find these early, you can go back and fix them before you build your models. This shows that the data science process often goes back and forth

==================================================================

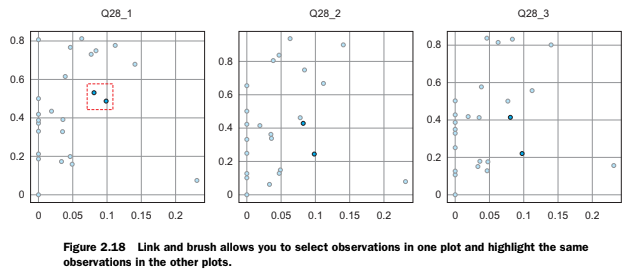
**Q) Pareto Diagram (80-20 Rule)**

* + The Pareto principle (or 80/20 rule) suggests that 80% of the problems usually come from 20% of the causes. A Pareto chart visually shows this, helping you focus on solving those 20% of problems.
  + Think of the Pareto diagram like a super-powered bar chart with an extra line on it. It shows problems in order of importance (from biggest to smallest), and the line helps you see how all the problems add up.
  + Each bar is one type of problem (e.g., "Late Delivery" or "Damaged Goods"). The taller the bar, the bigger that problem is.
  + The line moves across the tops of the bars and shows the *cumulative percentage*. This means it tells you how much each problem, when added together with the others, contributes to the total.
  + Look at the big bars on the left. Those are the biggest problems you should fix first. The line helps you figure out how many problems you'll solve if you focus on those big bars.



**Q) Link and Brush Technique** (Figure 2.18)

* + Allows **selection of data points** in one graph and highlights them in another graph.
  + Useful in **correlation analysis**.



==============================================================

### **Step 5: Building Models**

#### **1. Purpose of Model Building**

* Helps in making predictions, classification, and understanding patterns.
* Uses machine learning, data mining, and statistical techniques.
* Focused process after exploratory data analysis.

#### **2. Steps in Model Building**

1. **Model and Variable Selection**
   * Choose the right variables based on exploratory analysis.
   * Select a suitable model considering:
     + Performance
     + Production feasibility
     + Maintenance difficulty
     + Explanation clarity
2. **Model Execution**
   * Implement the model using programming languages like R, Python.
   * Use lm() function in R, and in python use libraries such as **StatsModels, Scikit-learn** for easier coding.
   * Example: **Linear Regression**

heightx <- c(1,2,3)

weighty <- c(1,3,4)

relation <- lm(weighty~heightx) # Apply the lm() function.

print(relation)

Output:

Coefficients:

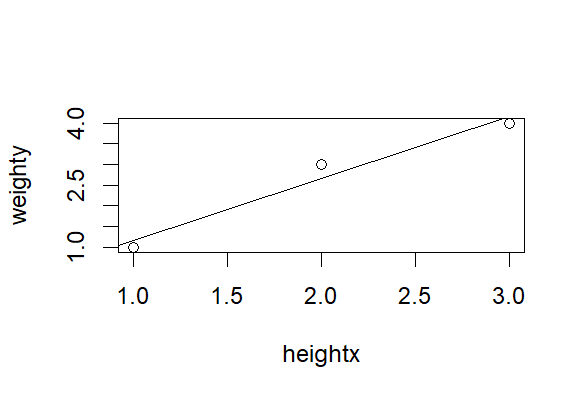
(Intercept) heightx

-0.3333 1.5000

Visualize the Regression Graphically

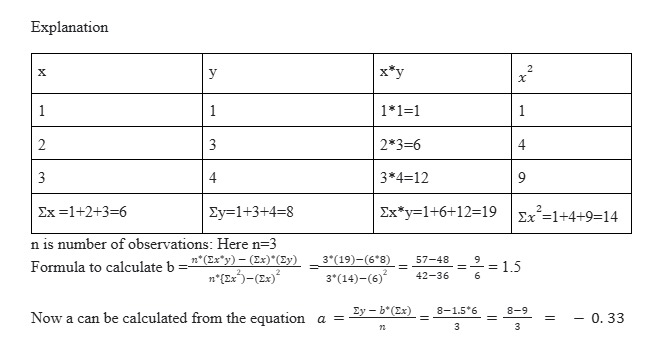
plot(heightx,weighty)

abline(lm(weighty~heightx))



**Key Results in Linear Regression:**

* **R-squared:** Measures model fit (closer to 1 is better).
* **Coefficients:** Show the influence of predictors on the target.
* **p-value:** Shows predictor significance (should be < 0.05).



Example 2: k- Nearest Neighbour Classification (kNN) Technique

# Install and load the required package

install.packages("class")

library(class)

# Define the dataset

data <- data.frame(x1 = c(1, 2, 4, 5),

x2 = c(1, 1, 3, 4),

class = c(0, 0, 1, 1))

# Define the new point to classify

new\_point <- data.frame(x1 = 3, x2 = 2)

# Apply k-NN

k <- 3

predicted\_class <- knn(train = data[, 1:2],

test = new\_point,

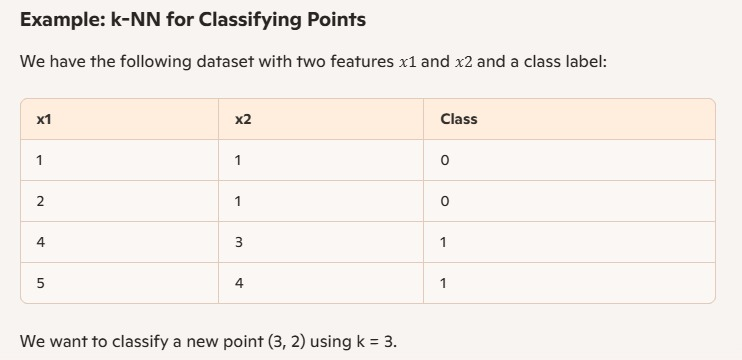
cl = data$class,

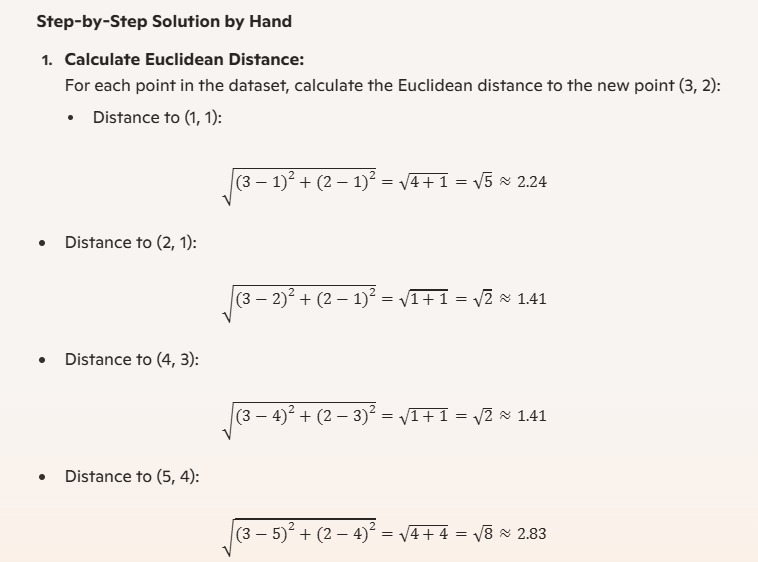
k = k)

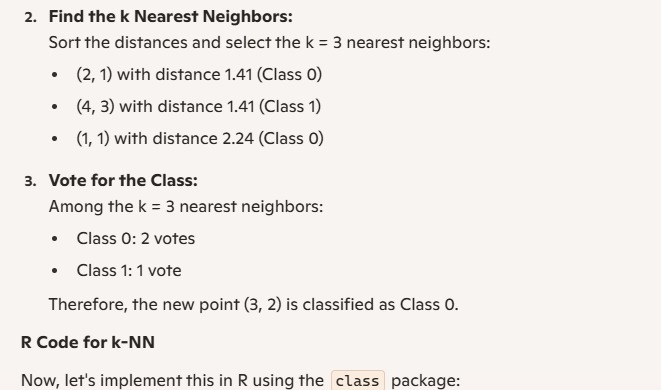
# Print the predicted class

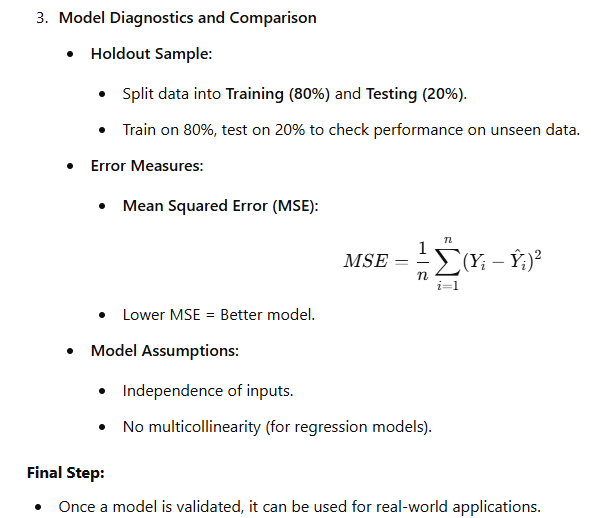
print(predicted\_class)

Output:  
[1] 0









### **============================================================**

### **Step 6:Presenting Findings and Building Applications**

1. **Presentation Matters** – Effectively communicating findings through reports, dashboards, or presentations is crucial.
2. **Automation is Key** – Instead of manually repeating analyses, automate model scoring, report updates, or even build applications for ongoing insights.
3. **Soft Skills are Essential** – Persuading stakeholders and ensuring your work has real-world impact requires storytelling and communication skills.
4. **Different Levels of Automation** – Whether it’s automated reports in Excel/PowerPoint or full-fledged applications, the goal is to make insights easily accessible and continuously updated.

### **Q) What functions can you use to read and write data in R?**

**Reading Data:** Here are some functions to read data into R:

* read.table and read.csv: For reading tabular data like CSV files.
* readLines: For reading lines from a text file.
* source: For reading R code files.
* dget: For reading R code files.
* load: For loading saved workspaces.
* unserialize: For reading single R objects in binary form.

**Writing Data:** Here are some functions to write data from R:

* write.table: For writing tabular data to text files.
* writeLines: For writing lines of text to a file.
* dump: For saving multiple R objects as text.
* dput: For saving an R object as text.
* save: For saving multiple R objects in binary format.
* serialize: For saving an R object in binary format.

**Reading Data with** read.table**:** The read.table() function is commonly used to read data into R. Some important arguments are:

* file: The name of the file.
* header: Whether the file has a header line.
* sep: The separator used in the file (e.g., commas for CSV).
* colClasses: The class of each column.
* nrows: The number of rows to read.
* comment.char: The character indicating comments in the file.
* skip: Number of lines to skip.
* stringsAsFactors: Whether to convert strings to factors.

Example:

data <- read.table("foo.txt")

R automatically handles:

* Skipping lines starting with #.
* Determining the number of rows.
* Identifying the type of each column.

**Reading Large Datasets:** For large datasets, consider the following:

* Read the help page for read.table.
* Estimate the memory required.
* Set comment.char to "" if there are no comments.
* Use colClasses for faster reading.
* Set nrows to help manage memory.

**Memory Calculation Example:** For a data frame with 1,500,000 rows and 120 columns of numeric data, calculate memory needs:

* Numeric data uses 8 bytes per value.
* Multiply rows, columns, and bytes to get total memory required.

**Q) Interfaces to outside world**

Connections let R talk to various external resources, such as databases, text files, or web services.

Common Connection Types

1. file: Opens a connection to a regular file.

2. gzfile: Opens a connection to a file compressed with gzip.

3. bzfile: Opens a connection to a file compressed with bzip2.

4. url: Opens a connection to a web page.

File Connections

You can create a connection to a text file using the file() function.

con <- file("example.txt")

Opening a Connection

You need to open the connection before using it. You can specify the mode in which to open the file:

"r": Read-only mode

"w": Write mode (creates a new file)

"a": Append mode (adds to an existing file)

open(con, "r")

Reading from a Connection

You can then read from the connection, for example using read.csv():

data <- read.csv(con)

Closing a Connection

Always remember to close the connection when you are done:

close(con)

Reading Lines from a file compressed with gzip

You can read a text file line by line using the readLines() function.

con <- gzfile("example.gz")

print(readLines(con, 10))

close(con)

Reading from a URL

You can also read data from web pages using the url() function.

con <- url("http://www.google.com", "r")

print(readLines(con)) # Print the first few lines

close(con)

**Q) Using the readr Package in R**

The **readr package in R is very helpful for working with data files**, especially when these files are **large**. It provides tools that are **faster and more efficient** than the basic R functions for reading data.

**Handling Large Datasets with readr**:

When you have **very big data files**, the readr package can make reading them into R much easier:

* **Speed**: The functions in readr, like **read\_table() and read\_csv()**, are generally **faster** at reading data compared to the standard R functions like read.table(). This can save you a lot of time when dealing with large files.

Example:

library(readr)  
data =read\_csv('C:/Users/HP/Downloads/customers-100.zip')  
str(data)

* **Progress Meter**: readr often shows a **progress bar** while it is reading your file. This helps you see how much of the file has been processed and gives you an idea of how long it will take to finish reading the entire dataset.
* **Memory**: **R loads data into memory**, so the size of your dataset will still be limited by your computer's RAM. For very large datasets with read.table(), it's important to estimate memory requirements. This principle would also apply to readr.

Maximum Rows (n\_max):

data <- read\_csv("data/2016-07-19.csv.bz2", n\_max = 10)

The n\_max = 10 argument tells read\_csv() to read only the first 10 rows of the CSV file.

This is useful when you want to preview a small portion of the data without loading the entire file into memory.

* **Automatic Handling of Compressed Files**: If your large data files are **compressed** (for example, as .gz or .bz2 files), readr can usually **read them directly** without you having to uncompress them first. This saves you an extra step and disk space.

Example:

logdates <- read\_csv("data/2016-07-19.csv.bz2",

col\_types = cols\_only(date = col\_date()),

n\_max = 10)

col\_types = cols\_only(date = col\_date()) means that only the date column is read from the CSV file, and it should be parsed as a date.

**Specifying Column Types with col\_types**:

When you read data with readr, especially **large datasets**, it's a good idea to tell R what **type of data** is in each column. You can do this using the **col\_types argument** in functions like read\_csv().

Here's how it works:

* You use the col\_types argument and give it a **string** that describes the type of each column.
* For example, "cc" means the first two columns are **character (text)** data.
* "i" stands for **integer** (whole numbers), and "n" stands for **numeric** (numbers with decimal points). You can see more options in the readr documentation.

Here is an example:

library(readr)

data <- read\_csv("large\_data.csv", col\_types = "icn")

In this code, we are telling read\_csv() that:

* The first column ("i") contains **integers**.
* The second column ("c") contains **character (text)** data.
* The third column ("n") contains **numeric** data.

**Why Specify col\_types?**

* **Speed**: Telling readr the column types can make it **read your data faster**, especially for large files, because R doesn't have to spend time guessing the types itself.

data <- read\_csv("data/team\_standings.csv", col\_types = "cc")

Here, col\_types = "cc" specifies that both columns in the CSV file should be read as character vectors.

This ensures that the data in those columns is treated as text, regardless of the original data type in the file.

* **Accuracy**: By specifying the types, you can make sure that R **interprets your data correctly**. Sometimes R's automatic guessing might not be what you want, especially if a column has missing values or a mix of data.
* **Safety Check**: If the structure of your data file changes unexpectedly (for example, a column that used to have numbers now has text), specifying col\_types can help you **find these problems early** because readr will give you a warning if the data doesn't match the types you told it to expect.