Islamic University of Technology (IUT)

Organization of Islamic Cooperation (OIC)

Department of Electrical and Electronic Engineering (EEE)

COURSE NO	**:**	**EEE 4416**
LAB NO	**:**	**08 (Part B)**
TOPIC	**:**	**DATA HANDLING USING MATLAB**

DATA HANDLING USING MATLAB

⇨ In this course, we are mostly interested in structured (tabular) data. In later courses, you will learn how to work with other unstructured data.

⇨ In this lab, you will learn the following things.

- How to import data into MATLAB?
- How to export data from MATLAB.
- Data exploration
- Data cleaning
- Data manipulation
- Data analysis
- Data visualization (will be covered in lab – 09)

Data Importing

There are two different ways that you can use to import data into MATLAB.

I. You can use the ‘Import Data’ section from the ‘Home’ tab to directly import data.

II. You can use different built-in commands to load different types of data.

MATLAB supports importing data from various formats:

- **Text files** (.txt, .csv) → readtable, readmatrix, importdata, textscan, fileread
- **Excel files** (.xlsx) → readtable, xlsread
- **MAT-files** (.mat) → load
- **Images** → imread
- **Audio** → audioread
- **Web data, JSON, XML** → webread, jsondecode, xmlread

Use the above commands to read different data. Make sure the data files are stored in the **same directory**.

Data Exporting

- **Save to text:** writetable, writematrix
- **Save to Excel:** writetable, xlswrite
- **Save to .mat:** save
- **Export figures:** saveas, exportgraphics

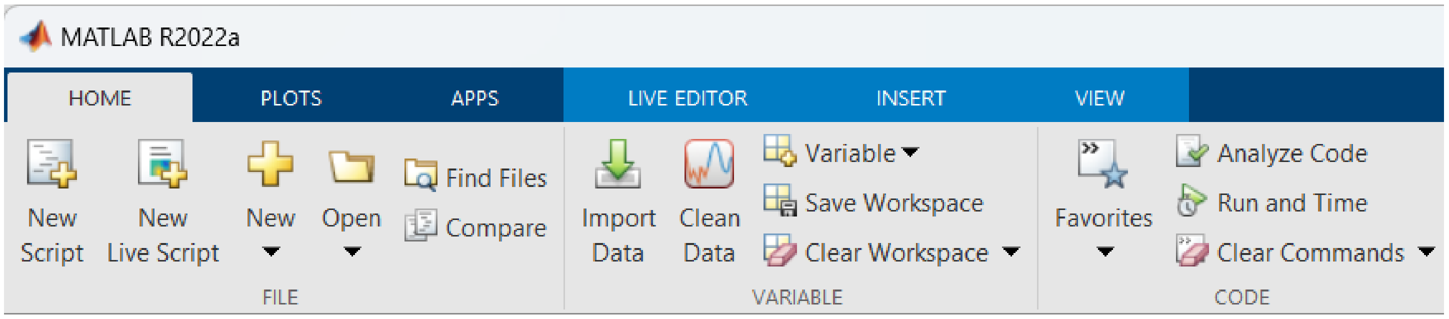
Image Data

```
im= imread('tsushima.png')    % reads the image file into MATLAB; saved in variable im

imshow(im)                    % displays the image
```

- ⇨ The '**imread**' function allows you to read any image file into the workspace.
- ⇨ You need to include the extension (e.g., .jpeg, .png) with the filename.
- ⇨ The resulting data is a 3D array of size 1080 × 1920 × 3, indicating a color image (also known as an **RGB image**) with three channels: **Red**, **Green**, and **Blue**. The third dimension corresponds to these color channels.
- ⇨ Each pixel value ranges from **0 to 255**, as the image is stored in **uint8** (8-bit unsigned integer) format. This allows **256 intensity levels** per channel, representing different shades of gray. This concept is often referred to as "**256 shades of white**."
- ⇨ The combination of values across the three channels at each pixel location determines the final color we perceive. With 256 possible values in each of the three channels, this allows for the representation of **over 16 million** distinct colors.

Excel File

You can read an Excel file into MATLAB using the '**readtable**' function or directly using the '**import data**' tab.

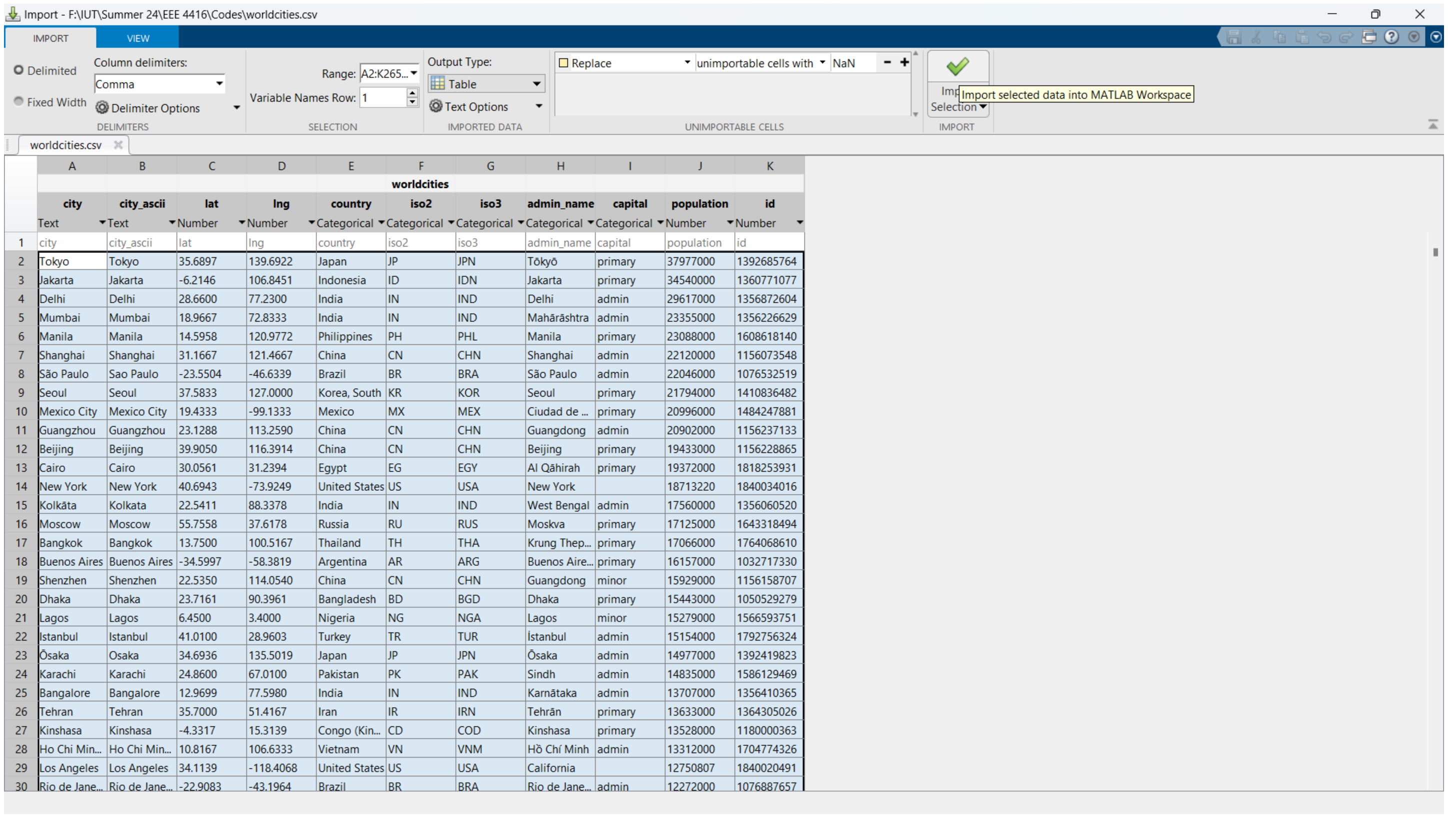


- ⇨ If you click on the import data tab, it will open a new window. You can choose your file if it is kept in that directory or you can go to another location and import the file from there. A new window with the data file will open (image shown below). Click on 'import selection' to import the file into your workspace.

- ⇨ Another approach is to use the 'readtable' function. You need to include the extension (.csv, .xlsx). Avoid using other functions like 'xlsread' or 'readmatrix'.

```
w= readtable('worldcities.csv')
```

  - ○ If you check the data, it is a 26562 x 11 **table**. This means there are 26,562 samples (cities) with 11 columns (also called attributes/features).
  - ○ The data contains both strings and numeric values. So, a table is a **heterogeneous data type**. We will learn more about this new data type in today's session.


Import - F:\IUT\Summer 24\EEE 4416\Codes\worldcities.csv

Delimited · Column delimiters: Comma · Range: A2:K265... · Variable Names Row: 1 · Output Type: Table · Replace unimportable cells with NaN

Import selected data into MATLAB Workspace

worldcities.csv

	A	B	C	D	E	F	G	H	I	J	K
	worldcities										
	city	city_ascii	lat	lng	country	iso2	iso3	admin_name	capital	population	id
	Text	Text	Number	Number	Categorical	Categorical	Categorical	Categorical	Categorical	Number	Number
1	city	city_ascii	lat	lng	country	iso2	iso3	admin_name	capital	population	id
2	Tokyo	Tokyo	35.6897	139.6922	Japan	JP	JPN	Tōkyō	primary	37977000	1392685764
3	Jakarta	Jakarta	-6.2146	106.8451	Indonesia	ID	IDN	Jakarta	primary	34540000	1360771077
4	Delhi	Delhi	28.6600	77.2300	India	IN	IND	Delhi	admin	29617000	1356872604
5	Mumbai	Mumbai	18.9667	72.8333	India	IN	IND	Mahārāshtra	admin	23355000	1356226629
6	Manila	Manila	14.5958	120.9772	Philippines	PH	PHL	Manila	primary	23088000	1608618140
7	Shanghai	Shanghai	31.1667	121.4667	China	CN	CHN	Shanghai	admin	22120000	1156073548
8	São Paulo	Sao Paulo	-23.5504	-46.6339	Brazil	BR	BRA	São Paulo	admin	22046000	1076532519
9	Seoul	Seoul	37.5833	127.0000	Korea, South	KR	KOR	Seoul	primary	21794000	1410836482
10	Mexico City	Mexico City	19.4333	-99.1333	Mexico	MX	MEX	Ciudad de ...	primary	20996000	1484247881
11	Guangzhou	Guangzhou	23.1288	113.2590	China	CN	CHN	Guangdong	admin	20902000	1156237133
12	Beijing	Beijing	39.9050	116.3914	China	CN	CHN	Beijing	primary	19433000	1156228865
13	Cairo	Cairo	30.0561	31.2394	Egypt	EG	EGY	Al Qāhirah	primary	19372000	1818253931
14	New York	New York	40.6943	-73.9249	United States	US	USA	New York		18713220	1840034016
15	Kolkāta	Kolkata	22.5411	88.3378	India	IN	IND	West Bengal	admin	17560000	1356060520
16	Moscow	Moscow	55.7558	37.6178	Russia	RU	RUS	Moskva	primary	17125000	1643318494
17	Bangkok	Bangkok	13.7500	100.5167	Thailand	TH	THA	Krung Thep...	primary	17066000	1764068610
18	Buenos Aires	Buenos Aires	-34.5997	-58.3819	Argentina	AR	ARG	Buenos Aire...	primary	16157000	1032717330
19	Shenzhen	Shenzhen	22.5350	114.0540	China	CN	CHN	Guangdong	minor	15929000	1156158707
20	Dhaka	Dhaka	23.7161	90.3961	Bangladesh	BD	BGD	Dhaka	primary	15443000	1050529279
21	Lagos	Lagos	6.4500	3.4000	Nigeria	NG	NGA	Lagos	minor	15279000	1566593751
22	Istanbul	Istanbul	41.0100	28.9603	Turkey	TR	TUR	İstanbul	admin	15154000	1792756324
23	Ōsaka	Osaka	34.6936	135.5019	Japan	JP	JPN	Ōsaka	admin	14977000	1392419823
24	Karachi	Karachi	24.8600	67.0100	Pakistan	PK	PAK	Sindh	admin	14835000	1586129469
25	Bangalore	Bangalore	12.9699	77.5980	India	IN	IND	Karnātaka	admin	13707000	1356410365
26	Tehran	Tehran	35.7000	51.4167	Iran	IR	IRN	Tehrān	primary	13633000	1364305026
27	Kinshasa	Kinshasa	-4.3317	15.3139	Congo (Kin...	CD	COD	Kinshasa	primary	13528000	1180000363
28	Ho Chi Min...	Ho Chi Min...	10.8167	106.6333	Vietnam	VN	VNM	Hồ Chí Minh	admin	13312000	1704774326
29	Los Angeles	Los Angeles	34.1139	-118.4068	United States	US	USA	California		12750807	1840020491
30	Rio de Jane...	Rio de Jane...	-22.9083	-43.1964	Brazil	BR	BRA	Rio de Jane...	admin	12272000	1076887657

- You can see from the above image that some of the columns like 'id' or 'lat' are of numeric data type. While other columns like 'country' are of '**Categorical**' data type.

Text files

You can read a .txt file using several functions depending on the file's structure and your requirements. Not every method is suitable for all files.

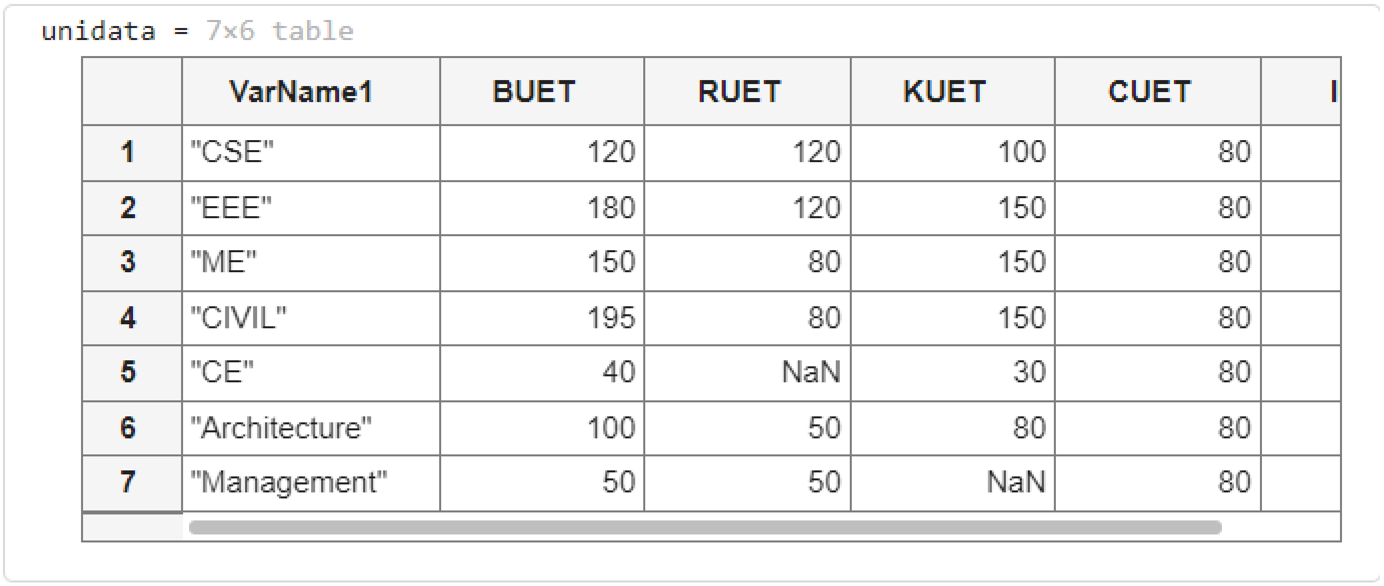
I. If your file contains numeric data arranged in rows and columns, use '**readmatrix**'.
II. Use '**fileread**' for text files where you need the full contents as a single string. It will read the entire file as a string.
III. You can also use the '**import data'** tab to directly import into MATLAB. This will import the data using a table or array format.

There are two .txt files provided - 'Uni data.txt' and 'high impact conference.txt'. Both are of different types. The first one is similar to a .csv file with 'space' as a delimiter. The second one is a simple string.

Using all 3 methods mentioned above is not suitable – check the image below. It's for the first data.

- The first output is using the **import data'** tab.
- The second output is from using the 'fileread' function – this reads the entire data as a single string which does not serve the purpose of this data.
- The third output is from using the 'readmatrix' function – this reads the data only as numeric values. The strings are termed as NaN.

Evidently, the first one is most suitable for this data. Find out which one will be more suitable for the other file.



unidata = 7x6 table

	VarName1	BUET	RUET	KUET	CUET	I
1	"CSE"	120	120	100	80	
2	"EEE"	180	120	150	80	
3	"ME"	150	80	150	80	
4	"CIVIL"	195	80	150	80	
5	"CE"	40	NaN	30	80	
6	"Architecture"	100	50	80	80	
7	"Management"	50	50	NaN	80	

```
t =
     '   BUET     RUET     KUET     CUET     IUT
      CSE         120      120      100      80       40
      EEE         180      120      150      80       80
      ME 150      80       150      80       55
      CIVIL       195      80       150      80       45
      CE 40       nan      30       80       nan
      Architecture         100      50       80       80       nan
      Management 50        50       nan      80       30
      '
```

```
ta = 7x6
     NaN   120   120   100    80    40
     NaN   180   120   150    80    80
     NaN   150    80   150    80    55
     NaN   195    80   150    80    45
     NaN    40   NaN    30    80   NaN
     NaN   100    50    80    80   NaN
     NaN    50    50   NaN    80    30
```

Mat files

.mat files are MATLAB's proprietary binary format used to store variables, arrays, and other data. They allow you to save and load your workspace variables efficiently between sessions or across programs. There are some sample .mat files already provided by MATLAB.

To read .mat files in MATLAB, you use the *load* function. This loads all variables stored in a .mat file into the workspace. You don't need to assign a separate variable.

```
load fatalities.mat
```

This will load a 49x8-sized table into the workspace under the variable name 'fatalities'.

Data Analysis/Processing

Data exploration, data cleaning, and data manipulation – all these can be considered part of data processing tasks. Let's look at some simple data processing tasks.

Image Data

Image processing is a field in Electrical and Computer engineering that focuses on performing operations on images to analyze, enhance, or extract useful information from them. 'Digital Image Processing' is an elective course offered by the EEE department to 4th year students. So, we will not explore much of this task in this course. If you are interested, you can look into the following GitHub repositories or YouTube to get some introduction (these will not be included in the assessment of this course).

- https://github.com/newaz-aa/Digital-Image-Processing
- https://github.com/newaz-aa/Biomedical_Image_Processing

Image processing includes tasks like removing noise, image enhancement, segmentation, feature extraction, object recognition, etc.

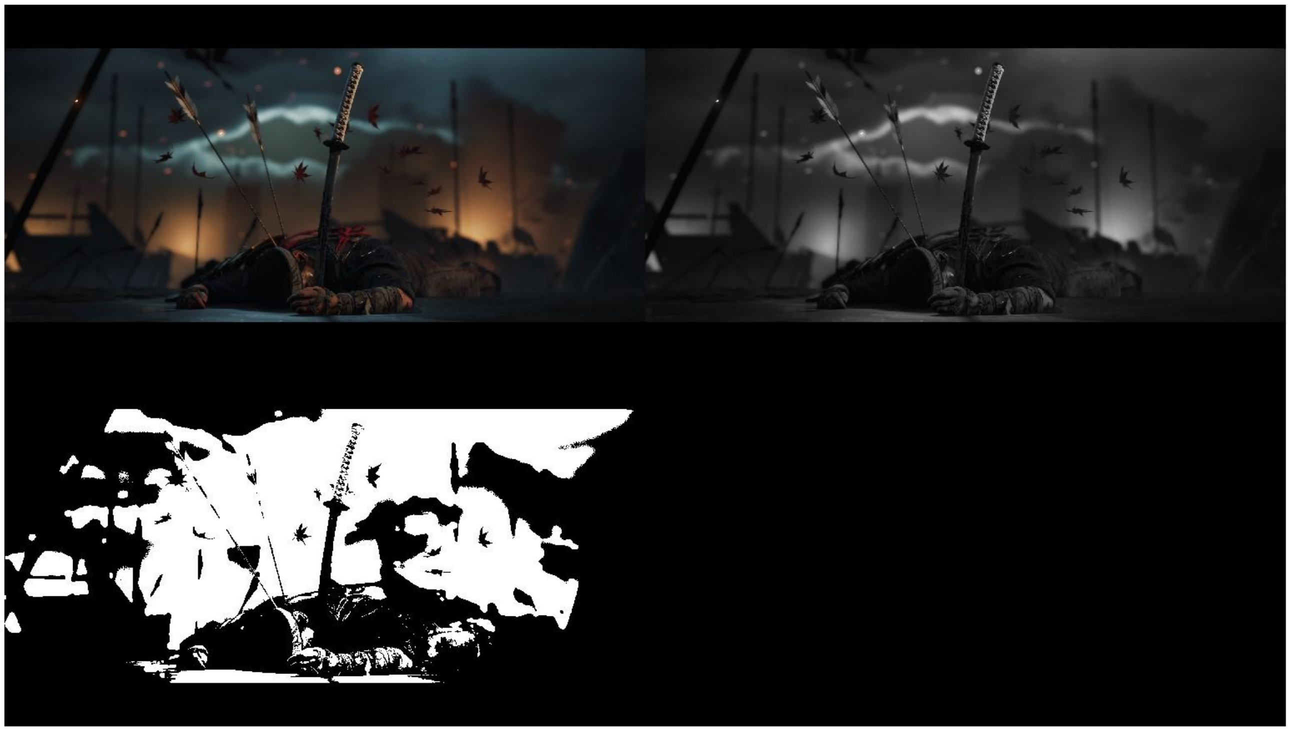
Some common operations on images include –

Operation	Example
Grayscale conversion	RGB → black & white
Filtering	Blur, sharpen, denoise
Edge detection	Detect object boundaries
Morphological ops	Erosion, dilation, thinning
Image segmentation	Separate foreground/background
Object recognition	Identify faces, digits, etc.
Compression	JPEG, PNG, etc.

Let's try one of them. We have imported an RGB image into MATLAB. Let's convert it into a grayscale and a binary image.

The grayscale image has the same width and height as the RGB image but with only 1 channel (values ranging from 0 to 255). The binary image also has the same width and height but with only logical values 0 and 1 – representing black and white color, respectively.

```
im_gray= rgb2gray(im)
imshow(im_gray)


im_bin= imbinarize(im_gray)
imshow(im_bin)

montage({im, im_gray, im_bin})
```



Applications of Image Processing:

- **Medical imaging**: MRI, CT scan analysis
- **Computer vision**: self-driving cars, facial recognition
- **Remote sensing**: satellite image analysis
- **Document analysis**: OCR: Optical Character Recognition
- **Security and surveillance:** Number plate recognition
- **Industrial automation**: defect detection, barcode reading

Tools for Image Processing:

- **MATLAB**: it provides an Image Processing Toolbox to perform different tasks. You can also find different built-in apps for 'image processing and computer vision' in the 'APPS' tab. These apps make different image-processing tasks much easier.
- **Python**: libraries like OpenCV, PIL (pillow), scikit-image, Pytesseract
- **C++/OpenCV**
- **TensorFlow/PyTorch**: for deep learning-based tasks

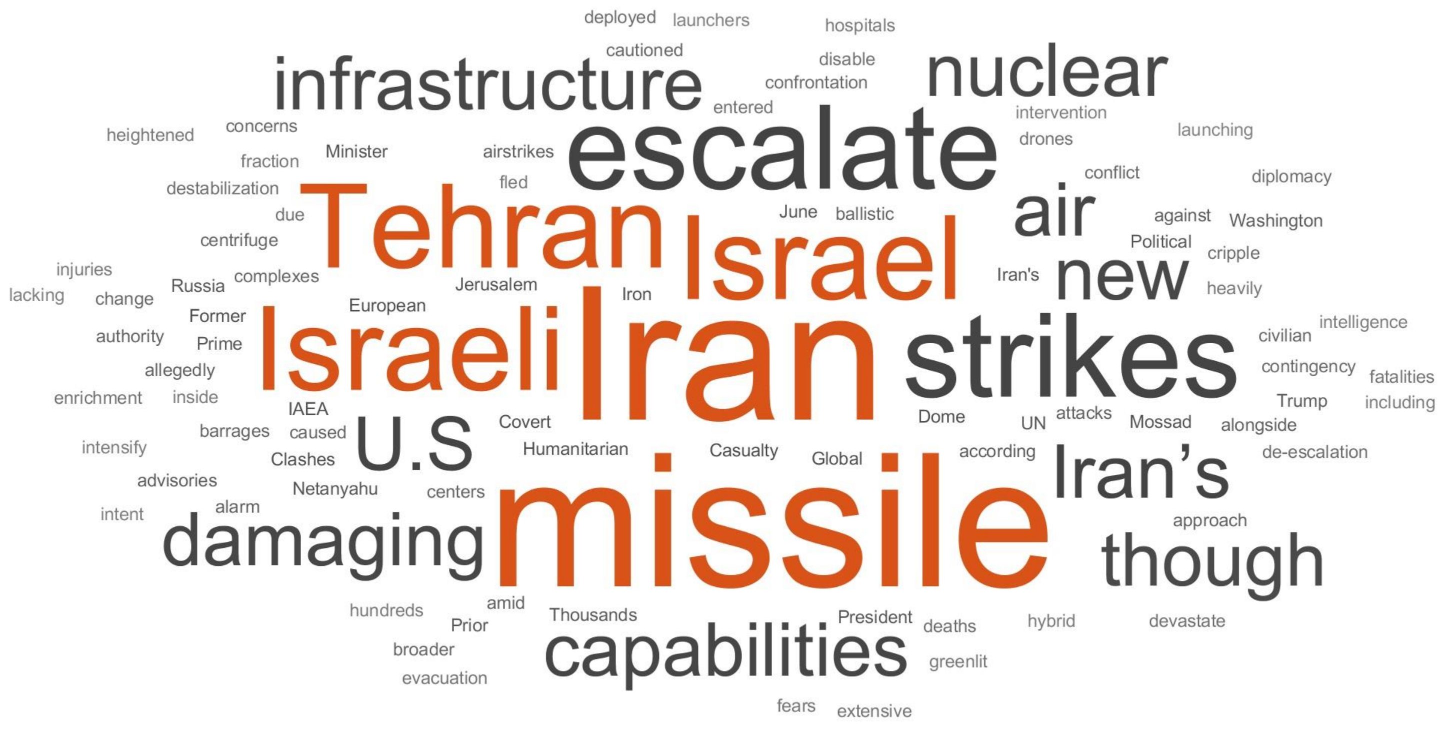
Text Data

Text data processing refers to the methods and techniques used to clean, transform, and analyze text data so it can be used in applications like natural language processing (NLP), sentiment analysis, and information retrieval. It primarily falls under the domain of Computer Science, so we will not explore this segment in depth.

MATLAB provides simple tools to handle text. For example, the 'fileread' function allows you to import text data directly into MATLAB as a string (character array). Once imported, you can perform various processing tasks.

The following example demonstrates how to create a **word cloud** from a text passage. A word cloud visually highlights the **most frequent words**, helping identify key terms in the passage.

```
war= fileread('War_2025.txt')

wordcloud(war)
```



Applications of Text Processing

- Spam detection
- Sentiment analysis
- Chatbots / virtual assistants
- Machine translation
- Document classification
- Search engines

Asif Newaz
Lecturer, EEE, IUT

Tabular Data

Tabular data is data organized in rows and columns, similar to a spreadsheet or a database table. Each row represents an observation or record (also called 'sample'), and each column represents a variable or feature.

ID	Name	Age	Country	Score
1	Alice	23	USA	87.5
2	Bob	30	Canada	92.0
3	Charlie	27	Bangladesh	78.2

MATLAB uses the '**table**' data format to represent tabular data. Other programming languages like Python have libraries such as **Pandas** to work with tabular data.

You have already seen how to import tabular data into your workspace from .csv, .mat, and .txt files. Let's try and understand the 'table' data type to learn how to work with such datasets.

Table Data Structure

It is a data container that allows you to store columns of data of different types (e.g., numbers, strings, dates) all in one variable—perfect for structured data. It is similar to a spreadsheet or database table.

Key Characteristics of a table:

- Each column can have a different data type.
- Columns have names.
- Rows can have names **(optional).**
- You can access columns like fields in a structure.

Creating a Table

Suppose the following table contains the data of some students in IUT. How can we represent this data properly? The data contains numbers as well as strings. So, we need a heterogeneous data type.

All the students here have 4 **attributes** (ID, Gender, CGPA, Dep). There are 5 students in total – so, 5 rows.

Name	**ID**	**Gender**	**CGPA**	**Department**
Ben Affleck	112	Male	3.7	EEE
Henry Cavil	170	Male	3.8	ME
Zack Snyder	214	Male	3.9	CSE
Hermione	120	Female	3.85	EEE
Lucy Pavensie	220	Female	3.65	CE

Let's try to create this table in MATLAB.

I. Start by creating several **column vectors** that contain your data. Each vector will represent a column in the table. These columns can contain different data types — like numbers, text, or logical values. For the above data, we need 5 such column vectors.

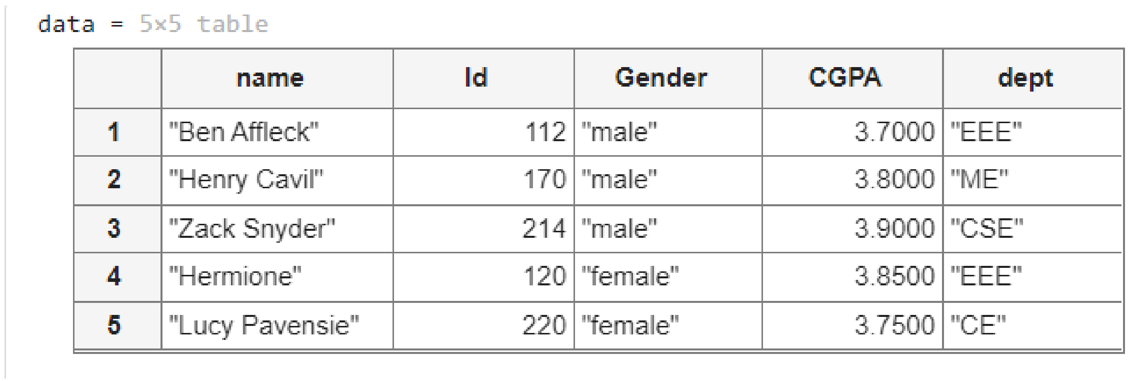
```
name= ["Ben Affleck"; "Henry Cavil"; "Zack Snyder"; "Hermione"; "Lucy Pavensie"]
Id= [112; 170; 214; 120; 220]
Gender = ["male"; "male"; "male"; "female"; "female"]
CGPA = [3.7, 3.8, 3.9, 3.85, 3.75]'
dept= ["EEE", "ME", "CSE", "EEE", "CE"]
```

There is one simple mistake in the above code. Can you figure out what that is?

II. Then, combine these vectors into a single table using the *table ( )* function.

```
Data_v1= table(name, Id, Gender, CGPA, dept)
```

This will create a table of size (5, 5). As you can see, the 'name' variable is considered as an attribute in the following table.

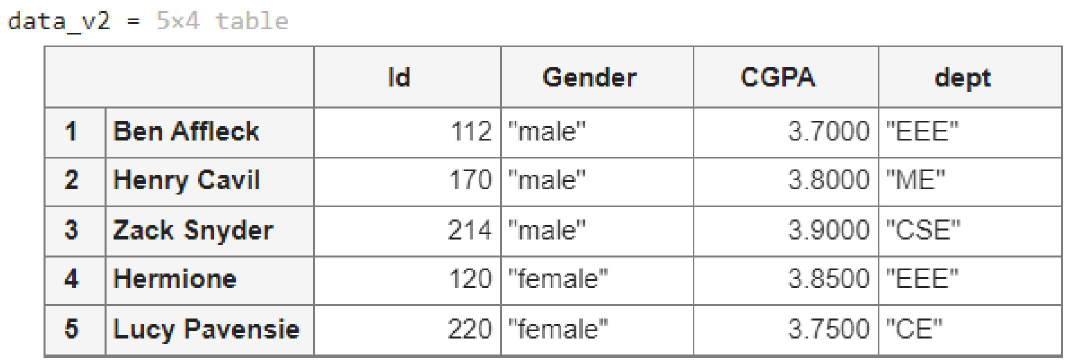


data = 5x5 table

	name	Id	Gender	CGPA	dept
1	"Ben Affleck"	112	"male"	3.7000	"EEE"
2	"Henry Cavil"	170	"male"	3.8000	"ME"
3	"Zack Snyder"	214	"male"	3.9000	"CSE"
4	"Hermione"	120	"female"	3.8500	"EEE"
5	"Lucy Pavensie"	220	"female"	3.7500	"CE"

III. We can consider labeling each row of the table. That way, we can access the rows by that label, similar to column names.
Check the difference between the two tables – one is 5x5, the other is 5x4. Instead of using 'name', we could also have used 'ID' as the label for rownames.

```
data_v2= table(Id, Gender, CGPA, dept, RowNames=name)
```



data_v2 = 5x4 table

		Id	Gender	CGPA	dept
1	Ben Affleck	112	"male"	3.7000	"EEE"
2	Henry Cavil	170	"male"	3.8000	"ME"
3	Zack Snyder	214	"male"	3.9000	"CSE"
4	Hermione	120	"female"	3.8500	"EEE"
5	Lucy Pavensie	220	"female"	3.7500	"CE"

Accessing Data

You can access data from a table in several ways:

- By column name (like accessing fields in a structure).
- By using curly braces {} for numeric indexing to extract **data directly**.
- By using parentheses () to extract a **sub-table**.

```
data_v1.Gender


data_v1{2,:}
data_v1{2:4, "CGPA"}
data_v1{:,[1, 3]}
data_v1{:,[2, 4]}


data_v1(1:3, 5)
data_v2(:, 3:4)


data_v2("Ben Affleck", :)
data_v2{"Hermione", :}
```

Modifying Table

You can update existing data in the table by assigning new values to specific cells or columns. You can also:

- Add a new column by assigning a new vector to a new column name.
- Add a new row by using the same syntax and providing values for all columns.

i. Let's say, you want to change the ID of Ben Affleck to 200.

```
data_v1.Id(1)=200
```

ii. Change the dept of Lucy to ME.

```
data_v1.dept(5)='ME'
```

iii. Change the CGPA of Cavil.

```
data_v2("Henry Cavil","CGPA")= {3.9}
```

In this approach, you need to use { }.

iv. Add a new column age.

```
data_v1.age = [50, 45, 52, 24, 16]'
```

v. Add a new row

```
data_v1(6,:) = {"Triss", 300, "female", 3.99, "EEE", 100}   % hard coded

data_v1(size(data_v1,1),:)={"Geralt", 400, "male", 3.0, "CSE", 150}
```

Exporting Tables

You can save your modified table as a CSV or Txt file using the 'writetable' function. It will be saved in the current directory. Don't forget to add the extension with the filename.

```
writetable(data_v1, 'data_modified.csv')
```

Data Analysis/Processing

You can perform various data processing tasks on the table. Here is a summary of some common functions for that.

Function	**Description**
summary(T)	Summary of variables (useful for large tables)
head(T, n)	First n rows
tail(T, n)	Last n rows
sortrows(T)	Sort table rows
innerjoin / outerjoin	Join tables
removevars	Remove specific variables (columns)
renamevars	Rename columns
rowfun	Apply a function to each row

i. Say, you want to sort your table based on student ID or CGPA. You can use the 'sortrows' function with an optional argument indicating the column based on which you want to sort the rows.

```
sortrows(data_v1,"CGPA")
```

ii. You can remove any column using the 'removevars' function.

```
removevars(data_v1, "dept")
```

iii. You can obtain a summary of the data (max/min/avg value, variable type) using the summary function.

```
summary(data_v1)
```

Categorical Data Structure

MATLAB's categorical data type is used to store discrete, repeated text values more efficiently and meaningfully than plain strings or character arrays. It allows the **grouping** of similar strings.

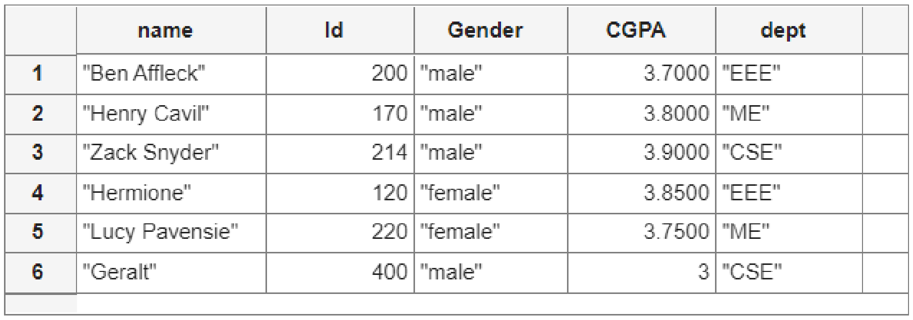
For instance, you have labels "Male" and "Female" in the above table you created. The 'Gender' column is of type string (you can see from the output of the summary function).

Let's change it to a categorical data type and see the difference.

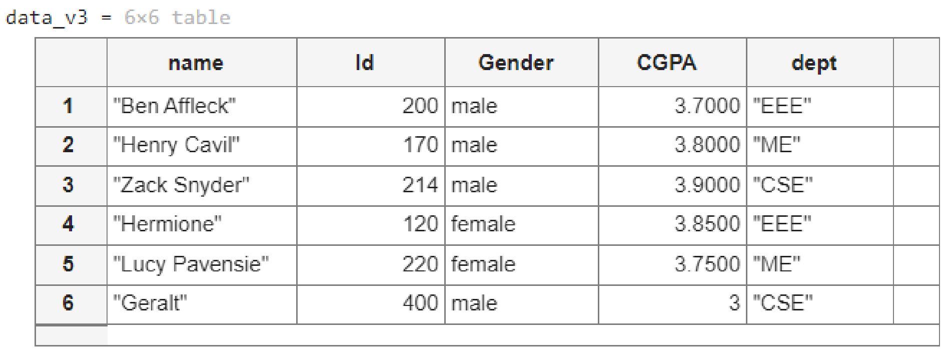
```
data_v3= data_v1

data_v3.Gender = categorical (data_v1.Gender)

summary(data_v3)
```

Can you spot the difference between the summary of data_v1 and data_v3? The group of students has now been classified into two categories: male and female. They are no longer in string format.

This allows for easier handling in many cases. For instance, you can extract or **summarize** similar types of data much more efficiently when the data type for that variable is categorical.



	name	Id	Gender	CGPA	dept
1	"Ben Affleck"	200	"male"	3.7000	"EEE"
2	"Henry Cavil"	170	"male"	3.8000	"ME"
3	"Zack Snyder"	214	"male"	3.9000	"CSE"
4	"Hermione"	120	"female"	3.8500	"EEE"
5	"Lucy Pavensie"	220	"female"	3.7500	"ME"
6	"Geralt"	400	"male"	3	"CSE"



```
data_v3 = 6×6 table
```

	name	Id	Gender	CGPA	dept
1	"Ben Affleck"	200	male	3.7000	"EEE"
2	"Henry Cavil"	170	male	3.8000	"ME"
3	"Zack Snyder"	214	male	3.9000	"CSE"
4	"Hermione"	120	female	3.8500	"EEE"
5	"Lucy Pavensie"	220	female	3.7500	"ME"
6	"Geralt"	400	male	3	"CSE"

The benefit of this will become more apparent when you work with a huge amount of data. If you have a very large data set containing all the students of IUT from the beginning, categorized data can make it very easy to find a particular group, e.g., students of batch 2010 or the students of the BTM department, etc.

- ⇨ Try to plot the gender column of data_v1 and data_v3 using the **plots** tab. See the difference.
- ⇨ If you have millions of repeated strings, converting them to categorical can drastically **reduce memory usage**. This is because categorical data stores only one instance of each category name, reducing memory footprint.

Class Task:

i. Import the ‘Worldcities.csv’ file.
ii. Check the variable data types before importing.
iii. Summarize the table.
iv. Find how many missing entries are there in each column.
v. See the histogram of the country column – which country has more cities reported in the data?
vi. See the word cloud of the country column.
vii. Remove some unnecessary columns such as 6, 7, and 8 from the table (using code, also using **data viewer** window using workspace).
viii. Find which city has the highest population.