

FUNDAMENTAL BUSINESS DATA VISUALIZATION NOTES

Prepared by:
Lester Dann G. Lopez

Based on course material provided through King Mongkut's
University of Technology Thonburi (KMUTT)

All Learning Unit Documents

Course Introduction

Human interprets picture better than words

like plotting chart to look pattern

- Start by Asking Question
- Understand data collection
 - transform data to proper data
 - data cleaning
- 3 Libraries

With data visualization → for users
→ for team

Introduction to Data Science for Business

Stages of Data Science

1. Data Acquisition & Understanding

- involves collecting raw data from various sources like databases, API, web scraping

3. Deployment

- used to make prediction and decision in REAL TIME

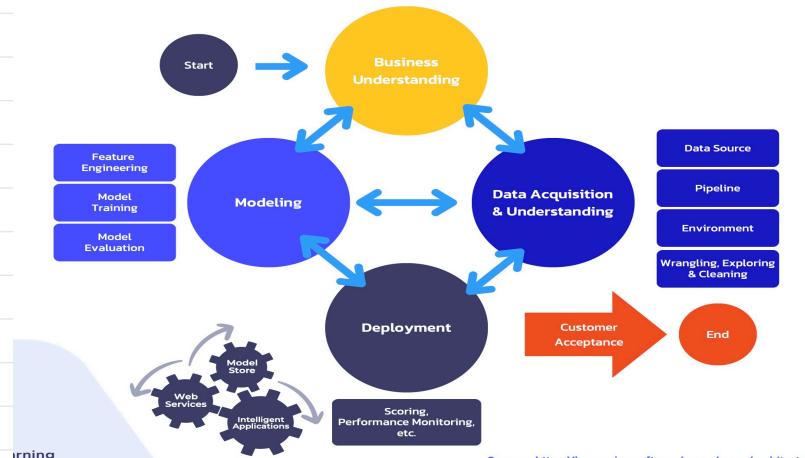
Data Modeling

- ##### 2.
- analyze and process data to create model that can identify pattern & prediction
 - involves STATISTICAL ANALYSIS

4. Customer Acceptance

- presented to stakeholders / end user for feedback

Stages of Data Science



Before data analysis, we need to understand Business

① What problem are we solving?

- ask the right questions
- find the right data

Business Understanding



Define Objectives

- Identify the business problem.
- Ask questions that define business goals achievable with data science.

Examples:

- How much will sales increase? (Regression)
- Which customer will churn? (Classification)
- Which products are similar? (Clustering)
- Is this transaction suspicious? (Anomaly Detection)
- What's the best product recommendation? (Recommendation)

Identify Data Sources

- Identify relevant data to answer your questions.
- Ensure data accurately reflects both the target outcome and related features.

Data Acquisition and Understanding

- prepare data before analyzing

Ingest the data

- get the data into the system

Explore the data

- check quality of data, ensure clean (data cleaning)

Set up the data pipeline

Modeling

Feature engineering

- process in which we transform raw data into features so that the model can understand

Model training

- test model and pick that solves problem accurately

Determine if your model is suitable for production

- test in real time

Deployment

Operationalize the model

transfer model to the special environment

Expose model with an API interface

API allows applications to easily talk to the model
and get predictions

Customer Acceptance

System Validation

Project handoff

Topics

- What is Business Intelligence (BI)?
- Why is Data Visualization so important for BI?
- How can businesses use Data Visualization to make decisions?
- Trends of Data visualization

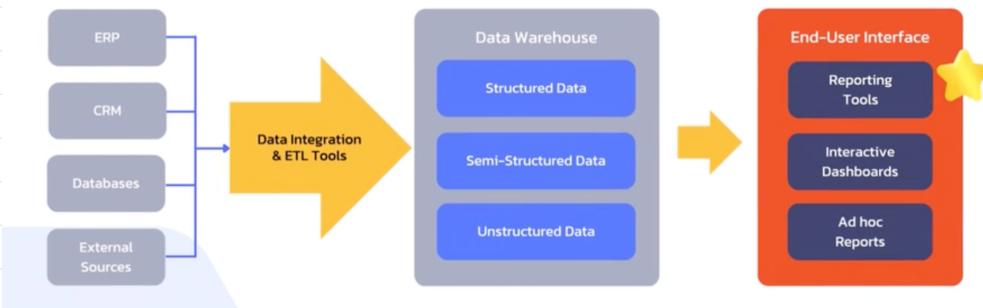
What is Business Intelligence (BI)

- tool to organize, analyze and understand data

BI takes raw numbers from convert to actionable insights



Business Intelligence Architecture (for Small Business)



Why data visualization Important?

- our brain grasp pictures easier than numbers

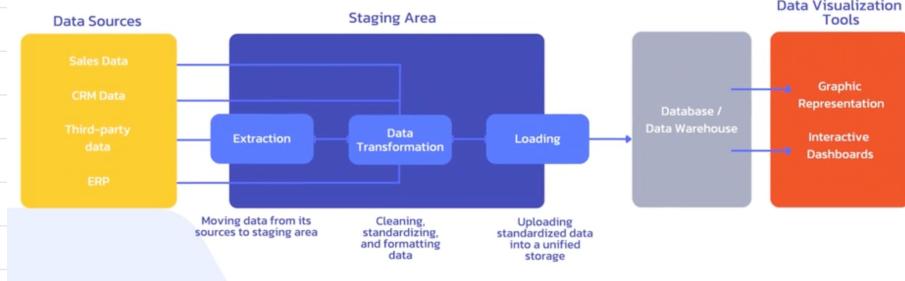
- easily spot pattern

- see the bigger picture

- significant numbers → opportunity / issue



Data Processing and ETL steps



By analyzing patterns and trends in visualized data, business can:

Improve Operational Process

Predict Future Trends



Example Trends of Data visualization



Emergence of High-Fidelity Digital Twins

Virtual model of physical object or system on stream to physical asset



Powerful Javascript Visualizations

Powerful JavaScript frameworks like React, Angular, Babylon.js, and Three.js are making it easier to create stunning 3D and immersive reality (VR) data visualizations.



Verticalized Data Visualization Offerings

Industry-specific insights

ex. Use to monitor performance of Robot

Data Visualization Technique

Topics

- Basic 2D Data Visualization
- Types of Connection
- 8 Common Chart types

Key of Data Visualization

1. Know your data

- what to show and how to use the chart to explain it

2. Types of connection

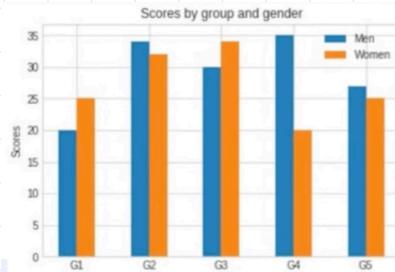
- Comparison
- Composition
- Relationship
- Distribution

8 Common chart Types

1. Bar Chart → Comparison Type

- comparing data units
- can be clustered or stack bars

* Use HORIZONTAL BARCHART for long data label



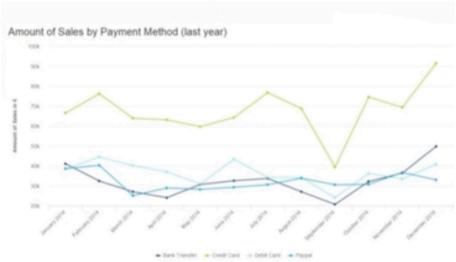
2. Pie chart → Composition

- composition of whole, percentage



3. Line Graph → Comparison, relationship

- change data over time
- can combine 1 or more line graph
- Useful for trends, pattern and make prediction



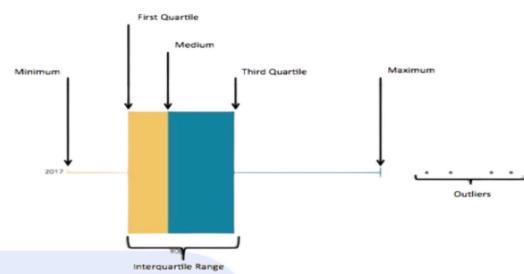
4. Box Plot → Distribution

- makes it easy to see data spread, skewness, variability & outliers

Shows:

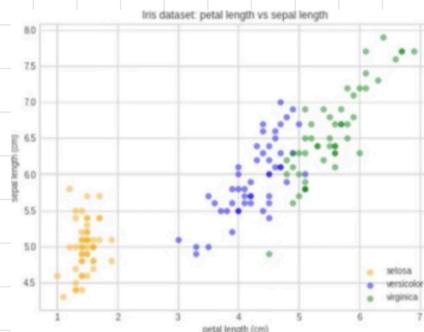
1. Minimum – lowest non-outlier value
2. Q1 (First Quartile) – 25% of data is below this
3. Median (Q2) – the midpoint of the data
4. Q3 (Third Quartile) – 75% of data is below this
5. Maximum – highest non-outlier value
6. Outliers – unusually high or low values (shown as small dots)

Together these form the box (Q1-Q3) and the whiskers (min-max), making it easy to see:



5. Scatter Plot → distribution

- shows dots.
- Use to show if there are
 - cluster
 - wider spread



6. Radar or Spider chart → distribution

- Compare multiple objects with different dimensions
- commonly used in
 - games, athlete stats, student performance



7. Density Map / dot map → distribution

↳ Geographical

- use dot or color density to represent the geographical distribution of data

Example use:

typhoon, covid-cases, location that sells best



8. funnel chart → Flow with Trends

- use how data move through a process in stages

A funnel chart typically displays:

1. A process with sequential stages
2. The number of items (customers, leads, tasks, etc.) at each stage
3. How many are lost or drop off between stages
4. The conversion rate from one stage to the next

The chart shape makes it visually clear where the biggest reductions happen.

What Makes Funnel Charts Valuable

- ✓ Immediate insight into conversion and loss
- ✓ Easy comparison of performance month to month
- ✓ Helps identify bottlenecks and areas needing improvement
- ✓ Supports forecasting (e.g., expected revenue from leads)



Overall Chart Types

Composition



Pie chart



Stacked column chart

Comparison



Column chart



Bar chart



Radar charts

Relationship



Line chart

Distribution



Scatter chart



Dot Map

Keys for Choosing the right chart

- 1 What you want your data to tell
- 2 Do you want to compare things, show trends, or show geographical patterns?

Introduction to Basic Python

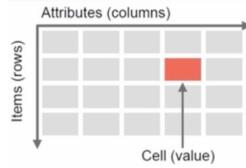
Already have background in Python

Will be using Jupyter Notebook

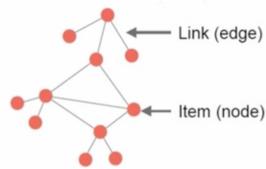
Short Review

Data Collection

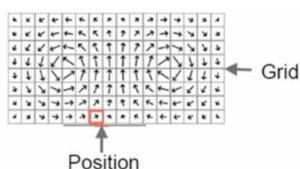
Data Types



Table



Networks and Trees



Grids and Positions

Table

- created using items (rows) and attribute (column)

Networks and Trees

- include items called nodes which are connected with link (edges)

Grids & Positions

- grids are like table
- positions are the coordinate (row, column)

Data Structure

- organize your data by storing and structuring it making it easier to access

Data Structure

- Foundation: Lists and Tuples
- Complex: Sets and Dictionaries
- Advanced: Series and Data frames

LIST

↳ []

TUPLES

↳ ()

Feature	List	Tuple
Changeable (mutable)	✓ Yes	✗ No
Syntax	[]	()
Speed	Slower	Faster
Use case	Data that changes	Data that stays constant
Methods available	Many (append, remove, etc.)	Very few
Memory usage	More	Less

List Example

```
python
fruits = ["apple", "banana"]
fruits.append("orange")
# Now: ["apple", "banana", "orange"]
```

Copy code

Tuple Example

```
python
colors = ("red", "green", "blue")
colors.append("yellow") # ✗ Error
```

Copy code

When to Use Which?

- ✓ Use a LIST when:
 - Items will change
 - You want to add/remove stuff
 - You're making dynamic data (e.g., user input, database data)

- ✓ Use a TUPLE when:

- Data should NEVER change
- You want faster performance
- You need something unchangeable (keys, constants, coordinates)

Example:

```
python
position = (10, 20) # A tuple makes sense for coordinates
```

Copy code

Set

- no duplicate, no order

↳ { }

If you do this:

```
python
my_set = {1, 2, 2, 3}
```

Python turns it into:

```
python
{1, 2, 3}
```

Because duplicates are automatically removed.

Dictionary

- key + value pairs

Think of it like a real dictionary:

- The word is the key
- The definition is the value

```
python
```

```
my_dict = {
    "name": "John",
    "age": 20
}
```

You look things up by their key.

SERIES

↳ we use `pandas` library

`import pandas as pd`

`pd.Series([..., ...])`

SERIES = One column

A Series is basically a single column of data.

Think of it like one column in Excel:

css

```
0 John  
1 Mark  
2 Ana
```

This is a Series.

Code example:

python

```
import pandas as pd  
  
s = pd.Series(["John", "Mark", "Ana"])
```

Series = 1D array
Dataframe = 2D array

Why use Series ↗ Dataframe?

Series has an index

```
makefile  
  
Index: 0 1 2  
Value: 10 20 30
```

You can change the index:

```
makefile  
  
Index: age1 age2 age3  
Value: 10 20 30
```

Try that in a normal array → ❌ impossible.

2. DataFrame has column names

A DataFrame looks like this:

```
css  
  
name age salary  
John 30 50000  
Mark 22 45000
```

This is NOT just a 2D array — because:

- Each column has a name
- Rows have index labels
- You can access data by column name, not just by number

Example:

```
python  
  
df["age"]
```

A normal 2D array cannot do that.

Data Frame

A DataFrame is basically a whole table — multiple columns, multiple rows.

Think of it like an Excel sheet with many columns:

css

```
name age  
0 John 30  
1 Mark 22  
2 Ana 28
```

`pd.DataFrame({
 "key": [..., .., ..],
 :
})`

This is a DataFrame.

Code example:

python

```
df = pd.DataFrame(  
    "name": ["John", "Mark", "Ana"],  
    "age": [30, 22, 28]  
)
```

Key Characteristics



Mutability: Refers to whether the collection can be changed after creation.



Order: Refers to whether elements have a specific sequence.



Duplicate: Refers to whether the collection can contain the same element multiple time.

	Mutability	Order	Duplicate
List	✓	✓	✓
Tuples	✗	✓	✓
Dictionary	✓	✗	✓
Sets	✓	✗	✗
Series	✓	✓	✓

Adding/Removing Elements:

- `append(x)`: Adds an element `x` to the end of the list.
- `insert(i, x)`: Inserts an element `x` at a specific index `i`.
- `remove(x)`: Removes the first occurrence of element `x` from the list.
- `pop(i)`: Removes and returns the element at index `i` (defaults to the last element if no index provided).
- `clear()`: Removes all elements from the list.

Length

- `len(list_name)`: Returns the number of elements in the list.

Tuples**Concatenate**

```
# Create Tuples
MarksCIS = (70, 85, 55)
MarksCIN = (90, 75, 60)
# Concatenating Tuples
Combined=MarksCIS + MarksCIN
print(Combined)

(70, 85, 55, 90, 75, 60)
```

Access

```
# Create Tuples
MarksCIS = (70, 85, 55)
MarksCIN = (90, 75, 60)
print ("The third mark in CIS is ", MarksCIS[2])
print ("The third mark in CIN is ", MarksCIN[2])

The third mark in CIS is 55
The third mark in CIN is 60
```

Ordering:

- `sort()`: Sorts the list elements in ascending order (in-place modification).
- `reverse()`: Reverses the order of elements in the list (in-place modification).

Finding Elements:

- `index(x)`: Returns the index of the first occurrence of element `x` (raises an error if not found).
- `count(x)`: Returns the number of times element `x` appears in the list.

**Tuples****Finding Elements:**

- `index(x)`: Returns the index of the first occurrence of element `x` (raises an error if not found).
- `count(x)`: Returns the number of times element `x` appears in the tuple.

Length:

- `len(tuple_name)`: Returns the number of elements in the tuple.

Slicing: Accessing elements by position range (similar to lists).

- `tuple[start:end:step]`: Returns a new tuple containing elements from index start (inclusive) to end (exclusive) with a step of step (defaults to 1).

Conversion:

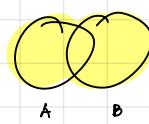
- `tuple(iterable)`: Converts an iterable (like a list or string) to a tuple.

Dictionary**Methods and Functions:**

- `get(key, default)`: Retrieves the value associated with the key. If the key is not found, returns the default value (optional).
- `pop(key, default)`: Removes the key-value pair and returns the value. If the key is not found, returns the default value (optional).
- `keys()`: Returns a view of all keys as a dictionary view object.
- `values()`: Returns a view of all values as a dictionary view object.
- `items()`: Returns a view of all key-value pairs as tuples in a dictionary view object.
- `update(dictionary)`: Updates the dictionary with key-value pairs from another dictionary (modifies the original dictionary).
 - `dict(iterable)`: Converts an iterable of key-value pairs into a dictionary.

Mathematic Set Operation

- **Union**
- **Intersection**
- **Difference**
- **Symmetric Difference**

1. Union

`A = {1, 2, 3, 4, 5}`

`B = {4, 5, 6, 7, 8}`

use | operator

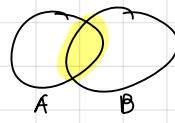
Output: {1, 2, 3, 4, 5, 6, 7, 8}

`print(A | B)`

`print(A.union(B))`

{1, 2, 3, 4, 5, 6, 7, 8}

{1, 2, 3, 4, 5, 6, 7, 8}

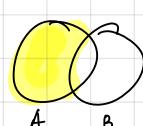
2. Intersection

`print(A & B)`

`print(A.intersection(B))`

{4, 5}

{4, 5}

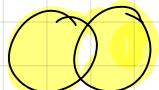
3. Difference

`print(A - B)`

`print(A.difference(B))`

{1, 2, 3}

{1, 2, 3}

4. Symmetric Difference

`print(A ^ B)`

`print(A.symmetric_difference(B))`

{1, 2, 3, 6, 7, 8}

{1, 2, 3, 6, 7, 8}

```
df = pd.DataFrame({
    "name": ["John", "Ana"],
    "age": [20, 25]
})
```

Column

Index	name	age
0	John	20
1	Ana	25

Index in Pandas

You can change index name using
`df.index`

Example:

```
python
df = pd.DataFrame({"name": ["A", "B", "C"]})
print(df)
```

Output:

```
css
name
0 A
1 B
2 C
```

Rows automatically numbered.

2. Custom Index

You can replace the index with anything:

```
python
df.index = ["a", "b", "c"]
```

Output:

```
css
name
a A
b B
c C
```

New rows are labeled a b c

4. Resetting the Index

You can convert the index BACK to a normal column:

`df.reset_index()`

This will make the old index a column and create a new default index.

Example:

Before:

```
bash
name
id
101 Mark
102 Jay
```

After:

```
bash
id name
0 101 Mark
1 102 Jay
```

Setting a column as index
`df.set_index()`

Most real datasets have a column like "id", "code", or "email" to uniquely identify row.

You can turn that into the index:

python

```
df = df.set_index("id")
```

Example:

Before:

bash

id	name	age
0	John	20
1	Ana	22

After:

bash

	name	age
id		
1	John	20
2	Ana	22

The column "id" is now the row identifier.

Index must be
UNIQUE

loc vs iloc

2. Selecting Data (loc vs iloc)

★ loc = label-based

Use names, not numbers.

python

```
df.loc["row1"]           # select row with label "row1"
df.loc[:, "age"]         # select age column
df.loc["row1", "age"]    # specific cell
```

★ iloc = position-based

Use numbers only (like array indexing).

python

```
df.iloc[0]                # first row
df.iloc[:, 1]              # second column
df.iloc[0, 1]              # row 0, column 1
```

Easy memory trick:

- loc = label
- iloc = integer index

How to add or change data in pandas df and series

1. INDEX in a Series

Example Series

```
python  
import pandas as pd  
  
s = pd.Series([10, 20, 30], index=["a", "b", "c"])  
print(s)
```

Visual:

index	value
a	10
b	20
c	30

2. ADDING a new index + value to Series

Method 1 — direct assignment

```
python  
s["d"] = 40
```

Result:

index	value
a	10
b	20
c	30
d	40

3. MODIFYING an existing index's value

```
python  
s["b"] = 999
```

Result:

index	value
a	10
b	999
c	30
d	40

4. RENAMING index labels (Series)

```
python  
s = s.rename({"a": "alpha", "b": "beta"})
```

Result:

index	value
alpha	10
beta	999
c	30
d	40

Apply Python coding to generate the expected output.

1. Write a program to create a list of names; then define a function to display all the elements in the received list. Call the function to execute its statements and display all names in the list. (Names: Ahmed, Ali, Salim, Abdullah, and Salwa)
2. Write a program to create a series to maintain five students' names above and GPA values. (GPA: 2.5, 3.0, 2.5, 3.5, 4.0)
3. Write a program to create a data frame to maintain three students' names associated with their grades in three courses and then add a new column named Mean to maintain the calculated mean mark per course. Display the final data frame.

Expected Result

Ahmed	Ali	Salim	Abdullah	Salwa	
Course1	65	70	69	75	92
Course2	60	78	68	80	94
Course3	78	80	72	84	98

→

Ahmed	Ali	Salim	Abdullah	Salwa	Mean
Course1	65	70	69	75	92
Course2	60	78	68	80	94
Course3	78	80	72	84	98

```
: import pandas as pd
```

1. Make List of names

Write a program to create a list of names; then define a function to display all the elements in the received list. Call the function to execute its statements and display all names in the list. (Names: Ahmed, Ali, Salim, Abdullah, and Salwa)

```
: names = ["Ahmed", "Ali", "Salim", "Abdullah", "Salwa"]  
print(names)  
['Ahmed', 'Ali', 'Salim', 'Abdullah', 'Salwa']
```

3. Create a Series of grades

Write a program to create a series to maintain five students' names above and GPA values. (GPA: 2.5, 3.0, 2.5, 3.5, 4.0)

```
: grades = pd.Series([2.5, 3.0, 2.5, 3.5, 4.0])  
print(grades)  
0    2.5  
1    3.0  
2    2.5  
3    3.5  
4    4.0  
dtype: float64
```

3. Create dataframe

Write a program to create a data frame to maintain three students' names associated with their grades in three courses and then add a new column named Mean to maintain the calculated mean mark per course. Display the final data frame.

```
table = pd.DataFrame({  
    "": ["Course1", "Course2", "Course3"],  
    "Ahmed": [65, 60, 78],  
    "Ali": [70, 78, 80],  
    "Salim": [69, 68, 72],  
    "Abdullah": [75, 80, 84],  
    "Salwa": [92, 94, 98],  
})  
  
# Set 'Course1' as index  
table = table.set_index("") # or use table.set_index("Course1", inplace=True)  
  
print(table)
```

	Ahmed	Ali	Salim	Abdullah	Salwa
Course1	65	70	69	75	92
Course2	60	78	68	80	94
Course3	78	80	72	84	98

```
table["Mean"] = table.mean(axis=1)  
print(table)
```

	Ahmed	Ali	Salim	Abdullah	Salwa	Mean
Course1	65	70	69	75	92	74.2
Course2	60	78	68	80	94	76.0
Course3	78	80	72	84	98	82.4

table["mean"] = table.mean(axis=1)

Create new
Column named
"mean"

1 = rows
0 = column

means calculate mean across rows