Uptrail

FOUNDATIONS OF
DATA ANALYSIS

Week 1

ROADMAP

Week **1** 

Foundations of Data Analysis

- Data roles & lifecycle
- Python setup (Jupyter, libraries)
- NumPy & Pandas basics

Week **2** 

Data Wrangling & Visualisation

- Filtering, merging, reshaping
- Matplotlib & Seaborn
- Feature engineering

Week **3** 

Statistics & Predictive Modelling

- Hypothesis testing
- Regression & classification
- Model evaluation

Week **4** 

Dashboards, BI & AI in Analytics

- Tableau dashboards
- AutoML & smart insights
- Storytelling with data

🧭 Each week builds core capability — from basics to business-ready analytics.

TABLE OF CONTENTS

Part **1**: Foundation

TABLE OF CONTENTS

Part 1: Foundation

TABLE OF CONTENTS

Part **2**: Code Walkthrough

TABLE OF CONTENTS

Part **2**: Code Walkthrough

INTRODUCTION TO DATA ANALYSIS

What is Data Analysis?

It is the process of systematically **collecting**, **cleaning**, **transforming**, and **interpreting** data to discover useful information, draw conclusions, and support **decision-making**.

Reading Material: https://www.geeksforgeeks.org/what-is-data-analysis/

WHERE DATA ANALYSIS IS USED?

01. Business Intelligence (BI)

Strategic reporting & decision-making
Example: Dashboards for KPIs, sales trends, performance metrics

02. Product

Improve features & user experience
Example: Track feature usage, run A/B tests, analyze user funnels

03. Marketing

Target audiences & optimize campaigns
Example: A/B testing, customer segmentation, ROI tracking

04. Operations

Boost efficiency & reduce costs
Example: Supply chain analysis, staffing optimization, process improvement

THE DATA ANALYTICS LIFECYCLE

Gather raw data from sources like databases, surveys, logs, or APIs.

Collect

Clean

Remove errors & inconsistencies: handle missing values, duplicates, and formatting.

Explore and model the data to uncover patterns, trends, and correlations.

Analyze

Interpret

Translate findings into meaningful insights aligned with business goals.

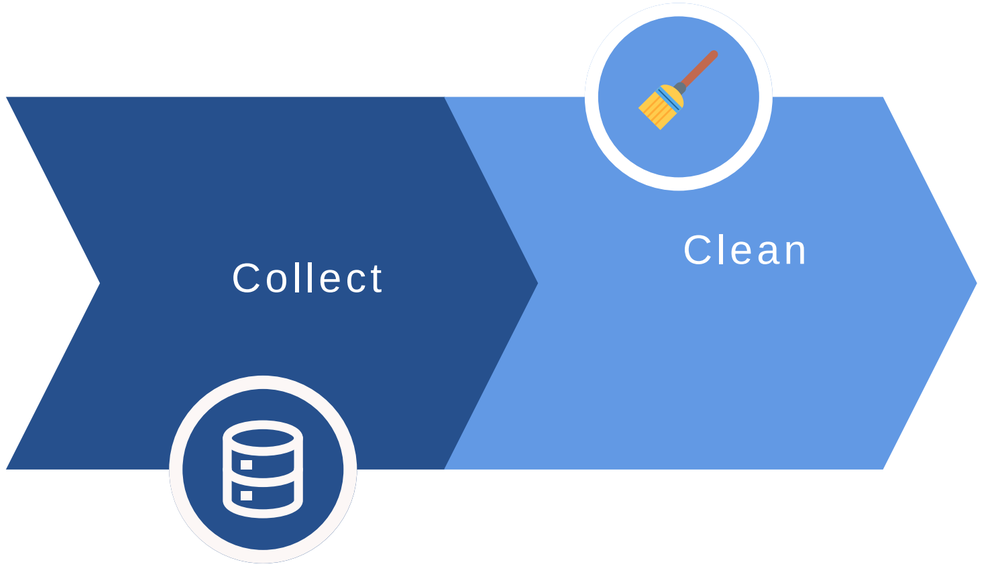
Make decisions or take action : implement changes, monitor results, and iterate.

Act

ROLES & CAREER PATHS

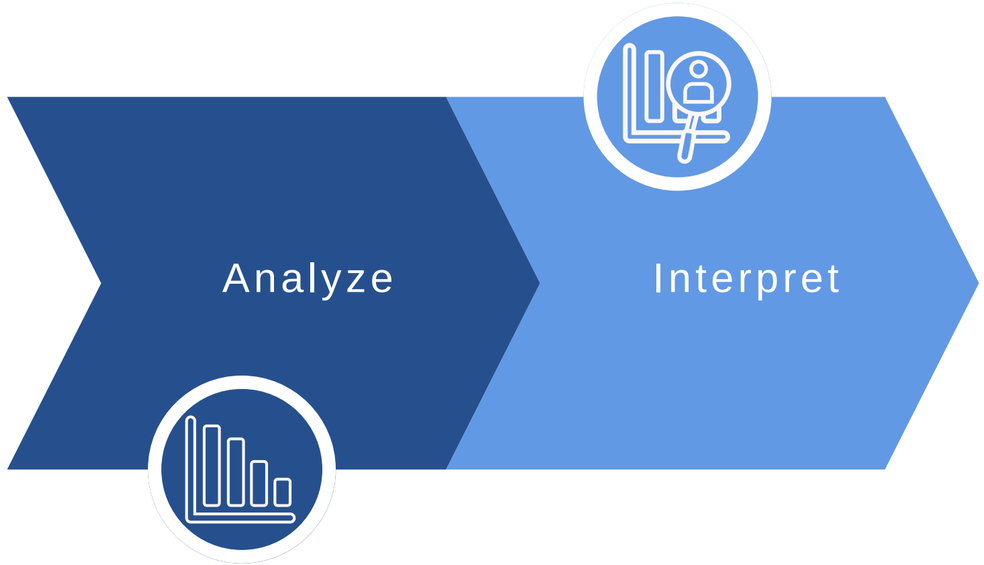
DATA ENGINEER

- **Focus:** *Building data infrastructure*
- **Tools:** SQL, Python, Spark, Airflow, Cloud (AWS/GCP/Azure)
- **Key Tasks:**
  - Data pipelines
  - ETL processes
  - Database architecture



DATA ANALYST

- **Focus:** *Understanding and interpreting data*
- **Tools:** SQL, Excel, Tableau, Power BI
- **Key Tasks:**
  - Data querying
  - Reporting & dashboards
  - Business insights



DATA SCIENTIST

- **Focus:** *Advanced analysis & prediction*
- **Tools:** Python, R, Jupyter, ML libraries
- **Key Tasks:**
  - Machine learning
  - Experimentation
  - Forecasting

TYPES OF DATA

Structured

- Excel **spreadsheets** with customer orders
- SQL database of employee records (name, salary, department)
- Inventory **tables** in e-commerce systems

tables

Semi-Structured

- JSON response from a weather API
- XML files used in RSS feeds
- NoSQL documents in MongoDB
- Config files (YAML, TOML) used in web apps

json/ xml

Unstructured

- Customer support emails
- Social media posts (X, Reddit)
- Product images on an online store
- Audio recordings from interviews or calls
- PDFs with freeform text

.mp3, jpg, .mp4

TYPES OF ANALYSIS

Descriptive:

"**What happened?**"

Summarises past data to identify trends and patterns.

Diagnostic:

"**Why did it happen?**"

Analyzes data to uncover the causes behind past outcomes.

Predictive:

"**What might happen?**"

Uses historical data and models to forecast future outcomes.

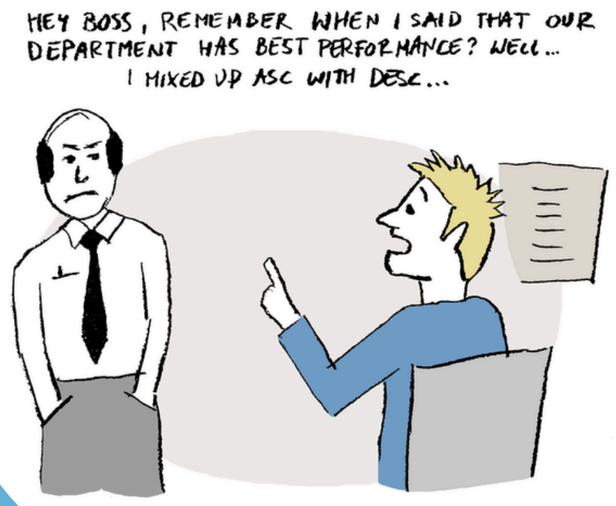
Prescriptive:

"**What should we do?**"

Recommends actions based on predicted scenarios to optimise results.

4 Types of Data Analytics to Improve Decision-Making ~ By Harvard Business School
https://online.hbs.edu/blog/post/types-of-data-analysis

DATA QUALITY PILLARS



Bad data = incorrect decisions

Accuracy
Are values correct?

Completeness
Are records / fields missing?

Consistency
Is data uniform across sources?

Timeliness
Is the data current?

Validity
Do values match allowed formats?

General Data Protection Regulation (GDPR) Guidelines
~ https://gdpr.eu/what-is-gdpr

TOOLS OF THE TRADE

CODE WALKTHROUGH

~ENVIRONMENT SETUP • DATA MANIPULATION • EDA

PRE-REQUISITES & INSTALLATION

How to Install Anaconda & Jupyter Notebook (Step-by-Step)" ~ ProgrammingKnowledge
https://www.youtube.com/watch?v=YJC6ldI3hWk

Official Guide from Anaconda for installation:
https://docs.anaconda.com/free/anaconda/install/

Markdown Cheat Sheet (PDF):
https://www.markdownguide.org/cheat-sheet

LIBRARIES NumPy & Pandas

- NumPy (Numerical Python)
  - Fast array operations and numerical computing
  - Supports multi-dimensional arrays (ndarray)
  - Ideal for mathematical and statistical operations
- Pandas (Panel Data)
  - Built on top of NumPy
  - Provides DataFrame and Series for tabular data
  - Makes data loading, cleaning, and analysis intuitive

```
# Install Jupyter in terminal/command prompt if needed:
# pip install notebook

# In Jupyter Notebook:
import pandas as pd
import numpy as np
```

Python for Data Science
~ https://www.youtube.com/watch?v=LHBE6Q9XlzI&t=6s

KEY CONCEPTS:

array()

Arrays

Core data structure in NumPy for storing and manipulating numerical data efficiently.

```
1  a = np.array([1, 2, 3])
2  print("Array:", a)
```

```
Array: [1 2 3]
```

“Why NumPy? It’s fast and memory-efficient.”

KEY CONCEPTS:

array() | arange()

Arange

Generates evenly spaced values within a given range, similar to Python's range()

```
1  b = np.arange(0, 10, 2)
2  print("Arange:", b)
```

```
Arange: [0 2 4 6 8]
```

KEY CONCEPTS:

array() | arange() | linspace()

linspace

Creates an array of evenly spaced values between two numbers, including both endpoints.

```
1  c = np.linspace(0, 1, 5)
2  print("Linspace:", c)
```

```
Linspace: [0.   0.25 0.5  0.75 1.  ]
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting

Slicing

Allows access to subsets of an array using index ranges

Broadcasting

Automatically expands arrays with different shapes during arithmetic operations

```
1  # Slicing and broadcasting
2  d = np.array([10, 20, 30, 40, 50])
3  print("Slice d[1:4]:", d[1:4])
4  d[1:4] = 100
5  print("After broadcasting:", d)
```

```
Slice d[1:4]: [20 30 40]
After broadcasting: [ 10 100 100 100  50]
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile()

```
# Summary functions
print("Mean:", np.mean(d))
print("Std Dev:", np.std(d))
print("90th percentile:", np.percentile(d, 90))
```

```
Mean: 72.0
Std Dev: 36.55133376499413
90th percentile: 100.0
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame

DataFrame:

A two-dimensional, labelled data structure in Pandas, similar to a table or spreadsheet.

```python
# Creating DataFrames
df1 = pd.DataFrame([[1, 2], [3, 4]], columns=['A', 'B'])
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
df3 = pd.DataFrame(np.random.rand(3, 2), columns=['X', 'Y'])

print("DataFrame 1:\n", df1)
print("DataFrame 2:\n", df2)
print("DataFrame 3:\n", df3)
```

```
DataFrame 1:
    A  B
0   1  2
1   3  4
DataFrame 2:
    A  B
0   5  7
1   6  8
DataFrame 3:
          X         Y
0  0.007739  0.109376
1  0.408546  0.747955
2  0.230457  0.540678
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail()

```
1  print("Head Method:\n",df1.head())
2  print("\nTail Method:\n",df2.tail(10))
```

```
Head Method:
    A  B
0   1  2
1   3  4

Tail Method:
    A  B
0   5  7
1   6  8
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info()

info()

Displays a summary of the DataFrame's structure, including column types and non-null counts.

```
1 print(df2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       2 non-null      int64
 1   B       2 non-null      int64
dtypes: int64(2)
memory usage: 160.0 bytes
None
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe()

describe()

Generates descriptive statistics for numeric columns, such as mean, median, and standard deviation

```
1 print("\nDescribe Method:\n",df3.describe())
```

```
Describe Method:
                X         Y
count   3.000000  3.000000
mean    0.215581  0.466003
std     0.200817  0.325773
min     0.007739  0.109376
25%     0.119098  0.325027
50%     0.230457  0.540678
75%     0.319502  0.644316
max     0.408546  0.747955
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering

DataFrame 1:

	A	B
0	1	2
1	3	4

```python
# Selecting data
print("\nloc:\n",df1.loc[0, 'A'])  # Label-based
print("\niloc:\n",df1.iloc[1, 1])   # Position-based
print("\nFiltering:\n",df1[df1['A'] < 3])  # Filtering

```

```
loc:
 1

iloc:
 4

Filtering:
    A  B
0  1  2
```

.loc is label-based; .iloc is position-based

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename()

```
id             int64
birthdate     object
category      object
dtype: object
   ID category  birthdate
0   1        A 2000-01-01
1   2        B 1995-05-15
2   3        A 1988-07-30
```

Data Types and Data Cleaning

```python
df = pd.DataFrame({
    'id': [1, 2, 3],
    'birthdate': ['2000-01-01', '1995-05-15', '1988-07-30'],
    'category': ['A', 'B', 'A']
})

# Check types
print(df.dtypes)

# Convert types
df['birthdate'] = pd.to_datetime(df['birthdate'])
df['category'] = df['category'].astype('category')

# Rename columns
df.rename(columns={'id': 'ID'}, inplace=True)

# Reorder columns
df = df[['ID', 'category', 'birthdate']]
print(df)
```

Data Cleaning in Pandas ~ https://www.youtube.com/watch?v=bDhvCp3_lYw

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN

Handling Missing and Duplicate Data

1. Identify missing values

```python
df = pd.DataFrame({
    'A': [1, np.nan, 3, 3],
    'B': [4, 5, np.nan, 4]
})

# Detect missing
print(df.isnull().sum())
```

```
A    1
B    1
dtype: int64
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate()

```python
# Fix missing
df_filled = df.fillna(0)
df_dropped = df.dropna()
df_interp = df.interpolate()

print("Replaced:\n",df_filled)
print("Removed:\n",df_dropped)
print("Interpolate\n",df_interp)
```

```
Replaced:
      A    B
0  1.0  4.0
1  0.0  5.0
2  3.0  0.0
3  3.0  4.0
Removed:
      A    B
0  1.0  4.0
3  3.0  4.0
Interpolate
      A    B
0  1.0  4.0
1  2.0  5.0
2  3.0  4.5
3  3.0  4.0
```

2. Handle missing values

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates()

```python
df = pd.DataFrame({
    'A': [1, 2, 3, 1],
    'B': [4, 5, 5, 4]
})

print("Duplicate Rows:\n",df.duplicated())
# Remove duplicates
df_no_dupes = df.drop_duplicates()
print("\nCleaned DataFrame:\n",df_no_dupes)
```

```
Duplicate Rows:
 0    False
1    False
2    False
3     True
dtype: bool

Cleaned DataFrame:
    A  B
0  1  4
1  2  5
2  3  5
```

2. Handle duplicate records

ERROR HANDLING AND DEBUGGING

Common errors in data workflow:

KeyError

Occurs when referencing a non-existent column or key

➤ df['unknown_column'] →
❌ KeyError

TypeError

Happens when using incompatible data types in operations

➤ df['age'] + 'years' →
❌ TypeError

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates() | Try/Except | is_numeric_dtype()

```python
# Try/Except
try:
    print(df['Nonexistent'])
except KeyError as e:
    print("KeyError caught:", e)

# Common issues
# TypeError example:
try:
    df + 'string'
except TypeError as e:
    print("TypeError:", e)

# Best practice: check before operations
if 'A' in df.columns:
    print("A exists!")
    if pd.api.types.is_numeric_dtype(df['A']):
        print("Mean: ", df['A'].mean())
```

```
KeyError caught: 'Nonexistent'
TypeError: ufunc 'add' did not contain a loop with signature matching types (dtype('float64'), dtype('<U6')) -> None
A exists!
Mean:  2.3333333333333335
```

OUTLIER DETECTION: WHAT, WHY, AND HOW

What Are Outliers?

Outliers are unusually high or low values that differ significantly from other data points, they can skew analysis or reveal important anomalies.

~ Boxplot, IQR, Z-Score

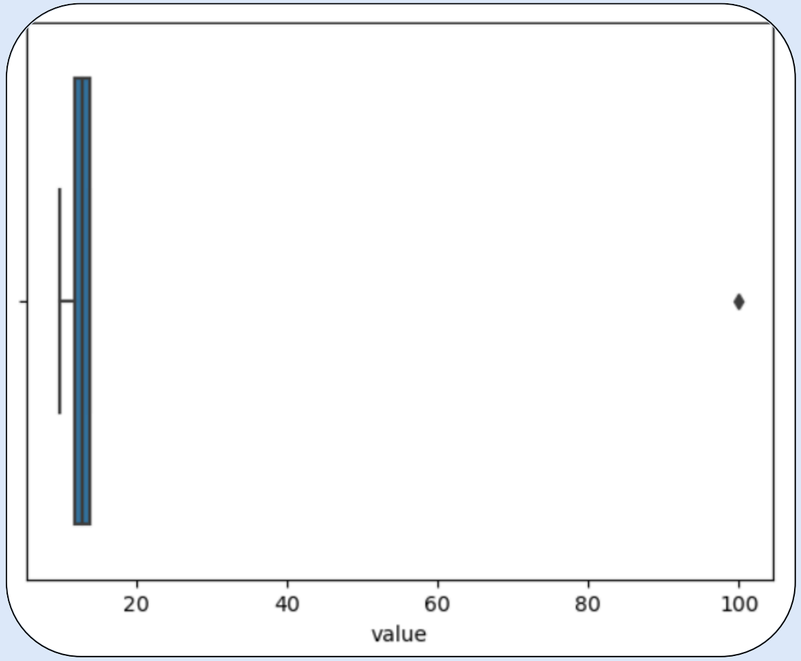
Complete Hands-on Outlier Treatment
~ https://www.youtube.com/watch?v=4FBPtgLbehY

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates() | Try/Except | is_numeric_dtype() | boxplot()

Boxplot (Visual)

- Great for quick visual checks
- Uses IQR logic internally




```python
# Boxplot
import seaborn as sns

data = pd.DataFrame({
    'value': [10, 12, 13, 14, 100]
})
sns.boxplot(x=data['value'])
plt.show()
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates() | Try/Except | is_numeric_dtype() | boxplot() | quantile()

IQR

- Best when data is not normally distributed
- Uses percentiles to flag outliers

```python
# IQR method
Q1 = data['value'].quantile(0.25)
Q3 = data['value'].quantile(0.75)
IQR = Q3 - Q1
outliers = data[(data['value'] < Q1 - 1.5 * IQR) | (data['value'] > Q3 + 1.5 * IQR)]
print("Outliers using IQR:", outliers)
```

```
Outliers using IQR:
    value
4     100
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates() | Try/Except | is_numeric_dtype() | boxplot() | quantile() | zscore()

Z-Score

- Suitable when data is normally distributed
- Flags values [> or < 3] standard deviations from the mean

```python
# Z-score method
from scipy.stats import zscore
z_scores = zscore(data['value'])
print("Z-scores:", z_scores)
print("Outliers (z > 2):", data[np.abs(z_scores) > 2])
```

```
Z-scores:
 0   -0.563702
1   -0.506763
2   -0.478293
3   -0.449823
4    1.998581
Name: value, dtype: float64
```

```
Outliers (z > 2):
 Empty DataFrame
Columns: [value]
Index: []
```

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates() | Try/Except | is_numeric_dtype() | boxplot() | quantile() | zscore() | value_counts() | groupby() | mean() | describe()

EDA

Exploratory Data Analysis: A process of visually and statistically exploring datasets to uncover **patterns**, **trends**, and insights **before** formal **modelling**.

```
Sample Data:
   gender  score
0       M     88
1       F     92
2       F     85
3       M     70
4       M     90
```

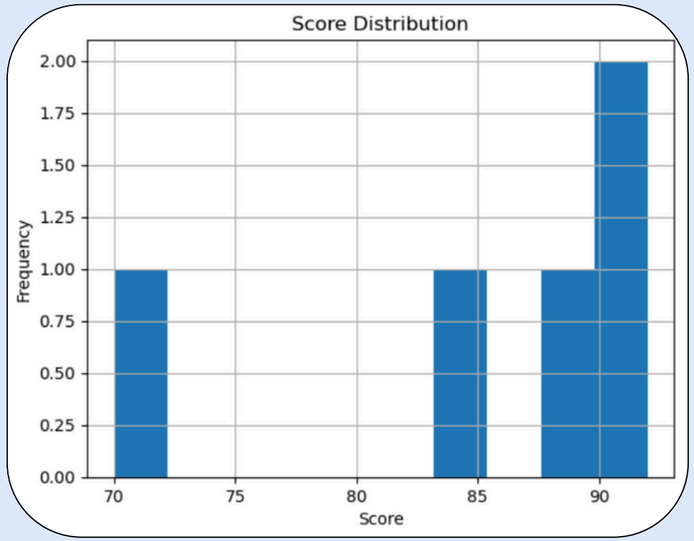
```
Value Counts:
 gender
M    3
F    2
Name: count, dtype: int64

GroupBy:
 gender
F    88.500000
M    82.666667
Name: score, dtype: float64

            score
count    5.000000
mean    85.000000
std      8.774964
min     70.000000
25%     85.000000
50%     88.000000
75%     90.000000
max     92.000000
```

```python
# Descriptive statistics
print("\nValue Counts:\n",df['gender'].value_counts())
print("\nGroupBy:\n",df.groupby('gender')['score'].mean())
print("\n",df.describe())
```

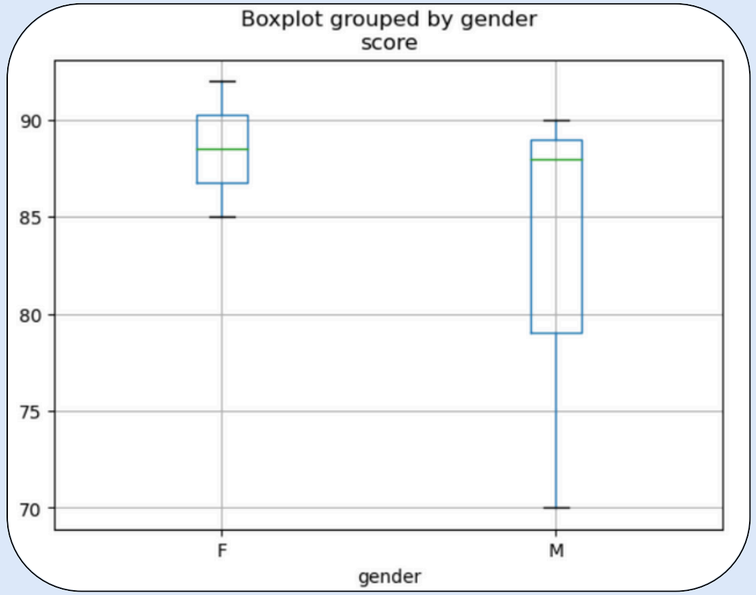
EDA with Pandas ~ https://www.youtube.com/watch?v=Liv6eeb1VfE

KEY CONCEPTS:

array() | arange() | linspace() | Slicing | Broadcasting | mean() | std() | percentile() | random() | DataFrame | head() | tail() | info() | describe() | loc() | iloc() | Filtering | dtypes | to_datetime() | astype() | rename() | isnull() | sum() | NaN | fillna() | dropna() | interpolate() | duplicated() | drop_duplicates() | Try/Except | is_numeric_dtype() | boxplot() | quantile() | zscore() | value_counts() | groupby() | mean() | describe() | hist()




```python
# EDA with Visuals
df['score'].hist()
plt.title('Score Distribution')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.show()

df.boxplot(column='score', by='gender')
plt.show()
```



Left skewed overall

Boxplot not only compares medians but also reveals **shape and spread** of distributions across groups.

Gender	Spread	Skewness	Central Tendency
F	Low	Symmetric	Median near 89
M	High	Positively skewed	Median near 88 (lower)

Uptrail

QUESTIONS?

www.uptrail.co.uk

support@uptrail.com