Course: Data Science Tools and Techniques

Data Preprocessing

Dr. Safdar Ali

Explore and discuss the process of data cleaning, with understanding of its importance, common challenges, and effective techniques along with data transformation.

Number of Lectures	Topics	Material
3	**Introduction** Data Science Life Cycle, Motivation, Market Value, Issues, Challenges and Opportunities	Reference Textbook, Online references
6	**Data Preprocessing** Clean and filter the data, convert the data from one format to another	Reference Textbook, Online references
3	**Data Visualization** Visualizing the data in different ways	Reference Textbook
6	**Probabilistic View of Data** Basics of probability and statistics, Bayes rule, text Modelling	Reference Textbook
8	**Data Modelling** Machine Learning, classification, regression, clustering	Online Reference
2	**Model Evaluation & Performance Metrics** Train/val/test splits, accuracy, precision-recall, F-1, etc.	Reference Textbook
8	**Big Data Processing Tools** Hadoop and its Ecosystem, Spark	Online references, Research Papers
6	**Diverse Topics in Data Science** Various recent Trends in Data Science, Ethical Issues, Research Opportunities	Research Papers

Data preprocessing

- It is a **broader concept** that includes data cleaning and *other steps* to prepare the data for machine learning algorithms.
- Other steps may include data transformation, feature selection, normalization, and reduction.
- **Goal of data preprocessing** is to convert raw data into a suitable format that machine learning algorithms can learn.

Data Collection

- **Foundational step** where raw data is gathered from various sources, such as databases, spreadsheets, APIs, or surveys.
- **Essential to ensure** that the data collected is relevant to the analysis goals.
- **Define the scope and objectives** of the data collection process, determining the sources of data, and gathering it systematically.
- **High-quality initial data** leads to better cleaning outcomes, making this step crucial for setting the stage for the subsequent cleaning process.

- ## Original data (fixed column format)

000000000130.06.19971979-10-3080145722   #000310 111000301.01.000100000000004
0000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.0000
00000000000. 000000000000000.000000000000000.0000000......
000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.00
0000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.0000
00000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000
000000000.000000000000000.000000000000000.000000000000000.00 0000000000300.00 0000000000300.00

- ## Clean data

0000000001,199706,1979.833,8014,5722   ,   ,#000310 ....
,111,03,000101,0,04,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0300,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0300,0300.00

Data Profiling

- It involves **analyzing the dataset** to understand its characteristics, structure, and quality.
- **Assessing** data types, distributions, and overall completeness.
- **Identify issues** such as missing values, duplicates, inconsistencies, and outliers.
- **By generating summary statistics and visualizations**, gain insights into the data's integrity and identify areas that require attention.
- This **initial assessment informs** the specific cleaning actions needed and helps prioritize efforts based on the data's condition.

Data Cleaning in Data Science

- **Quality of data** is fundamental in ensuring accurate and meaningful analysis.
- **Raw data**, fresh from its source, is often messy and riddled with inconsistencies, errors, and missing values
- **Data cleaning**, also known as data cleansing or scrubbing, is a critical step in the data science process, ensuring that dataset is accurate, consistent, and ready for analysis
- **Without proper data cleaning**, the insights drawn from analysis may be flawed, leading to incorrect conclusions and potentially costly decisions.
- **Poor data quality** can lead to unreliable outcomes, regardless of *how advanced the algorithms or techniques used are*.

Data quality

Why data quality matters and how it affects outcomes:

- **Impact of data quality on analysis**
- **Key aspects of data quality**
- **How to ensure high-quality data**
- **Real-world examples of data quality's impact**

Let us discuss these one by one

Impact of Data Quality on Analysis

High-quality data leads to:

- **More accurate models**: If the data is clean, machine learning models will generalize more accurate predictions.

- **Valid insights**: insights drawn are valid and applicable to real-world problems. Garbage-in, garbage-out (GIGO) applies —bad data leads to misleading conclusions.

- **Better decision-making**: Whether for business, healthcare, or research, good data supports informed and confident decision-making. It allows analysts to spot trends and patterns that would otherwise be hidden.

Impact of Data Quality on Analysis

Low-quality data causes:

- **Model performance degradation**: If data is noisy, inconsistent, incomplete, or biased, models may overfit, underfit, or perform poorly, leading to erroneous predictions.

- **Inaccurate analysis**: Insights derived from flawed data are often misleading and could result in wrong business strategies, policy decisions, or scientific conclusions.

- **Increased costs**: Poor data quality can lead to costly errors down the line. Fixing incorrect analysis after decisions have been made often requires rework and may damage reputations or relationships.

Underfitting and Overfitting

Reasons for Underfitting:

- Model is too simple and not capable to represent complexities in the data.
- Input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.
- Size of the training dataset used is not enough.
- Features are not scaled.

Reasons for Overfitting:

- High variance and low bias.
- The model is too complex.
- The size of the training data

Key Aspects of Data Quality

Accuracy: Closeness of the data to the true or correct value.

- **Impact**: Inaccurate data leads to misleading conclusions. For example, if a dataset contains incorrect medical diagnoses, it will lead to poor predictions or treatment recommendations.

Completeness: Extent to which all required data is available.

- **Impact**: Missing data can skew results and introduce bias. For instance, missing customer information in a sales dataset could lead to incorrect segmentation or targeting in marketing campaigns.

Consistency: Data that is uniform and without contradictions across different datasets.

- **Impact**: Inconsistencies in data — like different formats for dates or conflicting records—can lead to confusion and incorrect conclusions.

Key Aspects of Data Quality

Timeliness: Whether data is up-to-date and relevant to the analysis.

- **Impact**: Outdated data can lead to faulty predictions. For instance, an economic model trained on outdated consumer spending data would fail to reflect current trends.

Relevance: Data that is appropriate and aligned with the problem at hand.

- **Impact**: Irrelevant data adds noise and can overwhelm valuable insights. For example, including demographic data in a model for product sales might not be relevant, unless it's shown to impact sales patterns.

Uniqueness: Data without unnecessary duplication.

- **Impact**: Redundant data can inflate model training time and lead to overfitting, as the same information is repeatedly learned by the algorithm.

How to Ensure High-Quality Data

Implement best practices to achieve high-quality data:

Data Cleaning

- **Missing Data**: Handle missing values through imputation, interpolation, or deletion (depending on the nature of the dataset).
- **Outliers**: Identify and handle outliers using statistical techniques or domain-specific knowledge.
- **Duplication**: Remove duplicate entries from datasets to avoid skewed analysis.

Data Standardization

- Ensure consistency in data formatting (e.g., date formats, numerical precision).
- Normalize values where necessary, especially in machine learning models that are sensitive to scale (e.g., neural networks, k-NN).

How to Ensure High-Quality Data

Data Integration

- When combining datasets from multiple sources, ensure they are consistent and harmonized to avoid discrepancies.

Data Validation

- Apply rules or algorithms to verify the accuracy and integrity of data.
- Regular audits of the data collection process help maintain accuracy and consistency.

Data Enrichment

- Enhance data by integrating external datasets to fill gaps or add more context. For example, augmenting demographic data with geographical or psychographic details.

Real-World Examples of Data Quality's Impact

- **Healthcare:** In healthcare, poor data quality (like incorrect patient data or incomplete medical records) can lead to **misdiagnoses** or ineffective treatments.

- **Finance:** In finance, inaccurate financial data or outdated market data can lead to poor investment decisions, **loss of capital**, or failure to comply with regulations.

- **E-Commerce:** For e-commerce, missing or incorrect customer information (like incorrect addresses or invalid payment methods) can result in **lost sales** or **customer dissatisfaction**.

- **Supply Chain:** In supply chain management, incomplete or outdated data regarding inventory levels or supplier lead times can cause **stockouts** or **delays** in product delivery.

Summary

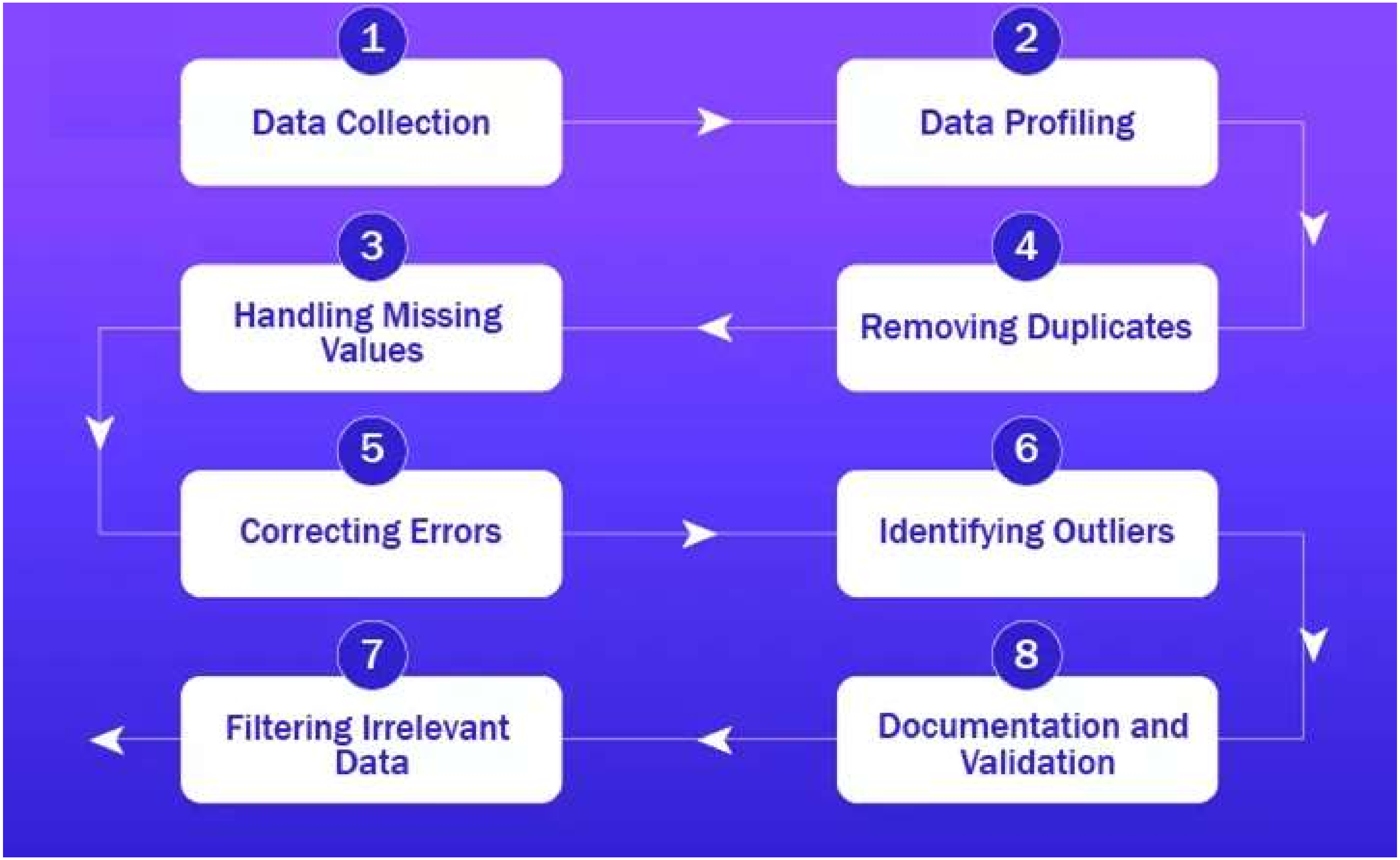
- **Data cleaning** is the process of fixing or removing incorrect, corrupted, in correctly formatted, duplicate, or incomplete data within a dataset.

- **Data quality directly influences** the **success and reliability** of any analysis, model, or decision-making process

- **Investing time in ensuring data quality** through thorough cleaning, validation, and continuous monitoring

- It leads to **more reliable outcomes**, whether in machine learning models, business intelligence, or scientific research

Summary - Data Cleaning Process




What is Data Cleaning in Data Science?
Handling Missing Values
Removing Duplicates
Correcting Errors
Identifying Outliers
Standardizing Formats
Removing Irrelevant Information

Example:

Based on various market surveys, the consulting firm has gathered a large dataset of different types of used cars across the market.

Data Dictionary:

1.Sales_ID (Sales ID)
2.name (Name of the used car)
3.year (Year of the car purchase)
4.selling_price (Current selling price for used car)
5.km_driven (Total km driven)
6.Region (Region where it is used)
7.State or Province (State or Province where it is used)
8.City (City where it is used)
9.fuel (Fuel type)
10.seller_type (Who is selling the car)
11.transmission (Transmission type of the car)
12.owner (Owner type)
13.mileage (Mileage of the car)
14.engine (engine power)
15.max_power (max power)
16.seats (Number of seats)
17.sold (used car sold or not)

https://www.kaggle.com/datasets/shubham1kumar/usedcar-data

Example data

	A	B	C	D	E	F	G	H	I	J	K
1	car_name	car_prices_in_rupee	kms_driven	fuel_type	transmissi	ownership	manufactu	engine	Seats	manufacture	
2	Jeep Compass 2.0 Longitude Option BSIV	10.03 Lakh	86,226 kms	Diesel	Manual	1st Owner	2017	1956 cc	5 Seats	January 7, 2018	
3	Renault Duster RXZ Turbo CVT	12.83 Lakh	13,248 kms	Petrol	Automatic	1st Owner	2021	1330 cc	5 Seats	January 15, 2018	
4	Toyota Camry 2.5 G	16.40 Lakh	60,343 kms	Petrol	Automatic	1st Owner	2016	2494 cc	5 Seats	August 1, 2018	
5	Honda Jazz VX CVT	7.77 Lakh	26,696 kms	Petrol	Automatic	1st Owner	2018	1199 cc	5 Seats	June 8, 2018	
6	Volkswagen Polo 1.2 MPI Highline	5.15 Lakh	69,414 kms	Petrol	Manual	1st Owner	2016	1199 cc	5 Seats	June 20, 2018	
7	Volkswagen Vento 1.2 TSI Highline AT	7.66 Lakh	49,719 kms	Petrol	Automatic	1st Owner	2017	1197 cc	5 Seats	March 26, 2017	
8	Volkswagen Vento 1.2 TSI Highline Plus AT	7.58 Lakh	43,688 kms	Petrol	Automatic	1st Owner	2017	1197 cc	5 Seats	January 7, 2018	
9	Honda WR-V VX Diesel	11.60 Lakh	14,470 kms	Diesel	Manual	1st Owner	2021	1498 cc	5 Seats	January 15, 2018	
10	Honda City i VTEC CVT SV	6.99 Lakh	21,429 kms	Petrol	Automatic	1st Owner	2015	1497 cc	5 Seats	August 1, 2018	
11	Renault Duster Petrol RXS CVT	7.53 Lakh	31,750 kms	Petrol	Automatic	1st Owner	2017	1498 cc	5 Seats	June 8, 2018	
12	Maruti Baleno 1.2 Alpha	6.43 Lakh	38,203 kms	Petrol	Manual	1st Owner	2017	1197 cc	5 Seats	June 20, 2018	
13	Honda City i VTEC CVT SV	5.43 Lakh	1,10,284 kms	Petrol	Automatic	1st Owner	2014	1497 cc	5 Seats	March 26, 2017	
14	Mahindra XUV300 W6	8.62 Lakh	10,381 kms	Petrol	Manual	1st Owner	2020	1197 cc	5 Seats	January 7, 2018	
15	Jeep Compass 1.4 Limited Plus BSIV	16.78 Lakh	32,378 kms	Petrol	Automatic	1st Owner	2019	1368 cc	Seats	January 15, 2018	
16	Honda City V MT	10.03 Lakh		Petrol	Manual	1st Owner	2020	1498 cc	5 Seats	August 1, 2018	
17	Hyundai Grand i10 AT Asta	5.63 Lakh	59,313 kms	Petrol	Automatic	2nd Owne	2016	1197 cc	5 Seats	June 8, 2018	

car_price

Problems in data

Missing value

Mixed data: (e.g. in 1st Col, car_name with company name, in 2nd col. Car_price amount with Lakh, in last Col. Date is in unstructured form.

- Steps for Data analytic and ML
- For this, necessary knowledge of:
  - Python and following powerful modules or libraries for data analysis and visualization:
    - Pandas (for data manipulation and cleaning)
    - Matplotlib (for general-purpose plotting)
    - Seaborn (builds on Matplotlib for advanced statistical visualizations)

Pandas

- This module is employed for data manipulation and analysis.
- Easy to work and it gives data structures like
  - Series (1D = a single column) ; *series = pd.Series()*
  - DataFrame (2D = a collection of columns provides merging, joining, and reshaping data); *df = pd.DataFrame(), where df stands for "DataFrame"*
  - handle large datasets.
- **General practice for**:
  - Cleaning, filtering, and transforming data.
  - Handling missing data and combining datasets.
  - Analyzing time series and statistics.
- **Example**: use it to read data from CSV files for cleaning/ analysis.

.csv file extension stands for "comma-separated value" file, and it's one of the most common outputs for any spreadsheet program.

https://flatfile.com/demo/

Example: Series (1D) and DataFrame (2D)

- **Series (1D)**

```
import pandas as pd
data = [10, 20, 30, 40]
series = pd.Series(data, index=['A', 'B', 'C', 'D'])
print(series)
```

Output

```
A    10
B    20
C    30
D    40
dtype: int64
```

- **DataFrame (2D)**

```
data = {
 "Name": ["Alice", "Bob", "Charlie"],
"Age": [25, 30, 35],    "Salary": [50000, 60000, 70000]
}
df = pd.DataFrame(data)
print(df)
```

Output

```
     Name     Age    Salary
0 Alice       25     50000
1 Bob         30      60000
2 Charlie     35      70000
```

Matplotlib

- A plotting module used for creating static, animated, and interactive visualizations

- **General practice for**:
  - Plotting line graph, histograms, bar charts, scatter plots, etc.
  - Modifying for interactive plots using titles, labels, legends, and other annotations.

- **Example**: use it for a given dataset to visualize trends over time, to create line charts or bar charts.

Seaborn

- A higher-level plotting interface builds on Matplotlib used for making attractive and informative statistical graphics by simplifying the complex visualizations.

- **General practice for**:
  - Making more sophisticated plots like heatmaps, violin plots (combining of box and density plots), pair plots, etc.
  - Adding statistical features like regression lines, correlation coefficients, and distributions.

- **Example**: use it for creating correlation heatmap or distribution of data.

seaborn.heatmap()
seaborn.violinplot()
seaborn.pairplot()

Real world sample employee salary dataset-1

Index	Empl_ID	Name	Depart	Age	Salary	Joining_Date
0	101	Alice	HR	25.0	50000.0	2020-01-15
1	102	Bob	IT	30.0	60000.0	2018-06-23
2	103	Charlie	Finance	NaN	70000.0	2017-08-19
3	104	David	IT	40.0	NaN	2015-09-10
4	105	Eve	HR	35.0	65000.0	2019-12-11
5	106	NaN	Finance	28.0	72000.0	2021-07-01
6	107	Grace	IT	NaN	55000.0	2016-05-14

Tasks perform in python

- Using dataset-1 perform following operations in python:
- **Loaded sample employee salary dataset**
- **Handled missing values** (Filled missing ages & salaries, removed missing names)
- **Filtered data** (Employees with salary > 60K, IT employees above 30)
- **Transformed data** (Added "Years of Experience", increased salary by 10%)
- **Merged datasets** (Added a Bonus column from another dataset)
- **Sorted & grouped data** (Sorted by salary, grouped by department)

Creating and displaying a sample employee dataset

```python
import pandas as pd
import numpy as np

# Creating a sample employee dataset
data = {
    "EmployeeID": [101, 102, 103, 104, 105, 106, 107],
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve", np.nan, "Grace"],
    "Department": ["HR", "IT", "Finance", "IT", "HR", "Finance", "IT"],
    "Age": [25, 30, np.nan, 40, 35, 28, np.nan],
    "Salary": [50000, 60000, 70000, np.nan, 65000, 72000, 55000],
    "Joining_Date": ["2020-01-15", "2018-06-23", "2017-08-19", "2015-09-10", "2019-12-11",
                     "2021-07-01", "2016-05-14"]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert Joining_Date to datetime
df["Joining_Date"] = pd.to_datetime(df["Joining_Date"])

# Display the dataset
print(df)
```

Load existing Sample Data

```python
import pandas as pd
# Load DataFrame from a CSV file
df =  pd.read_csv("path/to/your/folder/data.csv")
# Display the first 5 rows
print(df.head())
```

File Format	Method
CSV	pd.read_csv("file.csv")
Excel	pd.read_excel("file.xlsx")
JSON	pd.read_json("file.json")
Pickle	pd.read_pickle("file.pkl")
Multiple CS	Loop through files using os.listdir()

JSON:JavaScript Object Notation

```python
import pandas as pd
import numpy as np

# Creating a sample employee dataset
data = {
    "EmployeeID": [101, 102, 103, 104, 105, 106, 107],
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve", np.nan, "Grace"],
    "Department": ["HR", "IT", "Finance", "IT", "HR", "Finance", "IT"],
    "Age": [25, 30, np.nan, 40, 35, 28, np.nan],
    "Salary": [50000, 60000, 70000, np.nan, 65000, 72000, 55000],
    "Joining_Date": ["2020-01-15", "2018-06-23", "2017-08-19", "2015-09-10",
                     "2019-12-11", "2021-07-01", "2016-05-14"]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert Joining_Date to datetime
df["Joining_Date"] = pd.to_datetime(df["Joining_Date"])

# Display the dataset
print(df)
```

Console

Run

```
   EmployeeID     Name Department   Age   Salary Joining_Date
0         101    Alice         HR  25.0  50000.0   2020-01-15
1         102      Bob         IT  30.0  60000.0   2018-06-23
2         103  Charlie    Finance   NaN  70000.0   2017-08-19
3         104    David         IT  40.0      NaN   2015-09-10
4         105      Eve         HR  35.0  65000.0   2019-12-11
5         106      NaN    Finance  28.0  72000.0   2021-07-01
6         107    Grace         IT   NaN  55000.0   2016-05-14
```

Cleaning Data - Pandas

- **Removing Duplicates** `df.drop_duplicates(inplace=True)`

- **Renaming Columns**

  df.rename(columns={"OldColumn": "NewColumn"}, inplace=True)

- **Changing Data Types**

  df["Age"] = df["Age"].astype(int) # Convert to integer

  df["Date"] = pd.to_datetime(df["Date"]) # Convert to datetime*

- **Stripping Whitespace from Column Names**

  df.columns = df.columns.str.strip() #Remove spaces from column names or column values

*class datetime.date

An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Attributes: **year, month, and day**.

Handling Missing Data (NaN values)

- **Checking for Missing Values**
  - df.isnull().sum() # Count missing values per column

- **Removing Rows with Missing Data**
  - df.dropna(inplace=True) # Drop rows with NaN values

- **Filling Missing Values**
  - df.fillna(0, inplace=True) # Replace NaN with 0

  - df["Salary"].fillna(df["Salary"].mean(), inplace=True)
  - # Replace with column mean

Checking and filling missing values

```python
# Check missing values
print(df.isnull().sum())

# Fill missing 'Age' with the mean age
df["Age"].fillna(df["Age"].mean(), inplace=True)

# Fill missing 'Salary' with the median salary
df["Salary"].fillna(df["Salary"].median(), inplace=True)

# Drop rows where 'Name' is missing
df.dropna(subset=["Name"], inplace=True)

print(df)
```

Filtering Data

- **Filtering Rows Based on Condition**

```
df_filtered = df[df["Age"] > 30]  # Select rows where Age > 30
```

- **Filtering Multiple Conditions**

```
df_filtered = df[(df["Age"] > 30) & (df["Salary"] > 50000)]
```

- **Using .query() for Filtering**

```
df_filtered = df.query("Age > 30 and Salary > 50000")
```

Transforming Data

- **Transforming Data**

```
df["Salary"] = df["Salary"].apply(lambda x: x * 1.1)
# Increase salary by 10%
```

- **Creating a New Column**

```
df["Salary_After_Tax"] = df["Salary"] * 0.8
```

- **Replacing Values**

```
df["Department"] = df["Department"].replace({"HR":
"Human Resources", "IT": "Tech"})
```

In pandas -**apply()** - is a function that applies to each value in a column/row. **lambda x: x * 1.1** is a lambda function that **multiplies each value (x) by 1.1**, effectively increasing the salary by 10%.

Combining Datasets (Merging, Joining, and Concatenation)

- **Merging DataFrames on a Key (Like SQL JOIN*)**

```
df_merged = pd.merge(df1, df2, on="EmployeeID", how="inner")  # Inner join
df_merged = pd.merge(df1, df2, on="EmployeeID", how="left")   # Left join
df_merged = pd.merge(df1, df2, on="EmployeeID", how="outer")  # Outer join
```

*A **SQL JOIN** is used to combine rows from two or more tables based on a related column between them

Example

Employees Table

EmployeeID	Name	DepartmentID
101	Alice	1
102	Bob	2
103	Charlie	3
104	David	4

Departments Table

DepartmentID	DepartmentName
1	HR
2	IT
3	Finance

LEFT JOIN

Returns all records from the left table (Employees), and matching records from the right (Departments).
If no match is found, **NULL** is returned.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

EmployeeID	Name	DepartmentName
101	Alice	HR
102	Bob	IT
103	Charlie	Finance
104	David	NULL

Note that David is included, but with NULL in DepartmentName because no matching record exists in the Departments table.

INNER JOIN

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

Result

EmployeeID	Name	DepartmentName
101	Alice	HR
102	Bob	IT
103	Charlie	Finance

Note that David is missing because there's no matching DepartmentID = 4 in the Departments table.

RIGHT JOIN

Returns all records from the right table (Departments), and matching records from the left (Employees).

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

Result

EmployeeID	Name	DepartmentName
101	Alice	HR
102	Bob	IT
103	Charlie	Finance

FULL OUTER JOIN

Returns all records from both tables, with NULLs where there are no matches.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

Result

EmployeeID	Name	DepartmentName
101	Alice	HR
102	Bob	IT
103	Charlie	Finance
104	David	NULL
NULL	NULL	Sales

Note that David is included (no match in Departments) and "Sales" appears with **NULL** (Employees).

Combining Datasets (Merging, Joining, and Concatenation)

"Orders" Table

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

"Customers" Table

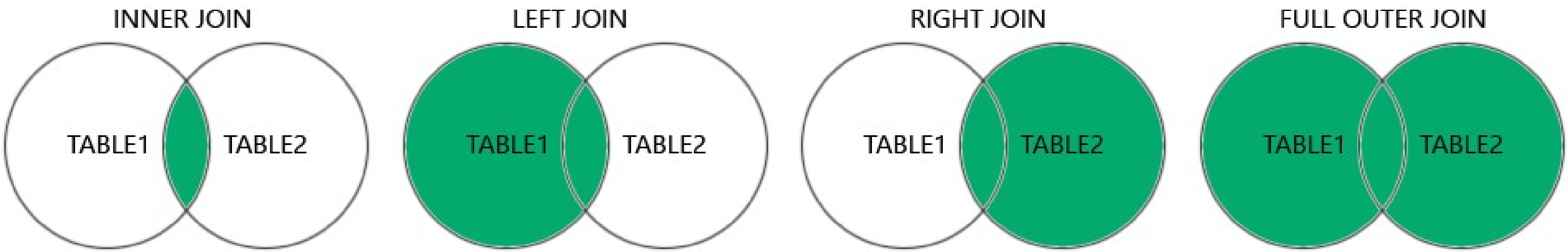
CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the **"CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table**. The relationship between the two tables above is the "CustomerID" column.

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Summary of Types of SQL JOINs

- **INNER JOIN** → Returns only matching records.
- **LEFT JOIN** (LEFT OUTER JOIN) → Returns all records from the **left table** and matching records from the right.
- **RIGHT JOIN** (RIGHT OUTER JOIN) → Returns all records from the **right table** and matching records from the left.
- **FULL JOIN** (FULL OUTER JOIN) → Returns all records from both tables (matching and non-matching).



Combining Datasets (Merging, Joining, and Concatenation)

- **Joining DataFrames on Index**

  df_joined = df1.join(df2.set_index("EmployeeID"), on="EmployeeID")

- **Concatenating DataFrames (Stacking)**

  df_combined = pd.concat([df1, df2], axis=0) # Stack rows

  df_combined = pd.concat([df1, df2], axis=1) # Merge side by side (columns)

Grouping and Aggregating Data

- **Grouping Data & Summarizing**

```
df_grouped = df.groupby("Department")["Salary"].mean()
# Mean salary per department
df_grouped = df.groupby("Department").agg({"Salary": "mean", "Age": "max"})
# Multiple aggregations
```

Sorting & Rearranging Data

- **Sorting Data**

```
df_sorted = df.sort_values("Salary", ascending=False)  # Sort by salary (descending)
```

- **Reset Index**

```
df.reset_index(drop=True, inplace=True)
```