Course: DS5002

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.

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.

Using Python- Advantages

- Syntax used is simple to understand code and reasonably fast to prototype
- Libraries designed for specific data science tasks
- Provides good ecosystem libraries that are robust and varied
- Links well with majority of the cloud platform service providers
- Tight-knit integration with big data frameworks such as Hadoop, Spark, etc.
- Supports both object oriented and functional programming paradigms
- Supports reading files from local, databases and cloud

Data Science using Python

- Python libraries provide key feature sets which essential for data science
- For this, necessary knowledge of:
  - **Python** and following powerful and basic 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)
    - NumPy (for numerical python)
  - **Machine learning** libraries like ‘Sci-kit learn’ or ‘Sklearn’ offer a bouquet of learning algorithms

Modules within a library e.g.,

```
Import numpy
content = dir (numpy)
print (content)
```

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 = [100, 200, 300, 400]
series = pd.Series(data, index=['A', 'B', 'C', 'D'])
print(series)
```

Output

```
A    100
B    200
C    300
D    400
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()

NumPy (Numerical Python)

- A powerful Python library used for numerical computing.
- Support to data structures such as:
  - Large, multi-dimensional arrays and matrices,
  - Mathematical functions (linear algebra, statistics, random number generation, etc.)

Installing:
pip install numpy

```
import numpy as np
# Creating a 1D array
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)
# Creating a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]]) print(arr2)
```

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")

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**.

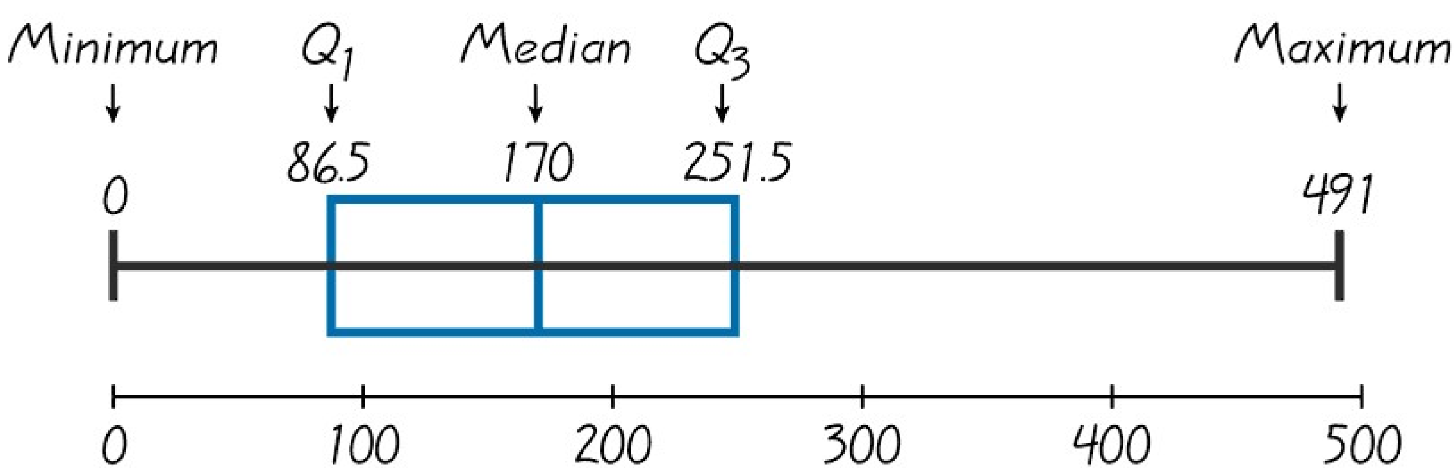
Handling Missing Data (NaN values)

- **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
  - print(df)

Boxplot (5-number statistic)

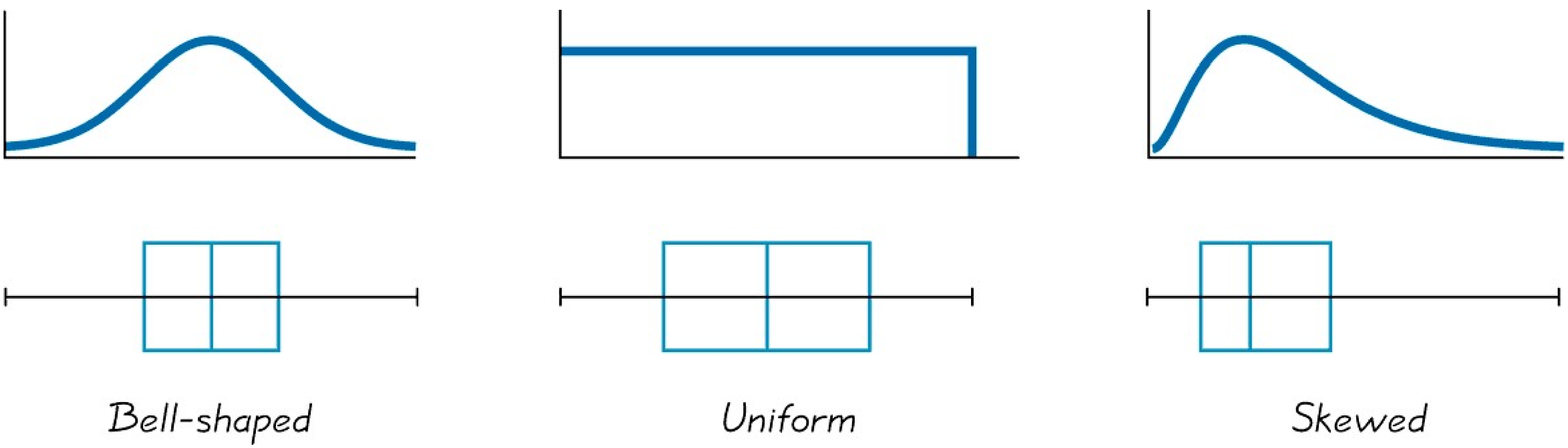
Box-and-whisker plot is a graphical representation of the distribution of a dataset

- **Minimum** (?) – The smallest data point, excluding outliers.
- **First Quartile (Q_1)** – 25th percentile (middle of lower half of data).
- **Median (Q_2)** – 50th percentile (middle value of the dataset).
- **Third Quartile (Q_3)** –75th percentile (middle of upper half of data).
- **Maximum** (?) – The largest data point, excluding outliers.



Boxplot (5-number statistic)

- **Skewness**: Median is closer to Q_1 or Q_3, data is skewed.
  - If the median is closer to Q_1, the distribution is **right-skewed** (longer tail on the right).
  - If the median is closer to Q_3, the distribution is **left-skewed** (longer tail on the left).
- **Spread of data**: A wider box means more variability in data.
- **Outliers**: Points beyond the whiskers suggest extreme values.



Boxplot (5-number statistic)

A box plot consists of:

- **A box** that represents the **interquartile range** (**IQR = Q_3 - Q_1**), which contains the middle 50% of the data.
- **A line inside the box** that shows the median (Q_2).
- **Whiskers extending from the box to the minimum and maximum values within 1.5 times the IQR**.
- **Outliers**, which are individual points outside the whiskers, marked as dots or small circles.

•**Lower Bound (Minimum)**: $Q_1 - 1.5 \times IQR$
Any data point **below** this bound is considered an **outlier**.

•**Upper Bound (Maximum): $Q_3 + 1.5 \times IQR$**
Any data point **above** this bound is also considered an **outlier**.

Identify Outliers

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Sample data
data = {
   'salary': [50000, 60000, 65000, 70000, 75000, 80000, 85000, 90000, 120000, 200000, 250000, 300000, 350000]
}
# Create a DataFrame
df = pd.DataFrame(data)
# Step 1: Calculate Q1, Q3, and IQR
Q1 = df['salary'].quantile(0.25)
Q3 = df['salary'].quantile(0.75)
IQR = Q3 - Q1
# Step 2: Calculate the outlier thresholds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
# Step 3: Identify outliers
outliers = df[(df['salary'] < lower_bound) | (df['salary'] > upper_bound)]
# Step 4: Visualize using a box plot
plt.figure(figsize=(8,6))
sns.boxplot(x=df['salary'])
plt.title('Box Plot of Salaries')
plt.show()
# Display outliers
print("Outliers:")
print(outliers)
```

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") # filter rows where Age is greater than 30 and Salary is greater than 5000

- **print(filtered_df)**

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"})

  # Replacing Islamabad' with ‘Rawalpindi‘

  df["City"] = df["City"].replace(" Islamabad ", " Rawalpindi ")

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 =df1

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

Departments Table =df2

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.

inner_merge = pd.merge(df1, df2, on='DepartmentID', how='left')

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

inner_merge = pd.merge(df1, df2, on='DepartmentID', how='inner')

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).

inner_merge = pd.merge(df1, df2, on=‘DepartmentID', how=‘right')

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.

full_outer_merge = df1.merge(df2, on='DepartmentID', how='outer').merge(df3, on='DepartmentID', how='outer')

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

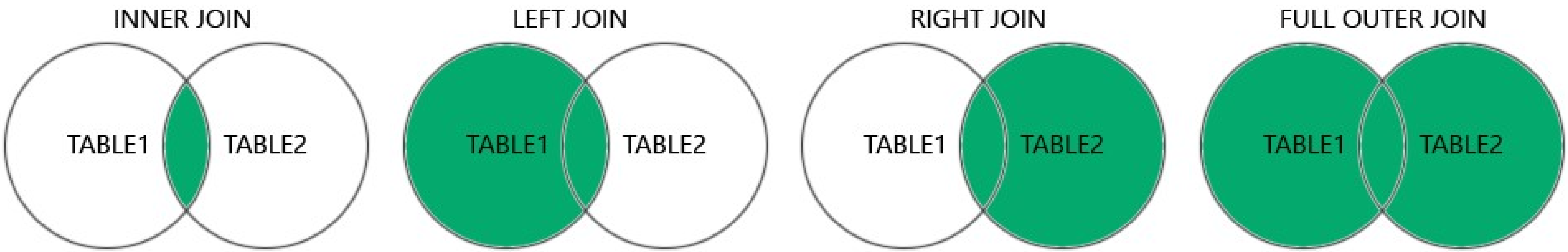
CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

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)

Practice

1. **Load datasets** using pandas.

2. **Merge the first two datasets** on the Department_ID column.

3. **Filter the merged dataset** to show only employees who earn a salary greater than some specific value X.

4. **Join** the merged dataset with a **third dataset** (managers.csv) that contains Manager_ID, Manager_Name, and Manager_Age.

5. **Concatenate** the resulting dataset with a new dataset (office_locations.csv) that contains Department_ID, Office_Location, and City, **showing the office locations** for each department.

- Provide the Python code to perform these tasks using pandas.

```python
import pandas as pd
# Load datasets
    employees = pd.read_csv('employees.csv')
    departments = pd.read_csv('departments.csv')
    managers = pd.read_csv('managers.csv')
    office_locations = pd.read_csv('office_locations.csv')
# 1. Merge employees with departments on 'Department_ID'
    merged_data = pd.merge(employees, departments, on='Department_ID')
# 2. Filter employees with salary > 60,000
    filtered_data = merged_data[merged_data['Salary'] > 60000]
# 3. Join the filtered dataset with managers on 'Manager_ID'
    final_data = pd.merge(filtered_data, managers, on='Manager_ID')
# 4. Concatenate with office locations on 'Department_ID'
    final_dataset = pd.merge(final_data, office_locations, on='Department_ID')
# Display the final result
print(final_dataset)
```

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)

MultiIndex

```
In [11]: data = pd.Series(np.random.uniform(size=9),
   ....:                  index=[["a", "a", "a", "b", "b", "c", "c", "d", "d"],
   ....:                         [1, 2, 3, 1, 3, 1
```

```
In [12]: data
Out[12]:
a  1    0.929616
   2    0.316376
   3    0.183919
b  1    0.204560
   3    0.567725
c  1    0.595545
   2    0.964515
d  2    0.653177
   3    0.748907
dtype: float64
```

```
In [13]: data.index
Out[13]:
MultiIndex([('a', 1),
            ('a', 2),
            ('a', 3),
            ('b', 1),
            ('b', 3),
            ('c', 1),
            ('c', 2),
            ('d', 2),
            ('d', 3)],
           )
```

```
In [14]: data["b"]
Out[14]:
1    0.204560
3    0.567725
dtype: float64
```

```
In [15]: data["b":"c"]
Out[15]:
b  1    0.204560
   3    0.567725
c  1    0.595545
   2    0.964515
dtype: float64
```

```
In [16]: data.loc[["b", "d"]]
Out[16]:
b  1    0.204560
   3    0.567725
d  2    0.653177
   3    0.748907
dtype: float64
```

National University of Computer and Emerging Sciences
Islamabad Campus

Data Science Tools and Techniques (DS5002)

Sessional-I Exam

Total Time (Hrs):	**1**
Total Marks:	**50**
Total Questions:	**4**

Course Instructor(s):
Dr. Safdar Ali
Section(s): (if applicable)

Date: Feb 25, 2025

______________ ______________ __________________________
Roll No **Course Section** **Student Signature**

Do not write below this line.

Attempt all the questions.

[CLO 1: Demonstrate the basic concepts of programming]

Q1: Question statement ………… **[10 marks]**

[CLO 2: Apply algorithmic solutions related to the degree program to recent related problems]

Q2: Question statement ………… **[10 marks]**

[CLO 1: Demonstrate the basic concepts of programming]

Q3: Question statement ………… **[10 marks]**