SQL DATA ANALYSIS

VS

PYTHON DATA ANALYSIS

What is SQL?

SQL (Structured Query Language) is a standardized language used to store, retrieve, and manipulate data in relational databases like MySQL, PostgreSQL, SQL Server, and SQLite.

Key Uses in Data Analysis:

- Extracting specific data using SELECT, WHERE, and JOIN
- Summarizing data with GROUP BY, COUNT, AVG, etc.
- Filtering and sorting records from large datasets.

What is Python?

Python is a general-purpose, high-level programming language known for its simplicity and powerful libraries used in data science, machine learning, and automation.

Key Uses in Data Analysis:

- Reading and cleaning datasets (using Pandas)
- Performing statistical calculations (using NumPy)
- Creating charts and graphs (using Matplotlib, Seaborn)
- Building machine learning models (using Scikit-learn, TensorFlow)

SQL vs Python for Data Analysis :

Feature/Aspect	SQL (Structured Query Language)	Python (Programming Language)
1. Primary Use	Querying and managing data in relational databases	Data manipulation, analysis, visualization, automation
2. Data Storage	Works directly with databases (MySQL, PostgreSQL, SQLite)	Imports data from files or databases for in-memory analysis
3. Type of Language	Declarative (describe *what* you want)	Procedural/Imperative (describe *how* to do it)
4. Output Type	Tabular data (rows and columns)	Tables, charts, statistical outputs, models
5. Key Libraries	Native SQL language	Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn
6. Best At	Data extraction, filtering, joining, summarization	Data cleaning, transformation, advanced analysis, ML
7. Loops & Logic	Limited support	Full support for conditions, loops, and functions
8. Visualization	Not supported	Strong support (Matplotlib, Seaborn, Plotly, etc.)
9. Machine Learning	Not possible directly in SQL	Fully supported via libraries (e.g., Scikit-learn, TensorFlow)
10. Learning Curve	Easier for beginners (simple queries)	Steeper learning curve but more flexible

SQL Concepts Used :

- SELECT (choose columns)
- FROM (choose table)
- WHERE (filter rows)
- LIMIT (restrict number of rows)
- DISTINCT (unique values)
- ORDER BY (sorting)
- AS (aliasing columns or tables)
- GROUP BY (grouping rows)
- Aggregate functions: AVG(), COUNT(), SUM(), MIN(), MAX()
- HAVING (filter groups after aggregation)
- LIKE (pattern matching)
- BETWEEN (range filtering)
- IN (multiple value filtering)

Python Concepts Used :

- DataFrame size and shape: len(), .shape
- Column selection: df[['col1', 'col2']], .unique()
- Row selection / filtering: Boolean indexing with .str.startswith(), &, .isin()
- Value assignment: df['col'] = value
- Sorting: .sort_values()
- Renaming: .rename()

- Grouping and aggregation: .groupby(), .mean(), .sum(), .min(), .max(), .size(), .nunique()
- Filtering grouped data: .filter()
- DataFrame/Series transformation: .to_frame(), .reset_index()

COUPON RECOMMENDATION ANALYSIS

-:SQL:-

➢ "Select all columns and all rows from the table named dataset_1."

```
select * from dataset_1;
```

dataset_1 1

select * from dataset_1 | Enter a SQL expression to filter results (use Ctrl+Space)

	destination	passanger	weather	temperature	time
1	No Urgent Place	Alone	Sunny	55	2PM
2	No Urgent Place	Friend(s)	Sunny	80	10AM
3	No Urgent Place	Friend(s)	Sunny	80	10AM
4	No Urgent Place	Friend(s)	Sunny	80	2PM
5	No Urgent Place	Friend(s)	Sunny	80	2PM
6	No Urgent Place	Friend(s)	Sunny	80	6PM
7	No Urgent Place	Friend(s)	Sunny	55	2PM
8	No Urgent Place	Kid(s)	Sunny	80	10AM
9	No Urgent Place	Kid(s)	Sunny	80	10AM
10	No Urgent Place	Kid(s)	Sunny	80	10AM
11	No Urgent Place	Kid(s)	Sunny	80	2PM
12	No Urgent Place	Kid(s)	Sunny	55	2PM
13	No Urgent Place	Kid(s)	Sunny	55	6PM
14	Home	Alone	Sunny	55	6PM
15	Home	Alone	Sunny	55	6PM
16	Home	Alone	Sunny	80	6PM

-: PYTHON :-

➢ Importing a dataset from a CSV file into Python (as a DataFrame) so you can perform data analysis using pandas, similar to how you would in SQL.

```
import pandas as pd
```

sql=pd.read_csv(r"C:\ …….\data.csv")

sql

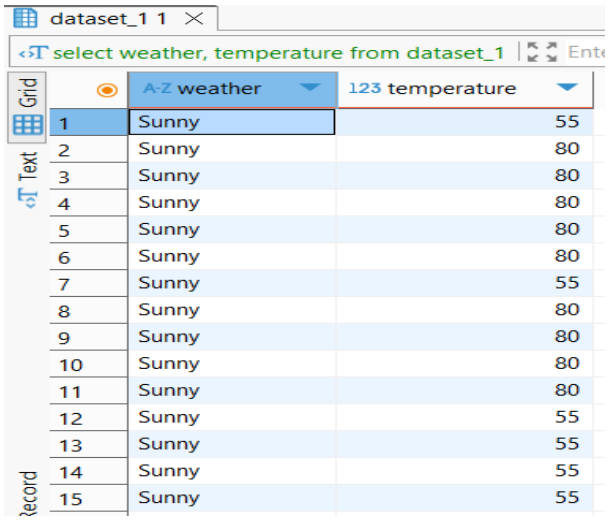
	destination	passanger	weather	temperature
0	No Urgent Place	Alone	Sunny	55
1	No Urgent Place	Friend(s)	Sunny	80
2	No Urgent Place	Friend(s)	Sunny	80
3	No Urgent Place	Friend(s)	Sunny	80
4	No Urgent Place	Friend(s)	Sunny	80
...	...	...	...	...
12679	Home	Partner	Rainy	55
12680	Work	Alone	Rainy	55
12681	Work	Alone	Snowy	30
12682	Work	Alone	Snowy	30
12683	Work	Alone	Sunny	80

12684 rows × 27 columns

-:SQL:-

- "Fetch the columns weather and temperature from all the rows in the table named dataset_1."

select weather, temperature from dataset_1;



-: PYTHON :-

- Select the columns weather and temperature from the DataFrame df.

```
df=sql
df[['weather','temperature']]
```

	weather	temperature
0	Sunny	55
1	Sunny	80
2	Sunny	80
3	Sunny	80
4	Sunny	80
...	...	...
12679	Rainy	55
12680	Rainy	55
12681	Snowy	30
12682	Snowy	30
12683	Sunny	80

12684 rows × 2 columns

-:SQL:-

- "Select all columns from the first 10 rows of the table dataset_1."

```
select * from dataset_1 limit 10;
```

	A-Z destination	A-Z passanger	A-Z weather
1	No Urgent Place	Alone	Sunny
2	No Urgent Place	Friend(s)	Sunny
3	No Urgent Place	Friend(s)	Sunny
4	No Urgent Place	Friend(s)	Sunny
5	No Urgent Place	Friend(s)	Sunny
6	No Urgent Place	Friend(s)	Sunny
7	No Urgent Place	Friend(s)	Sunny
8	No Urgent Place	Kid(s)	Sunny
9	No Urgent Place	Kid(s)	Sunny
10	No Urgent Place	Kid(s)	Sunny

-: PYTHON :-

- "Display the first 10 rows of the DataFrame df."

```
df.head(10)
```

	destination	passanger	weather	temperature	time
0	No Urgent Place	Alone	Sunny	55	2PM
1	No Urgent Place	Friend(s)	Sunny	80	10AM
2	No Urgent Place	Friend(s)	Sunny	80	10AM
3	No Urgent Place	Friend(s)	Sunny	80	2PM
4	No Urgent Place	Friend(s)	Sunny	80	2PM
5	No Urgent Place	Friend(s)	Sunny	80	6PM
6	No Urgent Place	Friend(s)	Sunny	55	2PM
7	No Urgent Place	Kid(s)	Sunny	80	10AM
8	No Urgent Place	Kid(s)	Sunny	80	10AM
9	No Urgent Place	Kid(s)	Sunny	80	10AM

10 rows × 27 columns

-:SQL:-

- "Return all unique (non-repeating) values from the passanger column in the dataset_1 table."

```
select distinct passanger from dataset_1;
```

	A-Z passanger
1	Alone
2	Friend(s)
3	Kid(s)
4	Partner

-: PYTHON :-

- "Return an array of all unique (non-repeating) values in the passanger column of the DataFrame df."

```
df.passanger.unique()
```

```
array(['Alone', 'Friend(s)', 'Kid(s)', 'Partner'], dtype=object)
```

-:SQL:-

➢ "Retrieve all rows from the table dataset_1 where the value in the destination column is exactly 'Home'."

```
select * from dataset_1 where destination='Home';
```

	A-Z destination	A-Z passanger	A-Z weather
1	Home	Alone	Sunny
2	Home	Alone	Sunny
3	Home	Alone	Sunny
4	Home	Alone	Sunny
5	Home	Alone	Sunny
6	Home	Alone	Sunny
7	Home	Alone	Sunny
8	Home	Alone	Sunny
9	Home	Alone	Sunny
10	Home	Alone	Sunny
11	Home	Alone	Sunny
12	Home	Alone	Sunny
13	Home	Alone	Sunny
14	Home	Alone	Sunny
15	Home	Alone	Sunny
16	Home	Alone	Sunny

-: PYTHON :-

➢ This shows only the rows where the destination is already 'Home', without changing the data.

```
df.destination="Home"
df
```

	Destination	passanger	weather	temperature
0	Home	Alone	Sunny	55
1	Home	Friend(s)	Sunny	80
2	Home	Friend(s)	Sunny	80
3	Home	Friend(s)	Sunny	80
4	Home	Friend(s)	Sunny	80
...	...	...	...	...
12679	Home	Partner	Rainy	55
12680	Home	Alone	Rainy	55
12681	Home	Alone	Snowy	30
12682	Home	Alone	Snowy	30
12683	Home	Alone	Sunny	80

12684 rows × 27 columns

-:SQL:-

- "Select all rows and columns from dataset_1 and sort the result by the coupon column in ascending order."

```
select * from dataset_1 order by coupon;
```

	A-Z destination	A-Z passanger	A-Z weather	123 temperature	A-Z time	A-Z coupon
1	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar
2	Home	Alone	Sunny	55	6PM	Bar
3	Work	Alone	Sunny	55	7AM	Bar
4	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar
5	Home	Alone	Sunny	55	6PM	Bar
6	Work	Alone	Sunny	55	7AM	Bar
7	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar
8	Home	Alone	Sunny	55	6PM	Bar
9	Work	Alone	Sunny	55	7AM	Bar
10	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar
11	Home	Alone	Sunny	55	6PM	Bar
12	Work	Alone	Sunny	55	7AM	Bar
13	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar
14	Home	Alone	Sunny	55	6PM	Bar
15	Work	Alone	Sunny	55	7AM	Bar
16	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar

-: PYTHON :-

- "Sort the entire DataFrame df by the values in the coupon column in ascending order."

```
df.sort_values("coupon")
```

	destination	passanger	weather	temperature	time	coupon	expi
11702	Home	Partner	Sunny	30	10PM	Bar	
9930	Home	Alone	Snowy	30	2PM	Bar	
10632	Home	Alone	Rainy	55	6PM	Bar	
7997	Home	Friend(s)	Rainy	55	10PM	Bar	
11166	Home	Alone	Snowy	30	7AM	Bar	
...	...	...	...	...	...	...	
10476	Home	Alone	Sunny	80	6PM	Restaurant(<20)	
5447	Home	Alone	Sunny	80	10PM	Restaurant(<20)	
10478	Home	Alone	Snowy	30	10PM	Restaurant(<20)	
5440	Home	Alone	Sunny	80	2PM	Restaurant(<20)	
0	Home	Alone	Sunny	55	2PM	Restaurant(<20)	

12684 rows × 27 columns

-:SQL:-

- "Select the destination column from the table dataset_1, and rename (alias) it as Destination."

```
select destination as Destination from dataset_1 d ;
```

	A-Z Destination
1	No Urgent Place
2	No Urgent Place
3	No Urgent Place
4	No Urgent Place
5	No Urgent Place
6	No Urgent Place
7	No Urgent Place
8	No Urgent Place
9	No Urgent Place
10	No Urgent Place
11	No Urgent Place
12	No Urgent Place
13	No Urgent Place
14	Home
15	Home
16	Home
17	Work

-: PYTHON :-

- Renames the column destination → Destination (with a capital D).
- The change is permanent in the DataFrame df because you used inplace=True.

```
df.rename(columns={'destination':'Destination'},inplace=True)
```

	Destination
0	Home
1	Home
2	Home
3	Home
4	Home
...	...
12679	Home
12680	Home
12681	Home
12682	Home
12683	Home

-:SQL:-

- "Return the unique values of the occupation column from the dataset_1 table."

```
select occupation from dataset_1 group by  occupation;
```

	A-Z occupation
1	Architecture & Engineering
2	Arts Design Entertainment Sports & Media
3	Building & Grounds Cleaning & Maintenance
4	Business & Financial
5	Community & Social Services
6	Computer & Mathematical
7	Construction & Extraction
8	Education&Training&Library
9	Farming Fishing & Forestry
10	Food Preparation & Serving Related
11	Healthcare Practitioners & Technical
12	Healthcare Support
13	Installation Maintenance & Repair
14	Legal
15	Life Physical Social Science
16	Management

-: PYTHON :-

- "Group the DataFrame by the occupation column, count the number of rows in each group, and return a DataFrame with occupation and the corresponding counts in a column named Count."

```
df.groupby('occupation').size().to_frame('Count').reset_index()
```

	occupation
0	Architecture & Engineering
1	Arts Design Entertainment Sports & Media
2	Building & Grounds Cleaning & Maintenance
3	Business & Financial
4	Community & Social Services
5	Computer & Mathematical
6	Construction & Extraction
7	Education&Training&Library
8	Farming Fishing & Forestry
9	Food Preparation & Serving Related
10	Healthcare Practitioners & Technical
11	Healthcare Support
12	Installation Maintenance & Repair
13	Legal
14	Life Physical Social Science
15	Management

-:SQL:-

- "For each unique weather type in the table dataset_1, calculate the average (AVG) temperature, and return the weather type along with its average temperature, aliased as avg_temp."

```
select weather ,AVG(temperature) as avg_temp from dataset_1
group by weather;
```

	A-Z weather	123 avg_temp
1	Rainy	55
2	Snowy	30
3	Sunny	68.9462707319

-: PYTHON :-

- "Group the DataFrame df by the weather column, calculate the average (mean) temperature for each weather group, then convert the result to a DataFrame with a column named avg_temp, and finally reset the index so that weather becomes a regular column."

```
df.groupby('weather')['temperature'].mean().to_frame('avg_temp').reset_index()
```

	weather	avg_temp
0	Rainy	55.000000
1	Snowy	30.000000
2	Sunny	68.946271

-:SQL:-

- "For each unique weather condition in dataset_1, count the number of temperature entries and label that count as count_temp."

```
select weather, COUNT(temperature) as count_temp from dataset_1 group by weather;
```

	A-Z weather	123 count_temp
1	Rainy	1,210
2	Snowy	1,405
3	Sunny	10,069

-: PYTHON :-

- "Group the DataFrame df by the weather column, count the number of rows in each group (including those with NaN values), convert the result to a DataFrame with the column name Count_temp, and reset the index so weather becomes a column."

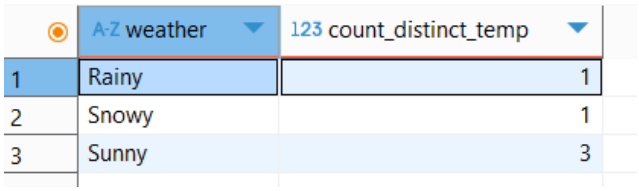
```
df.groupby('weather')['temperature'].size().to_frame('Count_temp').reset_index()
```

	weather	Count_temp
0	Rainy	1210
1	Snowy	1405
2	Sunny	10069

-:SQL:-

- "For each unique weather condition in dataset_1, count the number of distinct (unique) temperature values and name that count as count_distinct_temp."

```
select weather ,COUNT(DISTINCT temperature) as count_distinct_temp from dataset_1 group by weather ;
```



	A-Z weather	123 count_distinct_temp
1	Rainy	1
2	Snowy	1
3	Sunny	3

-: PYTHON :-

- "Group the DataFrame by weather, count the number of unique values in the temperature column for each weather group, convert the result to a DataFrame with the column name count_distinct_temp, and reset the index to make weather a column."

```
df.groupby('weather')['temperature'].nunique().to_frame('count_distinct_temp').reset_index()
```

	weather	count_distinct_temp
0	Rainy	1
1	Snowy	1
2	Sunny	3

-:SQL:-

- "For each unique weather condition in dataset_1, calculate the total (sum) of the temperature values, and label this total as sum_temp."

```
select weather ,SUM(temperature) as sum_temp from dataset_1 group by weather;
```

	A-Z weather	123 sum_temp
1	Rainy	66,550
2	Snowy	42,150
3	Sunny	694,220

-: PYTHON :-

- "Group the DataFrame df by the weather column, calculate the sum of temperature values for each weather group, convert the result into a DataFrame with the column name sum_temp, and reset the index so weather becomes a column."

```
df.groupby('weather')['temperature'].sum().to_frame('sum_temp').reset_index()
```

	weather	sum_temp
0	Rainy	66550
1	Snowy	42150
2	Sunny	694220

-:SQL:-

- "For each unique weather condition in dataset_1, find the minimum temperature and label it as min_temp."

```
select weather ,MIN(temperature) as min_temp from dataset_1 group by weather;
```

	A-Z weather	123 min_temp
1	Rainy	55
2	Snowy	30
3	Sunny	30

-: PYTHON :-

- "Group the DataFrame df by the weather column, find the minimum temperature for each weather group, convert the result to a DataFrame with column name min_temp, and reset the index to make weather a regular column."

df.groupby('weather')['temperature'].min().to_frame('min_temp').reset_index()

	weather	min_temp
0	Rainy	55
1	Snowy	30
2	Sunny	30

-:SQL:-

- "For each unique weather condition in dataset_1, find the maximum temperature and label it as max_temp."

select weather ,MAX(temperature) as *max_temp* from dataset_1 group by weather;

	A-Z weather	123 max_temp
1	Rainy	55
2	Snowy	30
3	Sunny	80

-: PYTHON :-

- "Group the DataFrame df by the weather column, find the maximum temperature for each weather group, convert the result into a DataFrame with the column name max_temp, and reset the index so weather becomes a column."

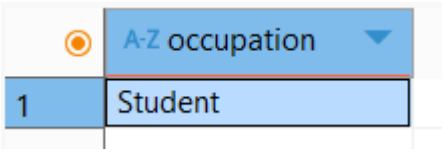
df.groupby('weather')['temperature'].max().to_frame('max_temp').reset_index()

	weather	max_temp
0	Rainy	55
1	Snowy	30
2	Sunny	80

-:SQL:-

- "Group the data by occupation and return only the group where the occupation is 'Student'."

select occupation from dataset_1 group by occupation having occupation='Student';



	A-Z occupation
1	Student

-: PYTHON :-

- "Group the DataFrame df by occupation, then filter to keep only the group where the occupation is 'Student'. Finally, count the number of rows in that group."

df.groupby('occupation').filter(lambda x: x['occupation'].iloc[0] == 'Student').groupby('occupation').size()

```
occupation
Student    1584
```

-:SQL:-

- "Select all rows from dataset_1 where the weather column starts with 'Sun'."

select * from dataset_1 where weather like 'Sun%';

	A-Z destination	A-Z passanger	A-Z weather
1	No Urgent Place	Alone	Sunny
2	No Urgent Place	Friend(s)	Sunny
3	No Urgent Place	Friend(s)	Sunny
4	No Urgent Place	Friend(s)	Sunny
5	No Urgent Place	Friend(s)	Sunny
6	No Urgent Place	Friend(s)	Sunny
7	No Urgent Place	Friend(s)	Sunny
8	No Urgent Place	Kid(s)	Sunny
9	No Urgent Place	Kid(s)	Sunny
10	No Urgent Place	Kid(s)	Sunny
11	No Urgent Place	Kid(s)	Sunny
12	No Urgent Place	Kid(s)	Sunny
13	No Urgent Place	Kid(s)	Sunny
14	Home	Alone	Sunny
15	Home	Alone	Sunny
16	Home	Alone	Sunny

-: PYTHON :-

- "Filter the DataFrame df to return only the rows where the weather column starts with 'Sun'."

df[df['weather'].str.startswith('Sun')]

	Destination	passanger	weather	temperature
0	Home	Alone	Sunny	55
1	Home	Friend(s)	Sunny	80
2	Home	Friend(s)	Sunny	80
3	Home	Friend(s)	Sunny	80
4	Home	Friend(s)	Sunny	80
...	...	...	...	...
12673	Home	Alone	Sunny	30
12676	Home	Alone	Sunny	80
12677	Home	Partner	Sunny	30
12678	Home	Partner	Sunny	30
12683	Home	Alone	Sunny	80

10069 rows × 27 columns

-:SQL:-

- "Select all unique (DISTINCT) temperature values from dataset_1 where the temperature is between 29 and 75 (inclusive)."

select distinct temperature from dataset_1 where temperature between 29 and 75;

	123 temperature
1	55
2	30

-: PYTHON :-

- "From the DataFrame df, filter rows where temperature is between 29 and 75 (inclusive), then return only the unique temperature values."

```
df[(df['temperature'] >= 29) & (df['temperature'] <=75)]['temperature'].unique()
```

```
array([55, 30],
```

-:SQL:-

- "Select the occupation column from dataset_1 where the occupation is either 'Sales & Related' or 'Management'. "

```
select occupation from dataset_1 where occupation in('Sales & Related','Management');
```

	A-Z occupation
1	Sales & Related
2	Sales & Related
3	Sales & Related
4	Sales & Related
5	Sales & Related
6	Sales & Related
7	Sales & Related
8	Sales & Related
9	Sales & Related
10	Sales & Related
11	Sales & Related
12	Sales & Related
13	Sales & Related
14	Sales & Related
15	Sales & Related
16	Sales & Related

-: PYTHON :-

- "Filter the DataFrame df to return only the rows where the occupation is either 'Sales & Related' or 'Management', and show only the occupation column."

```
df[df['occupation'].isin(['Sales & Related','Management'])][['occupation']]
```

	occupation
193	Sales & Related
194	Sales & Related
195	Sales & Related
196	Sales & Related
197	Sales & Related
...	...
12679	Sales & Related
12680	Sales & Related
12681	Sales & Related
12682	Sales & Related
12683	Sales & Related

1931 rows × 1 columns