

Day 33 – Data Analysis: SQL vs Python

In this notebook, I compared **SQL** and **Python (with Pandas)** for performing common data analysis tasks.

Both are powerful tools:

- **SQL** is great for working directly with structured data stored in relational databases.
- **Python** (especially with Pandas) excels in flexible in-memory data manipulation, advanced analytics, and integration with visualization libraries.

I have performed the same analysis using both approaches to understand their strengths and limitations.

```
In [1]: import pandas as pd
```

```
In [2]: # Load dataset
df = pd.read_csv(r'C:\Users\Arman\Downloads\dataset\dataset_1.csv')
```

Load Dataset

SQL Query:

```
select * from dataset_1;
```

Python Code:

```
In [3]: df
```

Out[3]:

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	n
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)		1d	Female	21
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House		2h	Female	21
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away		2h	Female	21
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House		2h	Female	21
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House		1d	Female	21
...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away		1d	Male	26
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away		1d	Male	26
12681	Work	Alone	Snowy	30	7AM	Coffee House		1d	Male	26
12682	Work	Alone	Snowy	30	7AM	Bar		1d	Male	26
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)		2h	Male	26

12684 rows × 27 columns



Display Specific Columns

SQL Query:

```
select weather, temperature from dataset_1 d ;
```

Python Code:

In [4]: df[['weather', 'temperature']]

Out[4]:

	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

View First 10 Rows

SQL Query:

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

Python Code:

In [5]: df.head(10)

Out[5]:

	destination	passanger	weather	temperature	time		coupon	expiration	gender	age	marit
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Un	
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	Un	
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Un	
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	Un	
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	Un	
5	No Urgent Place	Friend(s)	Sunny	80	6PM	Restaurant(<20)	2h	Female	21	Un	
6	No Urgent Place	Friend(s)	Sunny	55	2PM	Carry out & Take away	1d	Female	21	Un	
7	No Urgent Place	Kid(s)	Sunny	80	10AM	Restaurant(<20)	2h	Female	21	Un	
8	No Urgent Place	Kid(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Un	
9	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar	1d	Female	21	Un	

10 rows × 27 columns



Unique values in a column

SQL Query:

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

Python Code:

In [6]: `df['passanger'].unique()`

Out[6]: `array(['Alone', 'Friend(s)', 'Kid(s)', 'Partner'], dtype=object)`

Filter rows based on condition

SQL Query:

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

Python Code:

In [7]: `df[df['destination']=='Home']`

Out[7]:

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	n
13	Home	Alone	Sunny	55	6PM	Bar	1d	Female	21	
14	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1d	Female	21	
15	Home	Alone	Sunny	80	6PM	Coffee House	2h	Female	21	
35	Home	Alone	Sunny	55	6PM	Bar	1d	Male	21	
36	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1d	Male	21	
...
12675	Home	Alone	Snowy	30	10PM	Coffee House	2h	Male	26	
12676	Home	Alone	Sunny	80	6PM	Restaurant(20-50)	1d	Male	26	
12677	Home	Partner	Sunny	30	6PM	Restaurant(<20)	1d	Male	26	
12678	Home	Partner	Sunny	30	10PM	Restaurant(<20)	2h	Male	26	
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	

3237 rows × 27 columns

Order by a column

SQL Query:

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

Python Code:

In [8]: `df.sort_values('coupon')`

Out[8]:

	destination	passanger	weather	temperature	time		coupon	expiration	gender	age
11702	Home	Partner	Sunny	30	10PM		Bar	2h	Female	50plus
9930	No Urgent Place	Alone	Snowy	30	2PM		Bar	1d	Female	21
10632	Home	Alone	Rainy	55	6PM		Bar	1d	Male	21
7997	No Urgent Place	Friend(s)	Rainy	55	10PM		Bar	2h	Male	26
11166	Work	Alone	Snowy	30	7AM		Bar	1d	Female	41
...
10476	Home	Alone	Sunny	80	6PM	Restaurant(<20)		1d	Female	31
5447	Home	Alone	Sunny	80	10PM	Restaurant(<20)		2h	Female	50plus
10478	Home	Alone	Snowy	30	10PM	Restaurant(<20)		2h	Female	31
5440	No Urgent Place	Alone	Sunny	80	2PM	Restaurant(<20)		2h	Female	50plus
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)		1d	Female	21

12684 rows × 27 columns

Rename a Column

SQL Query:

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

Python Code:

In [9]: `df.rename(columns={'destination':'Destination'},inplace=True)`

In [10]: `df`

Out[10]:

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	r
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)		1d	Female	21
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House		2h	Female	21
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away		2h	Female	21
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House		2h	Female	21
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House		1d	Female	21
...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away		1d	Male	26
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away		1d	Male	26
12681	Work	Alone	Snowy	30	7AM	Coffee House		1d	Male	26
12682	Work	Alone	Snowy	30	7AM	Bar		1d	Male	26
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)		2h	Male	26

12684 rows × 27 columns



Group By with Count of Rows

SQL Query:

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

Python Code:

In [11]: df.groupby('occupation').size().to_frame('Count').reset_index()

Out[11]:

	occupation	Count
0	Architecture & Engineering	175
1	Arts Design Entertainment Sports & Media	629
2	Building & Grounds Cleaning & Maintenance	44
3	Business & Financial	544
4	Community & Social Services	241
5	Computer & Mathematical	1408
6	Construction & Extraction	154
7	Education&Training&Library	943
8	Farming Fishing & Forestry	43
9	Food Preparation & Serving Related	298
10	Healthcare Practitioners & Technical	244
11	Healthcare Support	242
12	Installation Maintenance & Repair	133
13	Legal	219
14	Life Physical Social Science	170
15	Management	838
16	Office & Administrative Support	639
17	Personal Care & Service	175
18	Production Occupations	110
19	Protective Service	175
20	Retired	495
21	Sales & Related	1093
22	Student	1584
23	Transportation & Material Moving	218
24	Unemployed	1870

Group By with Average Value

SQL Query:

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

Python Code:

In [12]: `df.groupby('weather')['temperature'].mean().to_frame('avg_temp').reset_index()`Out[12]:

	weather	avg_temp
0	Rainy	55.000000
1	Snowy	30.000000
2	Sunny	68.946271

	weather	avg_temp
0	Rainy	55.000000
1	Snowy	30.000000
2	Sunny	68.946271

Group By with Count of Entries in a Column

SQL Query:

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

Python Code:

```
In [13]: df.groupby('weather')[ 'temperature'].size().to_frame('Count_temp').reset_index()
```

```
Out[13]:   weather  Count_temp
0      Rainy       1210
1     Snowy        1405
2    Sunny        10069
```

Group By with Count of Distinct Values

SQL Query:

```
select weather, count(distinct temperature) as count_distinct_temp from dataset_1 group by weather;
```

Python Code:

```
In [14]: df.groupby('weather')[ 'temperature'].nunique().to_frame('count_distinct_temp').reset_index()
```

```
Out[14]:   weather  count_distinct_temp
0      Rainy             1
1     Snowy             1
2    Sunny             3
```

Group By with Sum

SQL Query:

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

Python Code:

```
In [15]: df.groupby('weather')[ 'temperature'].sum().to_frame('sum_temp').reset_index()
```

```
Out[15]:   weather  sum_temp
0      Rainy      66550
1     Snowy      42150
2    Sunny      694220
```

Group By with Minimum and Maximum Values

SQL Query:

```
select weather,min(temperature) as min_temp from dataset_1 group by weather;
select weather,max(temperature) as max_temp from dataset_1 group by weather;
```

Python Code:

```
In [16]: df.groupby('weather')['temperature'].min().to_frame('min_temp').reset_index()
```

```
Out[16]:   weather  min_temp
0      Rainy       55
1     Snowy        30
2    Sunny        30
```

```
In [17]: df.groupby('weather')['temperature'].max().to_frame('max_temp').reset_index()
```

```
Out[17]:   weather  max_temp
0      Rainy       55
1     Snowy        30
2    Sunny        80
```

Group By with HAVING Clause Equivalent in Pandas

SQL Query:

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

Python Code:

```
In [18]: df.groupby('occupation').filter(lambda x: x['occupation'].iloc[0] == 'Student').groupby('occupat')
```

```
Out[18]: occupation
Student    1584
dtype: int64
```

UNION in SQL and Drop Duplicates in Pandas

SQL Query:

```
select distinct destination from(select * from dataset_1 union select * from table_to_union);
```

Python Code:

```
In [19]: df1 = pd.read_csv(r'C:\Users\Arman\Downloads\dataset\dataset_2.csv')
```

```
In [20]: pd.concat([df, df1])['destination'].drop_duplicates()
```

```
Out[20]: 0      NaN
0    UNION
Name: destination, dtype: object
```

INNER JOIN

SQL Query:

```
select a.destination,a.time,b.part_of_day from dataset_1 a inner join table_to_join b  
on a.time=b.time
```

Python Code:

```
In [21]: df2 = pd.read_csv(r'C:\Users\Arman\Downloads\dataset\dataset_3.csv')
```

```
In [22]: pd.merge(df, df2[['time', 'part_of_day']], on='time', how='inner')[['Destination', 'time', 'part_of_day']]
```

```
Out[22]:
```

	Destination	time	part_of_day
0	No Urgent Place	2PM	Afternoon
1	No Urgent Place	2PM	Afternoon
2	No Urgent Place	2PM	Afternoon
3	No Urgent Place	2PM	Afternoon
4	No Urgent Place	2PM	Afternoon
...
33679533	Work	7AM	Morning
33679534	Work	7AM	Morning
33679535	Work	7AM	Morning
33679536	Work	7AM	Morning
33679537	Work	7AM	Morning

33679538 rows × 3 columns

Filter Rows by Exact Match

SQL Query:

```
select destination ,passanger from(select*from dataset_1 where passanger = 'Alone');
```

Python Code:

```
In [23]: df[df['passanger'] == 'Alone'][['Destination', 'passanger']]
```

Out[23]:

	Destination	passanger
0	No Urgent Place	Alone
13	Home	Alone
14	Home	Alone
15	Home	Alone
16	Work	Alone
...
12676	Home	Alone
12680	Work	Alone
12681	Work	Alone
12682	Work	Alone
12683	Work	Alone

7305 rows × 2 columns

Filter Rows by Prefix Match

SQL Query:

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

Python Code:

In [24]: `df[df['weather'].str.startswith('Sun')]`

Out[24]:

	Destination	passanger	weather	temperature	time	coupon	expiration	gender	age	r
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)		1d	Female	21
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House		2h	Female	21
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away		2h	Female	21
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House		2h	Female	21
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House		1d	Female	21
...
12673	Home	Alone	Sunny	30	6PM	Carry out & Take away		1d	Male	26
12676	Home	Alone	Sunny	80	6PM	Restaurant(20-50)		1d	Male	26
12677	Home	Partner	Sunny	30	6PM	Restaurant(<20)		1d	Male	26
12678	Home	Partner	Sunny	30	10PM	Restaurant(<20)		2h	Male	26
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)		2h	Male	26

10069 rows × 27 columns



Filter values within a range

SQL Query:

```
select distinct temperature from dataset_1 where temperature between 29 AND 75;
```

Python Code:

In [25]: df[(df['temperature'] >= 29) & (df['temperature'] <= 75)]['temperature'].unique()

Out[25]: array([55, 30])

Filter Rows with Multiple Matching Values

SQL Query:

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

Python Code:

In [26]: df[df['occupation'].isin(['Sales & Related', 'Management'])][['occupation']]

Out[26]:

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

SQL vs Python (Pandas) – Side-by-Side Comparison

Task	SQL Command	Pandas Equivalent
Select all columns	<code>SELECT * FROM table;</code>	<code>df</code>
Select specific columns	<code>SELECT col1, col2 FROM table;</code>	<code>df[['col1', 'col2']]</code>
View first N rows	<code>SELECT * FROM table LIMIT N;</code>	<code>df.head(N)</code>
Unique values in a column	<code>SELECT DISTINCT col FROM table;</code>	<code>df['col'].unique()</code>
Filter rows by condition	<code>SELECT * FROM table WHERE col = 'value';</code>	<code>df[df['col'] == 'value']</code>
Filter by multiple values	<code>SELECT * FROM table WHERE col IN ('val1','val2');</code>	<code>df[df['col'].isin(['val1','val2'])]</code>
Filter by range	<code>SELECT * FROM table WHERE col BETWEEN x AND y;</code>	<code>df[(df['col'] >= x) & (df['col'] <= y)]</code>
Filter by prefix/suffix	<code>WHERE col LIKE 'prefix%'</code>	<code>df[df['col'].str.startswith('prefix')]</code>
Sort ascending	<code>SELECT * FROM table ORDER BY col;</code>	<code>df.sort_values('col')</code>
Sort descending	<code>SELECT * FROM table ORDER BY col DESC;</code>	<code>df.sort_values('col', ascending=False)</code>
Rename column	<code>SELECT col AS new_name FROM table;</code>	<code>df.rename(columns={'col': 'new_name'})</code>
Group & count	<code>SELECT col, COUNT(*) FROM table GROUP BY col;</code>	<code>df.groupby('col').size().reset_index(name='count')</code>

Task	SQL Command	Pandas Equivalent
Group & average	<code>SELECT col, AVG(num_col) FROM table GROUP BY col;</code>	<code>df.groupby('col')['num_col'].mean().reset_index()</code>
Count distinct values	<code>COUNT(DISTINCT col)</code>	<code>.nunique()</code>
Aggregate sum	<code>SUM(col)</code>	<code>.sum()</code>
Aggregate min & max	<code>MIN(col), MAX(col)</code>	<code>.min(), .max()</code>
HAVING clause	<code>GROUP BY col HAVING condition;</code>	<code>groupby().filter(lambda x: condition)</code>
Union	<code>SELECT * FROM t1 UNION SELECT * FROM t2;</code>	<code>pd.concat([df1, df2]).drop_duplicates()</code>
Inner join	<code>SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;</code>	<code>pd.merge(df1, df2, on='id', how='inner')</code>
Left join	<code>LEFT JOIN</code>	<code>how='left'</code>
Right join	<code>RIGHT JOIN</code>	<code>how='right'</code>
Cross join	<code>CROSS JOIN</code>	<code>how='cross'</code>

Key Observations

- **Syntax style:**
 - SQL uses declarative syntax – you specify *what* you want, not *how* to do it.
 - Pandas uses method chaining – you apply functions step-by-step to manipulate DataFrames.
- **Execution environment:**
 - SQL queries run inside a database system.
 - Pandas operations run in-memory within Python.
- **Performance:**
 - SQL is optimized for very large datasets in databases.
 - Pandas is faster for small-to-medium datasets in memory but may slow down for very large data.
- **Functionality:**
 - Pandas can do everything SQL can, plus additional data manipulation and integration with Python libraries.
 - SQL is ideal for structured data retrieval and aggregation from relational databases.

Conclusion

Both **SQL** and **Python (Pandas)** are powerful tools for data analysis, but they excel in different scenarios:

- **SQL** is best suited for working directly with large datasets stored in relational databases. It is highly efficient for data extraction, filtering, aggregation, and performing joins without loading the entire dataset into memory.
- **Python (Pandas)** is ideal for in-memory analysis, advanced data manipulation, and integrating with other libraries such as **NumPy**, **Matplotlib**, and **Seaborn** for statistical analysis and visualization.

In practice:

- Use **SQL** to pull and prepare relevant datasets from the database.
- Use **Python** to clean, transform, analyze, and visualize the extracted data.

By combining both, you leverage the **strength of SQL for querying** and the **flexibility of Python for analysis and visualization**, creating a more efficient and comprehensive data analysis workflow.