

Data Analysis Using SQL vs PYTHON

PYTHON

VS

SQL

```
df = pd.read_csv(r'E:\Data Science & AI\Dataset files\dataset_1_202409050942.csv')
```

```
SELECT * FROM dataset_1
```

The image shows a side-by-side comparison of two data analysis environments. On the left is a Jupyter Notebook interface running in a browser window, titled 'Daily code2'. It displays Python code and its output. The code reads a CSV file named 'dataset_1_202409050942.csv' into a DataFrame 'df'. The output shows the first 12 rows of the dataset, which contains columns like destination, passanger, weather, temperature, time, coupon, expiration, gender, age, maritalStatus, and Carr. On the right is a screenshot of the DBeaver 24.2.0 application. It has a 'Script-2' tab open with the same SQL query: 'SELECT * FROM dataset_1'. Below the query is a results grid showing the same 12 rows of data from the CSV file. The DBeaver interface includes various toolbars, a sidebar with database connections, and a status bar at the bottom.

PYTHON

VS

SQL

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

```
SELECT weather,temperature FROM dataset_1;
```

The image displays a side-by-side comparison of two data retrieval methods: Python (left) and SQL (right).
Left Side (Python): A Jupyter Notebook cell with the code `df[['weather','temperature']]` is shown. Below the code, the resulting DataFrame is displayed:

| | 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

`[]:`

`[]:`

`[]:`

`[]:`

`[]:`

`[]:`

`[17]: df.head(10)`

Right Side (SQL): A DBeaver interface connected to an SQLite database. The SQL query `SELECT * FROM dataset_1` is run, and the results are displayed in a grid:

| | weather | temperature |
|----|---------|-------------|
| 1 | Sunny | 55 |
| 2 | Sunny | 80 |
| 3 | Sunny | 80 |
| 4 | Sunny | 80 |
| 5 | Sunny | 80 |
| 6 | Sunny | 80 |
| 7 | Sunny | 55 |
| 8 | Sunny | 80 |
| 9 | Sunny | 80 |
| 10 | Sunny | 80 |
| 11 | Sunny | 80 |
| 12 | Sunny | 55 |
| 13 | Sunny | 55 |

200 200+ 200 row(s) fetched - 0.001s, on 2024-09-09 at 19:58:05

IST en Writable Sm...rt

The two interfaces show identical data, demonstrating that the Python DataFrame and the SQL table represent the same dataset.

PYTHON

VS

SQL

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

```
SELECT * FROM dataset_1 LIMIT 10;
```

The image shows a side-by-side comparison between a Jupyter Notebook in a browser and a DBeaver database client.

Jupyter Notebook (Left):

- URL: localhost:8888/notebooks...
- Kernel: Python 3 (ipykernel)
- Code cell [17]: `df.head(10)`
- Output cell [17]: Displays the first 10 rows of a DataFrame named `dataset_1`. The columns are: destination, passanger, weather, temperature, time, coupon, expiration, gender, age, maritalStatus, ... , CarryAwa. The data includes various categories like 'No Urgent Place', 'Friend(s)', 'Kid(s)', and different weather conditions like 'Sunny' or 'Rainy'.
- Message: 10 rows x 27 columns

DBeaver (Right):

- Version: 24.2.0 - <SQLite Test.db> Script-2
- Database: SQLite Test.db
- Script Editor: Contains the SQL query: `SELECT * FROM dataset_1` followed by `SELECT weather,temperature FROM dataset_1;` and `SELECT * FROM dataset_1 LIMIT 10;`
- Result Grid: Shows the same 10 rows of data from the dataset_1 table, matching the output of the Jupyter cell.
- Message: 10 row(s) fetched - 0.002s (0.001s fetch), on 2024-09-09 at 19:59:49

PYTHON

VS

SQL

```
df['passanger'].unique()
```

```
SELECT DISTINCT passanger FROM dataset_1;
```

The image shows a dual-monitor setup. The left monitor displays a Jupyter Notebook interface with a code cell containing the Python command `df['passanger'].unique()`. The right monitor displays DBeaver 24.2.0 connected to an SQLite database named `SQLite Test.db`. In the SQL Editor tab, three queries are run:

```
SELECT weather,temperature FROM dataset_1;
SELECT * FROM dataset_1 LIMIT 10;
SELECT DISTINCT passanger FROM dataset_1;
```

The results of the third query are displayed in a grid view titled `dataset_11`, showing 10 rows of data with columns: destination, passanger, weather, temperature, time, and cou.

| | destination | passanger | weather | temperature | time | cou |
|----|-----------------|-----------|---------|-------------|------|-------|
| 1 | No Urgent Place | Alone | Sunny | 55 | 2PM | Resta |
| 2 | No Urgent Place | Friend(s) | Sunny | 80 | 10AM | Coffe |
| 3 | No Urgent Place | Friend(s) | Sunny | 80 | 10AM | Carry |
| 4 | No Urgent Place | Friend(s) | Sunny | 80 | 2PM | Coffe |
| 5 | No Urgent Place | Friend(s) | Sunny | 80 | 2PM | Coffe |
| 6 | No Urgent Place | Friend(s) | Sunny | 80 | 6PM | Resta |
| 7 | No Urgent Place | Friend(s) | Sunny | 55 | 2PM | Carry |
| 8 | No Urgent Place | Kid(s) | Sunny | 80 | 10AM | Resta |
| 9 | No Urgent Place | Kid(s) | Sunny | 80 | 10AM | Carry |
| 10 | No Urgent Place | Kid(s) | Sunny | 80 | 10AM | Bar |

PYTHON

VS

SQL

```
df[df['destination']=='Home']
```

```
SELECT * FROM dataset_1 WHERE destination = 'Home';
```

The image shows a side-by-side comparison between a Python Jupyter Notebook interface and a SQL DBeaver interface, both displaying results for a query to filter a dataset by destination.

Jupyter Notebook (Left):

- The browser tab is titled "Daily code2".
- The URL is "localhost:8888/notebooks...".
- The notebook cell [21] contains the Python code:

```
df[df['destination']=='Home']
```

.
- The output shows a Pandas DataFrame with 3237 rows and 27 columns, filtered for rows where destination is 'Home'.
- The columns include: destination, passanger, weather, temperature, time, coupon, expiration, gender, age, maritalStatus, ... , Carr.
- Sample data rows:
 - [13]: Home, Alone, Sunny, 55, 6PM, Bar, 1d, Female, 21, Unmarried partner, -
 - [14]: Home, Alone, Sunny, 55, 6PM, Restaurant(20-50), 1d, Female, 21, Unmarried partner, -
 - [15]: Home, Alone, Sunny, 80, 6PM, Coffee House, 2h, Female, 21, Unmarried partner, -
 - [35]: Home, Alone, Sunny, 55, 6PM, Bar, 1d, Male, 21, Single, -
 - [36]: Home, Alone, Sunny, 55, 6PM, Restaurant(20-50), 1d, Male, 21, Single, -

DBeaver (Right):

- The title bar says "DBeaver 24.2.0 - <SQLite Test.db> Script-2".
- The SQL Editor pane shows the query:

```
SELECT * FROM dataset_1 WHERE destination = 'Home';
```

.
- The Results pane displays the same dataset as the Jupyter Notebook, showing 3237 rows of data.
- The results table has columns: destination, passanger, weather, temperature, time.
- Sample data rows:
 - [1]: Home, Alone, Sunny, 55, 6PM, Bar
 - [2]: Home, Alone, Sunny, 55, 6PM, Restaurant(20-50)
 - [3]: Home, Alone, Sunny, 80, 6PM, Coffee House
 - [4]: Home, Alone, Sunny, 55, 6PM, Bar
 - [5]: Home, Alone, Sunny, 55, 6PM, Restaurant(20-50)

PYTHON

VS

SQL

```
df.sort_values('coupon')
```

```
SELECT * FROM dataset_1 ORDER BY coupon;
```

The image displays two side-by-side software interfaces for data analysis: a Jupyter Notebook in a browser window and DBeaver.

Jupyter Notebook (Left):

- The title bar shows "Daily code2" and "JupyterLab".
- The toolbar includes File, Edit, View, Run, Kernel, Settings, Help, and a Trusted button.
- The main area shows a command cell: `[23]: df.sort_values('coupon')` and its output: a table with 12684 rows and 27 columns.
- The table includes columns: destination, passanger, weather, temperature, time, coupon, expiration, gender, age, maritalStatus, ... Ca.
- Sample data rows are visible, such as:

 - Row 11702: Home, Partner, Sunny, 30, 10PM, Bar, 2h, Female, 50plus, Married partner, ...
 - Row 9930: No Urgent Place, Alone, Snowy, 30, 2PM, Bar, 1d, Female, 21, Single, ...
 - Row 10632: Home, Alone, Rainy, 55, 6PM, Bar, 1d, Male, 21, Single, ...
 - Row 7997: No Urgent Place, Friend(s), Rainy, 55, 10PM, Bar, 2h, Male, 26, Unmarried partner, ...
 - Row 11166: Work, Alone, Snowy, 30, 7AM, Bar, 1d, Female, 41, Married partner, ...
 - Row 10476: Home, Alone, Sunny, 80, 6PM, Restaurant(<20), 1d, Female, 31, Unmarried partner, ...
 - Row 5447: Home, Alone, Sunny, 80, 10PM, Restaurant(<20), 2h, Female, 50plus, Single, ...
 - Row 10478: Home, Alone, Snowy, 30, 10PM, Restaurant(<20), 2h, Female, 31, Unmarried partner, ...
 - Row 5440: No Urgent Place, Alone, Sunny, 80, 2PM, Restaurant(<20), 2h, Female, 50plus, Single, ...
 - Row 0: No Urgent Place, Alone, Sunny, 55, 2PM, Restaurant(<20), 1d, Female, 21, Unmarried partner, ...

DBeaver (Right):

- The title bar shows "DBeaver 24.2.0 - <SQLite Test.db> Script-2".
- The toolbar includes File, Edit, Navigate, Search, SQL Editor, Database, Window, Help.
- The SQL Editor tab is selected, showing the same SQL query: `SELECT * FROM dataset_1 ORDER BY coupon;`.
- The Results tab shows the same table data as the Jupyter Notebook, with 12 rows displayed.
- The status bar at the bottom indicates "200 row(s) fetched - 0.006s (0.005s fetch), on 2024-09-09 at 20:03:13".

PYTHON

VS

SQL

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

```
df()
```

The screenshot shows a Jupyter Notebook interface with two code cells and a preview of the resulting DataFrame.

- Cell 57:** Contains the Python code: `df.rename(columns={'destination':'Destination'},inplace=True)`.
- Cell 27:** Contains the command `df`, which displays the DataFrame.
- Data Preview:** Shows the first 10 rows of the DataFrame, which has 12684 rows and 27 columns. The columns include Destination, passenger, weather, temperature, time, coupon, expiration, gender, age, maritalStatus, and Carr.

```
SELECT destination as 'Destination' FROM dataset_1;
```

The screenshot shows the DBeaver interface with a script editor and a results grid.

- Script Editor:** Contains several SQL queries:
 - `SELECT * FROM dataset_1`
 - `SELECT weather,temperature FROM dataset_1;`
 - `SELECT * FROM dataset_1 LIMIT 10;`
 - `SELECT DISTINCT passanger FROM dataset_1;`
 - `SELECT * FROM dataset_1 WHERE destination = 'Home';`
 - `SELECT * FROM dataset_1 ORDER BY coupon;`
 - `SELECT destination as 'Destination' FROM dataset_1;`
- Results Grid:** Displays the results of the last query, showing a single column named "Destination" with 13 rows, all of which are "No Urgent Place".



PYTHON

VS

SQL

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

```
SELECT occupation FROM dataset_1 GROUP BY occupation;
```

The image shows a side-by-side comparison of two data analysis environments: a Jupyter Notebook in a web browser and DBeaver.

Jupyter Notebook (Left):

- The URL is `localhost:8888/notebooks...`.
- The code cell [29] contains the command: `df.groupby('occupation').size().to_frame('Count').reset_index()`.
- The resulting output is a DataFrame:

| | 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 |

DBeaver (Right):

- The database is `<SQLite Test.db>`.
- The script pane contains the following SQL queries:

```
SELECT destination as 'Destination' FROM dataset_1;
SELECT occupation FROM dataset_1 GROUP BY occupation;
```

- The results pane shows the same data as the Jupyter output:

| Name | |
|------|---|
| 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 |

PYTHON

VS

SQL

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

```
SELECT weather, AVG(temperature) AS  
'avg_temp' FROM dataset_1 GROUP BY weather;
```

The image shows a side-by-side comparison of two data analysis environments: a Jupyter Notebook in a browser window and a DBeaver database client.

Jupyter Notebook (Left):

- The browser tab is titled "Daily code2".
- The URL is "localhost:8888/notebooks...".
- The notebook cell [31] contains the Python code:

```
df.groupby('weather')['temperature'].mean().to_frame('avg_temp').reset_index()
```
- The output of the cell is a DataFrame:

| | weather | avg_temp |
|---|---------|-----------|
| 0 | Rainy | 55.000000 |
| 1 | Snowy | 30.000000 |
| 2 | Sunny | 68.946271 |

DBeaver (Right):

- The title bar says "DBeaver 24.2.0 - <SQLite Test.db> Script-2".
- The main area shows the following SQL query results:

| | weather | avg_temp |
|---|---------|---------------|
| 1 | Rainy | 55 |
| 2 | Snowy | 30 |
| 3 | Sunny | 68.9462707319 |

PYTHON

VS

SQL

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

```
SELECT weather,COUNT(temperature) AS  
'count_temp' FROM dataset_1 GROUP BY weather;
```

The image shows a side-by-side comparison between a Python Jupyter Notebook and a DBeaver SQL interface.

Jupyter Notebook (Left):

- Shows a code cell with the command: `df.groupby('weather')['temperature'].size().to_frame()('Count_temp').reset_index()`.
- Shows the resulting DataFrame:

| | weather | Count_temp |
|---|---------|------------|
| 0 | Rainy | 1210 |
| 1 | Snowy | 1405 |
| 2 | Sunny | 10069 |

DBeaver (Right):

- Shows the SQL query: `SELECT weather,COUNT(temperature) AS 'count_temp' FROM dataset_1 GROUP BY weather;`
- Shows the results in a table:

| | weather | count_temp |
|---|---------|------------|
| 1 | Rainy | 1,210 |
| 2 | Snowy | 1,405 |
| 3 | Sunny | 10,069 |

PYTHON

VS

SQL

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

```
SELECT weather ,COUNT(DISTINCT temperature) AS  
count_distinct_temp FROM dataset_1 GROUP BY weather;
```

The image shows a dual-monitor setup. The left monitor displays a Jupyter Notebook interface with a Python code cell and its output. The right monitor displays a DBeaver database client with an open SQL editor and a results grid.

Jupyter Notebook (Left Monitor):

```
[35]: df.groupby('weather')['temperature'].nunique().to_frame('count_distinct_temp').reset_index()  
[35]: weather count distinct temp  
[ 0: Rainy 1  
[ 1: Snowy 1  
[ 2: Sunny 3
```

DBeaver (Right Monitor):

```
SELECT weather ,COUNT(temperature) AS 'count_temp' FROM dataset_1 GROUP BY weather  
SELECT weather ,COUNT(DISTINCT temperature) AS count_distinct_temp FROM dataset_1 GROUP BY weather  
SELECT weather ,SUM(temperature) AS sum_temp FROM dataset_1 GROUP BY weather  
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  
SELECT occupation FROM dataset_1 GROUP BY occupation HAVING count_occupations > 1  
dataset_1 1 x  
SELECT weather ,COUNT(DISTINCT temperature) AS count_distinct_temp FROM dataset_1 GROUP BY weather  
Grid: A-Z weather 123 count_distinct_temp  
1 Rainy 1  
2 Snowy 1  
3 Sunny 3
```

PYTHON

VS

SQL

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

```
SELECT weather ,SUM(temperature) AS sum_temp FROM  
dataset_1 GROUP BY weather;
```

The image shows a side-by-side comparison of two data processing environments. On the left is a Jupyter Notebook interface running on a local server, displaying Python code and its output. On the right is a DBeaver database client showing the equivalent SQL query and its results.

Jupyter Notebook (Left):

```
[37]: df.groupby('weather')['temperature'].sum().to_frame('sum_temp').reset_index()  
[37]:  
      weather  sum_temp  
0   Rainy     66550  
1  Snowy     42150  
2  Sunny    694220
```

DBeaver (Right):

```
SELECT destination as 'Destination' FROM dataset_1;  
SELECT occupation FROM dataset_1 GROUP BY occupation;  
SELECT weather,AVG(temperature) AS 'avg_temp' FROM dataset_1 GROUP BY weather;  
SELECT weather,COUNT(temperature) AS 'count_temp' FROM dataset_1 GROUP BY weather;  
SELECT weather ,COUNT(DISTINCT temperature) AS count_distinct_temp FROM dataset_1 GROUP BY weather;  
SELECT weather ,SUM(temperature) AS sum_temp FROM dataset_1 GROUP BY weather;
```

| weather | sum_temp |
|---------|----------|
| Rainy | 66,550 |
| Snowy | 42,150 |
| Sunny | 694,220 |

PYTHON

VS

SQL

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

```
SELECT weather ,MIN(temperature) AS  
min_temp FROM dataset_1 GROUP BY weather;
```

The image shows a side-by-side comparison of two data processing environments. On the left, a Jupyter Notebook cell displays Python code to group a DataFrame by 'weather' and select the minimum temperature, resulting in a DataFrame with three rows: Rainy (55), Snowy (30), and Sunny (30). On the right, a DBeaver interface shows an equivalent SQL query to achieve the same result. Both results are displayed in a grid view, showing the same three rows of data.

Jupyter Notebook Cell Output:

```
[39]: df.groupby('weather')['temperature'].min().to_frame()  
[39]:  
weather min_temp  
0 Rainy 55  
1 Snowy 30  
2 Sunny 30
```

DBeaver Grid View:

| Name | weather | min_temp |
|------|---------|----------|
| 1 | Rainy | 55 |
| 2 | Snowy | 30 |
| 3 | Sunny | 30 |

Detailed description: The image captures a desktop environment with two windows side-by-side. The left window is a Jupyter Notebook interface running in a browser (localhost:8888). It shows a single code cell with the Python command to find the minimum temperature for each weather type. The output of this cell is a DataFrame with three rows: Rainy (55), Snowy (30), and Sunny (30). The right window is DBeaver, a database management tool. It has an open SQLite database named 'Test.db'. In the SQL Editor tab, there is a script containing multiple SELECT statements to calculate various metrics from a dataset. Below the editor is a results grid showing the same three rows of data as the Jupyter output, with columns labeled 'Name', 'weather', and 'min_temp'. The status bar at the bottom of the DBeaver window shows the time as 20:12:2024 and the date as 09-09-2024.

PYTHON

VS

SQL

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

```
SELECT weather ,MAX(temperature) AS  
max_temp FROM dataset_1 GROUP BY weather;
```

The image shows a side-by-side comparison of two data processing environments. On the left, a Jupyter Notebook cell displays Python code for grouping a dataset by weather and extracting the maximum temperature. On the right, a DBeaver interface shows the equivalent SQL query for the same operation. Both queries return a dataset with columns 'weather' and 'max_temp', containing three rows: Rainy (55), Snowy (30), and Sunny (80).

Jupyter Notebook Output:

```
[59]: df.groupby('weather')['temperature'].max().to_frame('max_temp').reset_index()  
[59]:   weather  max_temp  
0    Rainy      55  
1   Snowy      30  
2   Sunny      80
```

DBeaver Output:

```
*<SQLite Test.db> Script-2 <SQLite Test.db> Script-1  
SELECT weather,MAX(temperature) AS 'max_temp' FROM dataset_1 GROUP BY weather;  
dataset_1 1 dataset_1 2 dataset_1 3 dataset_1 4  
Grid A-Z weather 123 max_temp  
1 Rainy 55  
2 Snowy 30  
3 Sunny 80
```

PYTHON

VS

SQL

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

```
SELECT occupation FROM dataset_1 GROUP BY  
occupation HAVING occupation='Student';
```

The image shows a side-by-side comparison of two data processing environments. On the left is a Jupyter Notebook interface running on a local server, displaying Python code and its execution results. On the right is a DBeaver interface connected to an SQLite database, displaying the equivalent SQL query and its results.

Jupyter Notebook (Left):

- URL: localhost:8888/notebooks...
- Kernel: Python 3 (ipykernel)
- Code cell 43:

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

```
occupation  
Student    1584  
dtype: int64
```

DBeaver (Right):

- Version: DBeaver 24.2.0 - <SQLite Test.db> Script-2
- Database: SQLite Test.db
- Script 2:

```
SELECT weather,AVG(temperature) AS 'avg_temp' FROM dataset_1 GROUP BY weather  
SELECT weather,COUNT(temperature) AS 'count_temp' FROM dataset_1 GROUP BY weather  
SELECT weather,COUNT(DISTINCT temperature) AS count_distinct_temp FROM dataset_1 GROUP BY weather  
SELECT weather,SUM(temperature) AS sum_temp FROM dataset_1 GROUP BY weather  
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  
SELECT occupation FROM dataset_1 GROUP BY occupation HAVING occupation='Student'
```
- Table: **dataset_1** (Grid View)

| occupation |
|------------|
| Student |

PYTHON

VS

SQL

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

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

10069 rows x 27 columns

```
SELECT * FROM dataset_1 WHERE weather LIKE 'Sun%';
```

| Name | 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 |

200 200+ 200 row(s) fetched - 0.004s (0.004s fetch), on 2024-09-09 at 20:18:08



PYTHON

VS

SQL

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

```
SELECT DISTINCT temperature FROM  
dataset_1 WHERE temperature BETWEEN 29 AND 75;
```

The image shows a side-by-side comparison of two data processing environments. On the left is a Jupyter Notebook interface running on localhost:8888, displaying Python code to filter a DataFrame and extract unique temperatures. On the right is a DBeaver interface connected to an SQLite database, showing the equivalent SQL query to achieve the same result.

Jupyter Notebook (Left):

```
[53]: df[(df['temperature'] >= 29) & (df['temperature'] <= 75)]['temperature'].unique()  
[53]: array([55, 30], dtype=int64)
```

DBeaver (Right):

```
SELECT * FROM dataset_1 WHERE weather LIKE 'Sun%';  
SELECT DISTINCT temperature FROM dataset_1 WHERE temperature BETWEEN
```

The DBeaver interface also displays the results of the SQL query, showing a grid with two rows of data:

| temperature |
|-------------|
| 55 |
| 30 |

PYTHON

VS

SQL

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

```
[['occupation']]
```

```
SELECT occupation FROM dataset_1
```

```
WHERE occupation IN ('sales & Related','Management');
```

The image shows a side-by-side comparison of two data analysis environments: a Jupyter Notebook in a browser window and a DBeaver database client.

Jupyter Notebook (Left):

- The title bar says "jupyter Daily code2 Last Checkpoint: 3 days ago".
- The menu bar includes File, Edit, View, Run, Kernel, Settings, Help.
- The toolbar has icons for New, Open, Save, Run Cell, Stop, Kernel, Help, and Python 3 (ipykernel).
- In the code cell [55]:, the following Python code is shown:


```
df[df['occupation'].isin(['Sales & Related', 'Management'])][['occupation']]
```
- The output cell [55] displays the resulting DataFrame:

| 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 |

 Total: 1931 rows × 1 columns
- Below the output, there are several empty code cells starting with []:

DBeaver (Right):

- The title bar says "DBeaver 24.2.0 - <SQLite Test.db> Script-2".
- The menu bar includes File, Edit, Navigate, Search, SQL Editor, Database, Window, Help.
- The toolbar includes icons for New, Open, Save, Commit, Rollback, Auto, SQLite Test.db, and N/A.
- The main area shows a complex multi-line SQL query:


```
SELECT weather ,SUM(temperature) AS sum_temp FROM dataset_1 GROUP BY weather
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
SELECT occupation FROM dataset_1 GROUP BY occupation HAVING occupation = 'Management'
SELECT DISTINCT destination FROM(SELECT * FROM dataset_1 UNION SELECT * FROM dataset_1 WHERE weather LIKE 'Sun%')
SELECT DISTINCT temperature FROM dataset_1 WHERE temperature BETWEEN 10 AND 30
SELECT occupation FROM dataset_1 WHERE occupation IN ('sales & Related', 'Management')
```
- A result grid titled "dataset_1" shows the following data:

| Name | Record | occupation |
|------|------------|------------|
| 1 | Management | |
| 2 | Management | |
| 3 | Management | |
| 4 | Management | |
| 5 | Management | |
| 6 | Management | |
| 7 | Management | |
| 8 | Management | |
| 9 | Management | |
| 10 | Management | |
| 11 | Management | |
| 12 | Management | |
| 13 | Management | |

 Total: 200 row(s) fetched - 0.002s (0.001s fetch), on 2024-09-09 at 20:20:33
- The status bar at the bottom shows: IST en Writable Sm...rt 20:21 09-09-2024 ENG IN PRE

