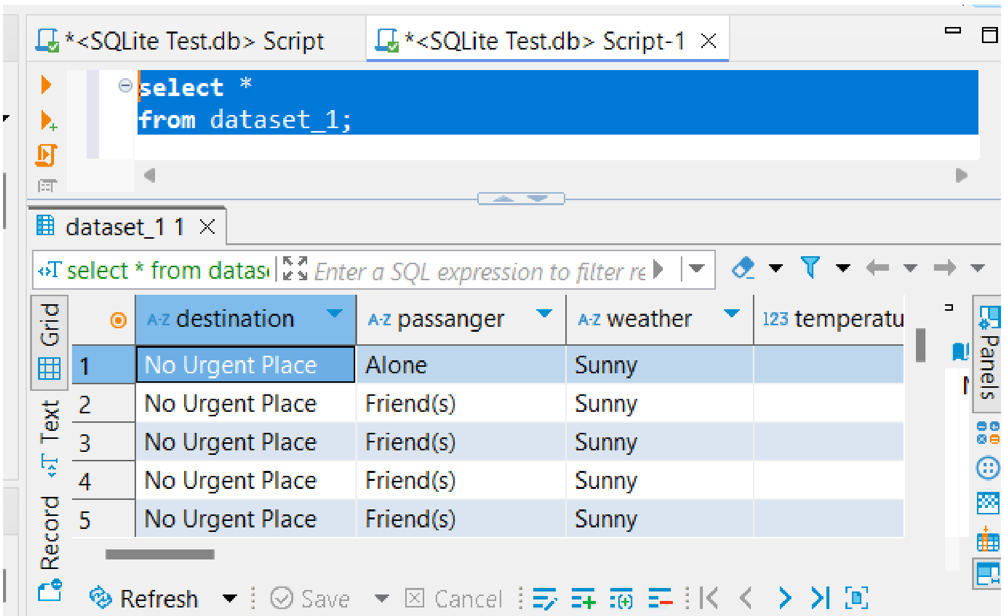
Comparison of SQL Queries in SQLite and Python Queries (By Sunitha Mekala)

1) In SQL:

Reading the data from the dataset – table name – dataset_1

select *
from dataset_1;



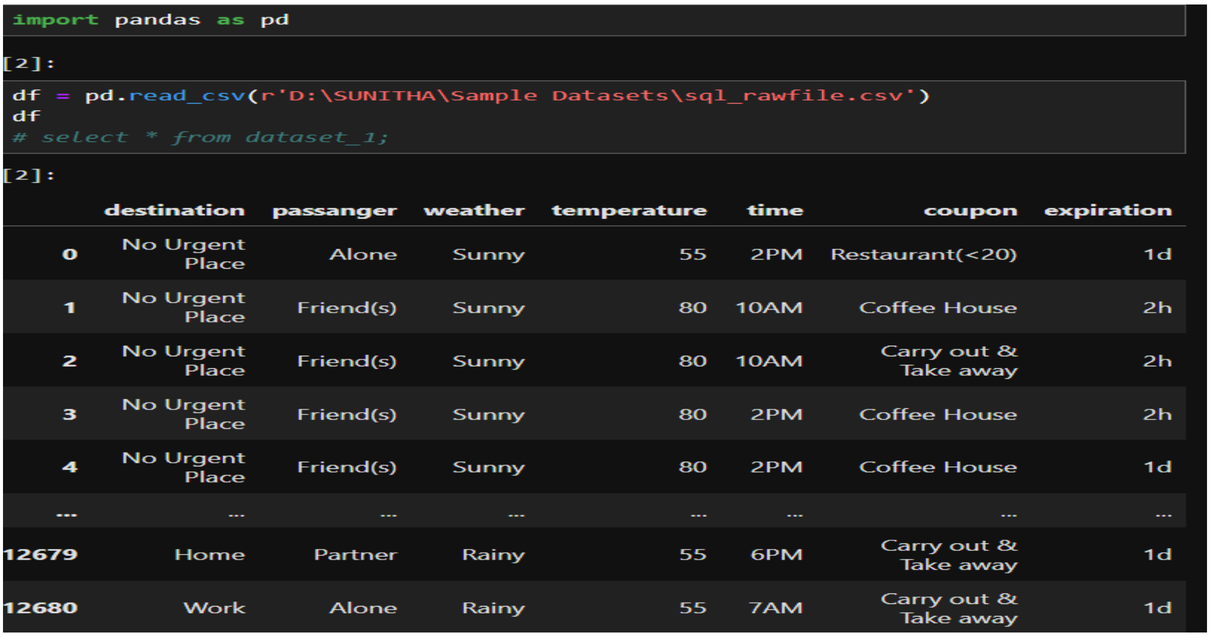
```
select *
from dataset_1;
```

	destination	passanger	weather	temperatu
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	

In Python:

df = pd.read_csv(r'D:\SUNITHA\Sample Datasets\sql_rawfile.csv')

df



```
import pandas as pd
```

```
[2]:
df = pd.read_csv(r'D:\SUNITHA\Sample Datasets\sql_rawfile.csv')
df
# select * from dataset_1;
```

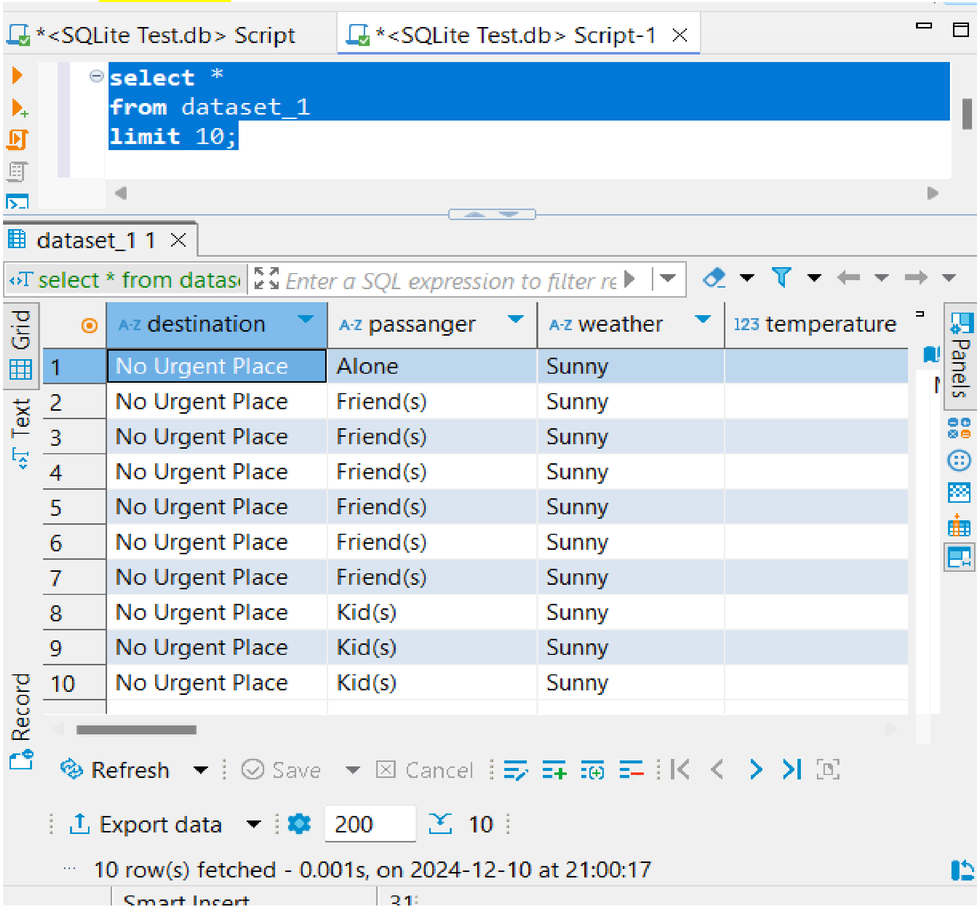
[2]:

	destination	passanger	weather	temperature	time	coupon	expiration
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d
...	...	...	...	...	...	...	...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d

2) In SQL:

To retrieve the top 10 records of the table:

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



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

select * from datas

Enter a SQL expression to filter re

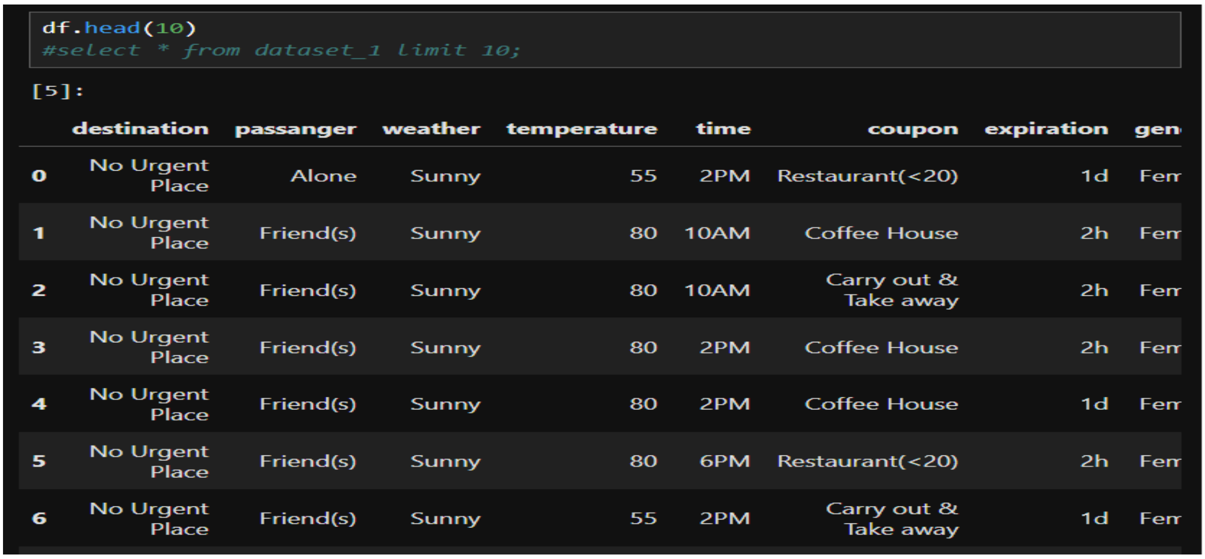
	destination	passanger	weather	temperature
1	No Urgent Place	Alone	Sunny	
2	No Urgent Place	Friend(s)	Sunny	
3	No Urgent Place	Friend(s)	Sunny	
4	No Urgent Place	Friend(s)	Sunny	
5	No Urgent Place	Friend(s)	Sunny	
6	No Urgent Place	Friend(s)	Sunny	
7	No Urgent Place	Friend(s)	Sunny	
8	No Urgent Place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

10 row(s) fetched - 0.001s, on 2024-12-10 at 21:00:17

In Python:

to retrieve top 10 rows

df.head(10)



```
df.head(10)
#select * from dataset_1 limit 10;
```

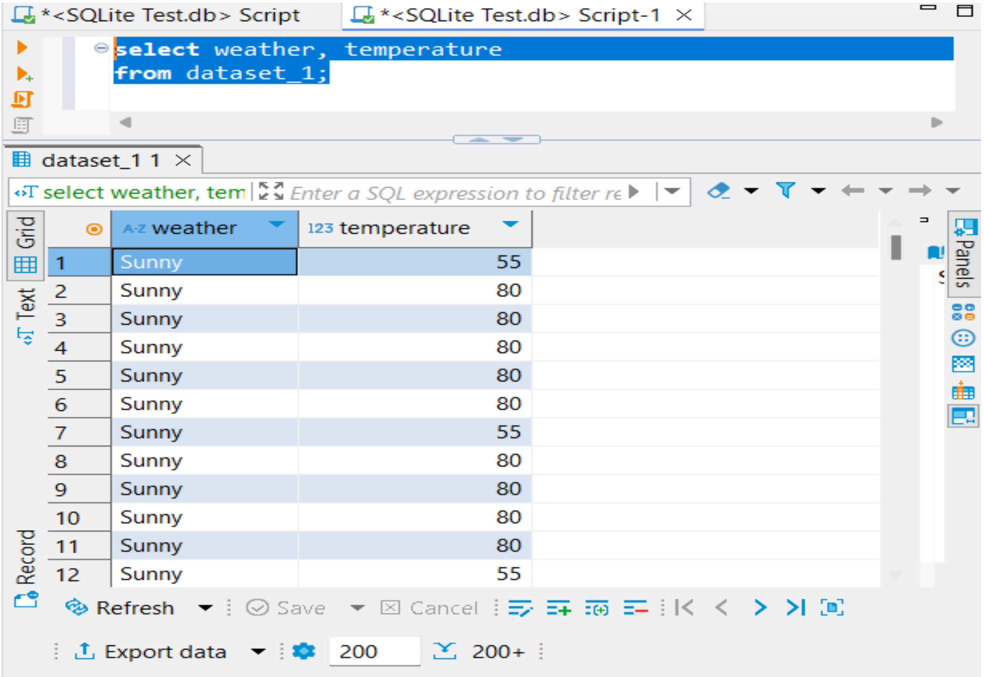
[5]:

	destination	passanger	weather	temperature	time	coupon	expiration	gen
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Fem
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Fem
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Fem
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Fem
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Fem
5	No Urgent Place	Friend(s)	Sunny	80	6PM	Restaurant(<20)	2h	Fem
6	No Urgent Place	Friend(s)	Sunny	55	2PM	Carry out & Take away	1d	Fem

3) **In SQL** :

to retrieve whether and temperature from dataset_1 table.

Select weather, temperature from dataset_1;



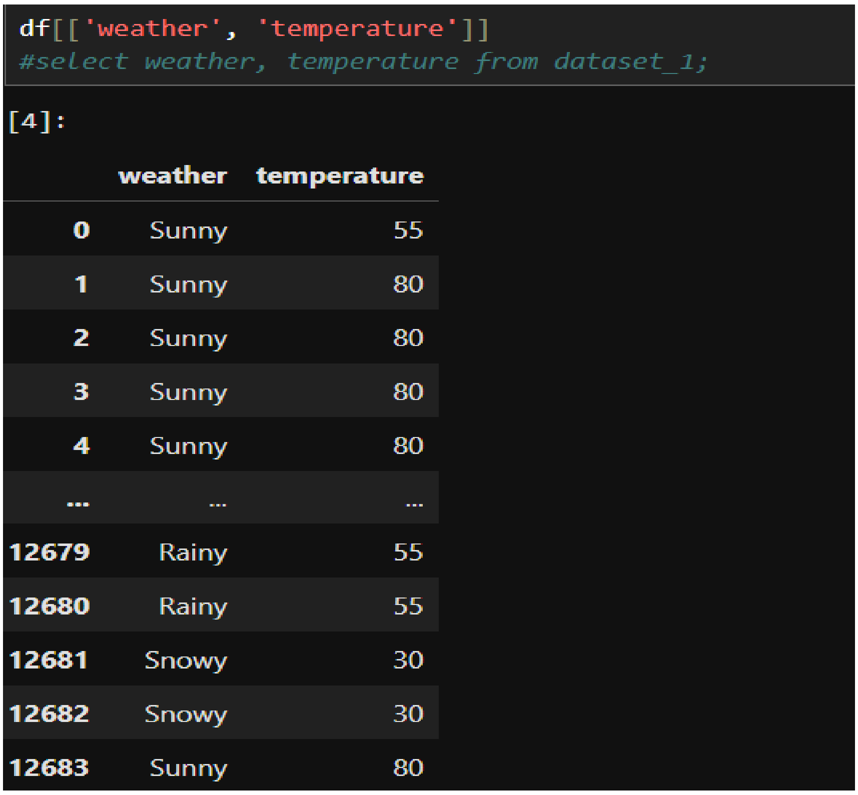
```
select weather, temperature
from dataset_1;
```

select weather, tem

	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

In Python:

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



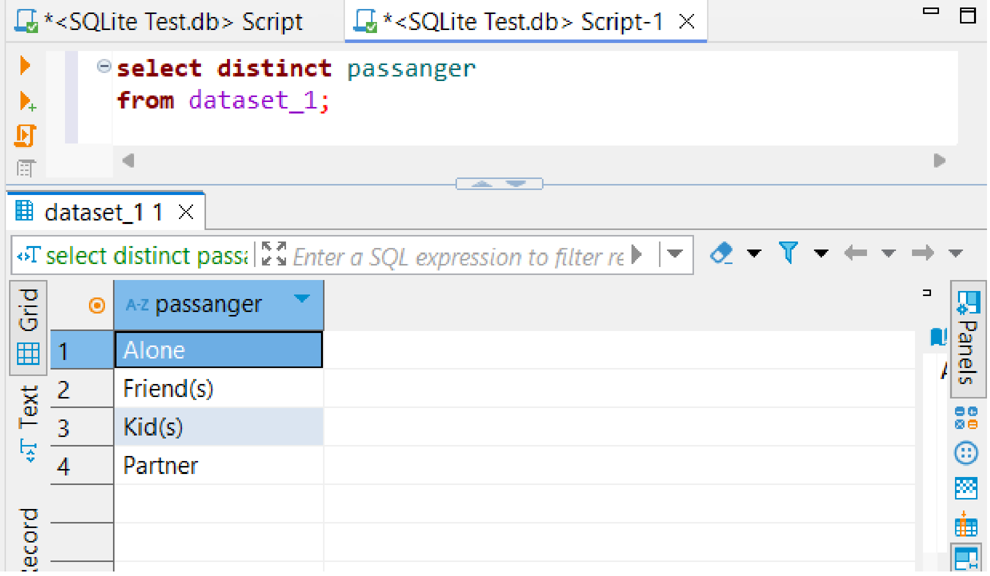
```
df[['weather', 'temperature']]
#select weather, temperature from dataset_1;
```

```
[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
```

4) In SQL:

Gives the distinct values for the respective attribute:

select distinct passanger **from** dataset_1;

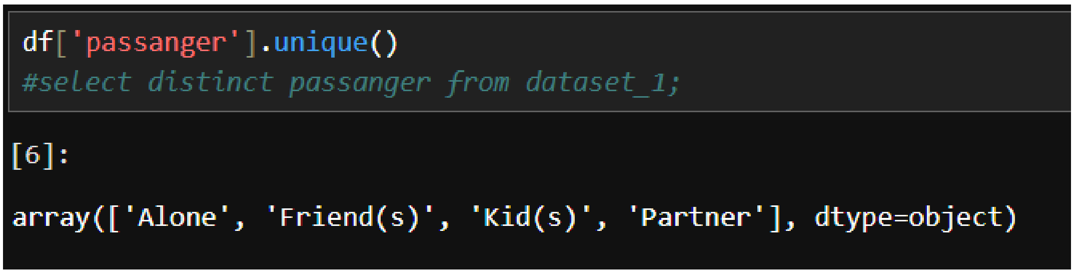


```
select distinct passanger
from dataset_1;
```

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

In Python:

df['passanger'].unique()



```
df['passanger'].unique()
#select distinct passanger from dataset_1;
```

```
[6]:

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

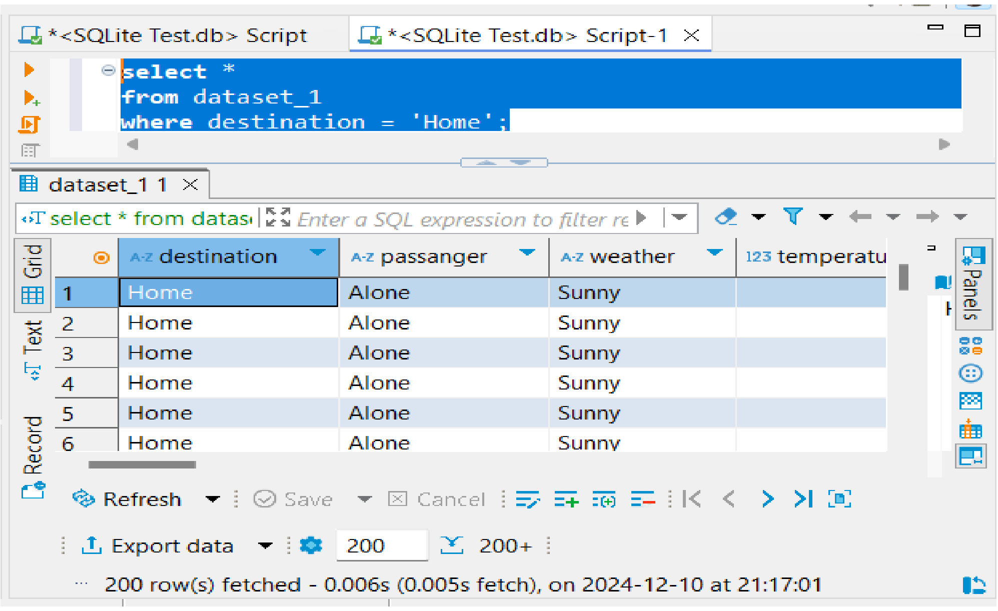
5) In SQL:

Retrieving the values where destination attribute has just home as value in it.

select *

from dataset_1

where destination = 'Home';



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

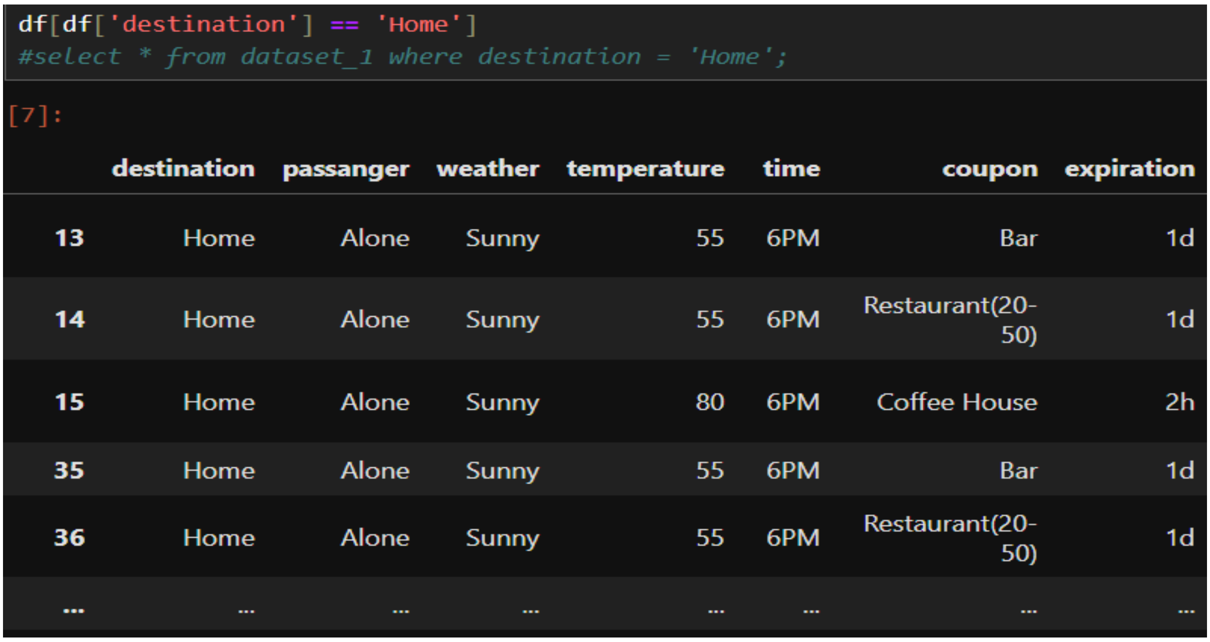
select * from datas | Enter a SQL expression to filter re

	destination	passanger	weather	temperatu
1	Home	Alone	Sunny	
2	Home	Alone	Sunny	
3	Home	Alone	Sunny	
4	Home	Alone	Sunny	
5	Home	Alone	Sunny	
6	Home	Alone	Sunny	

200 row(s) fetched - 0.006s (0.005s fetch), on 2024-12-10 at 21:17:01

In Python:

df[df[‘destination’] == ‘Home’]



```python
df[df['destination'] == 'Home']
#select * from dataset_1 where destination = 'Home';
```

[7]:

	destination	passanger	weather	temperature	time	coupon	expiration
13	Home	Alone	Sunny	55	6PM	Bar	1d
14	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1d
15	Home	Alone	Sunny	80	6PM	Coffee House	2h
35	Home	Alone	Sunny	55	6PM	Bar	1d
36	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1d
...	...	...	...	...	...	...	...

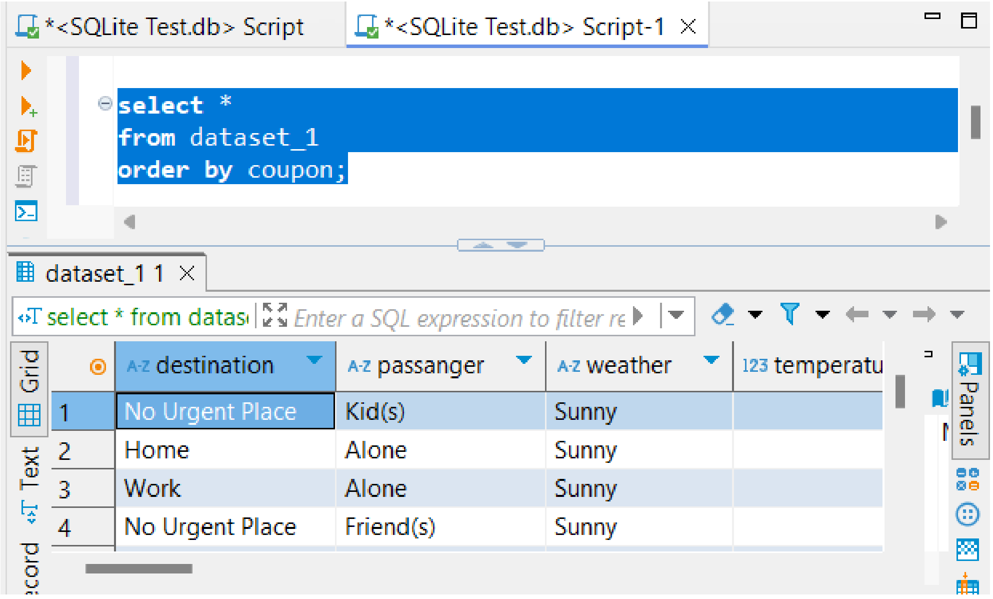
6) In SQL:

Sorting the values by ‘coupon’ attribute and retrieving the values.

select *

from dataset_1

order by coupon;

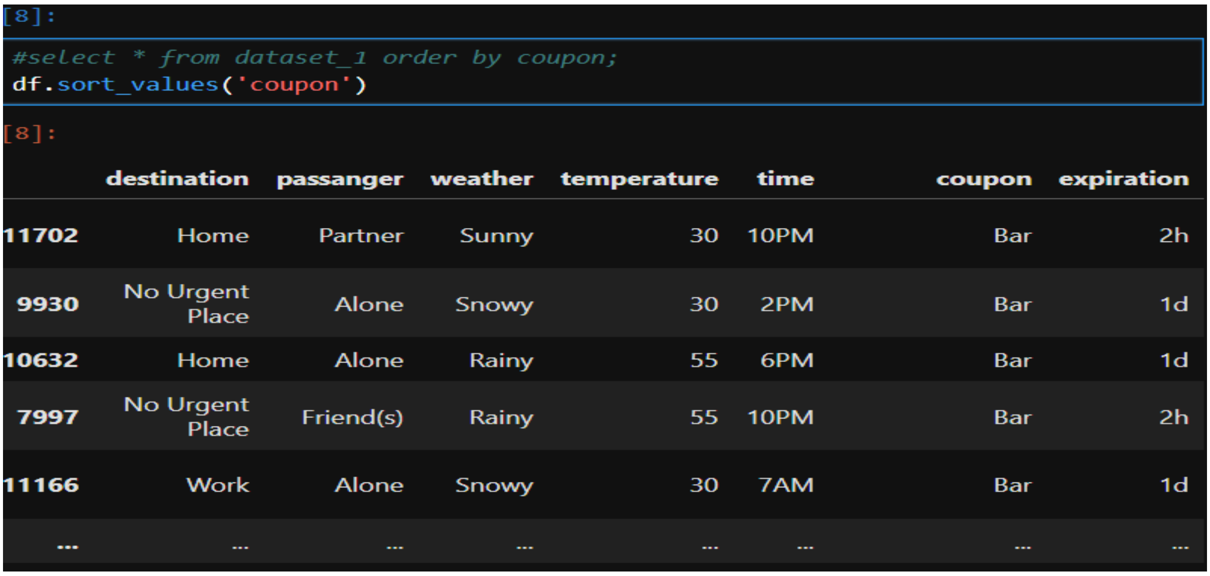


```
select *
from dataset_1
order by coupon;
```

	destination	passanger	weather	temperatu
1	No Urgent Place	Kid(s)	Sunny	
2	Home	Alone	Sunny	
3	Work	Alone	Sunny	
4	No Urgent Place	Friend(s)	Sunny	

In Python:

df.sort_values(‘coupon’)



```
[8]:
#select * from dataset_1 order by coupon;
df.sort_values('coupon')
```

[8]:

	destination	passanger	weather	temperature	time	coupon	expiration
11702	Home	Partner	Sunny	30	10PM	Bar	2h
9930	No Urgent Place	Alone	Snowy	30	2PM	Bar	1d
10632	Home	Alone	Rainy	55	6PM	Bar	1d
7997	No Urgent Place	Friend(s)	Rainy	55	10PM	Bar	2h
11166	Work	Alone	Snowy	30	7AM	Bar	1d
...	...	...	...	...	...	...	...

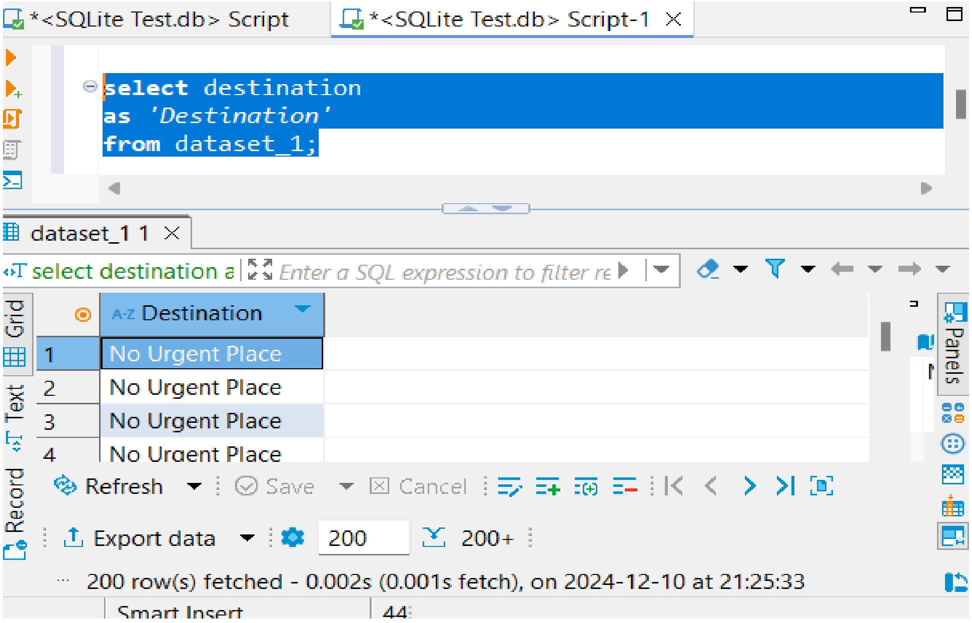
7) **In SQL:**

To rename the column names

select destination

as *'Destination'*

from dataset_1;



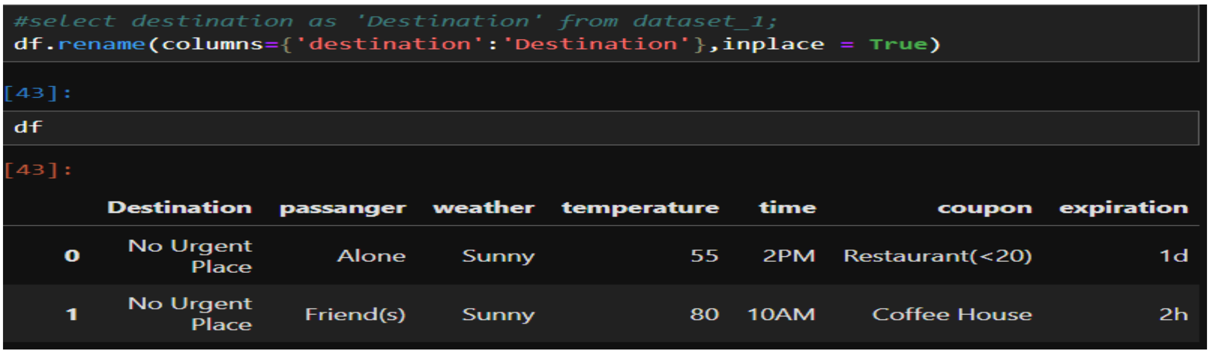
```sql
select destination
as 'Destination'
from dataset_1;
```

	Destination
1	No Urgent Place
2	No Urgent Place
3	No Urgent Place
4	No Urgent Place

```
200 row(s) fetched - 0.002s (0.001s fetch), on 2024-12-10 at 21:25:33
```

In Python:

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



```python
#select destination as 'Destination' from dataset_1;
df.rename(columns={'destination':'Destination'},inplace = True)
```

```python
df
```

	Destination	passanger	weather	temperature	time	coupon	expiration
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h

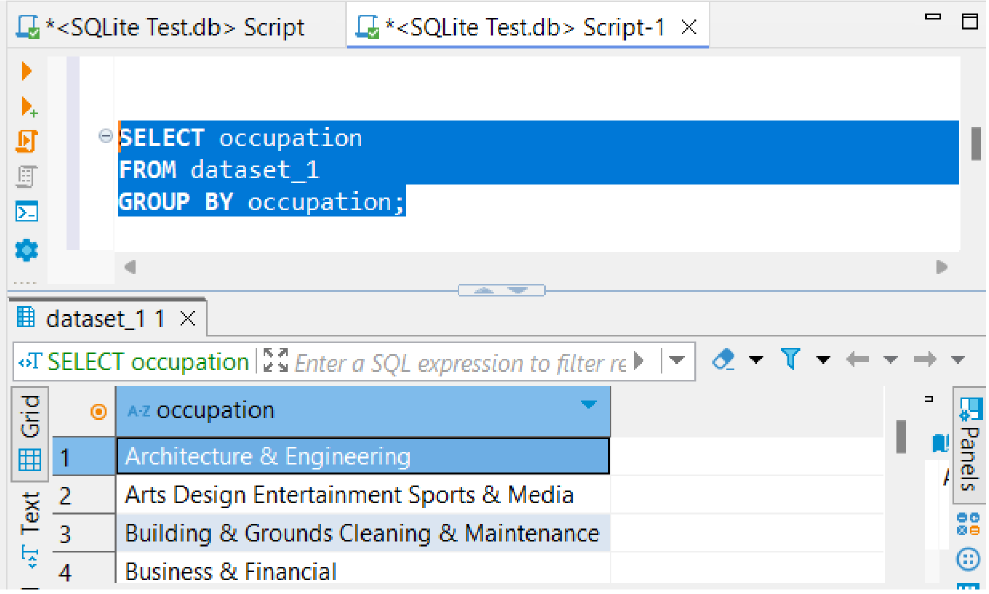
8) In SQL:

Using GroupBy on attribute

SELECT occupation

FROM dataset_1

GROUP BY occupation;



*<SQLite Test.db> Script
*<SQLite Test.db> Script-1
dataset_1 1
Enter a SQL expression to filter results
Grid
Text
Panels

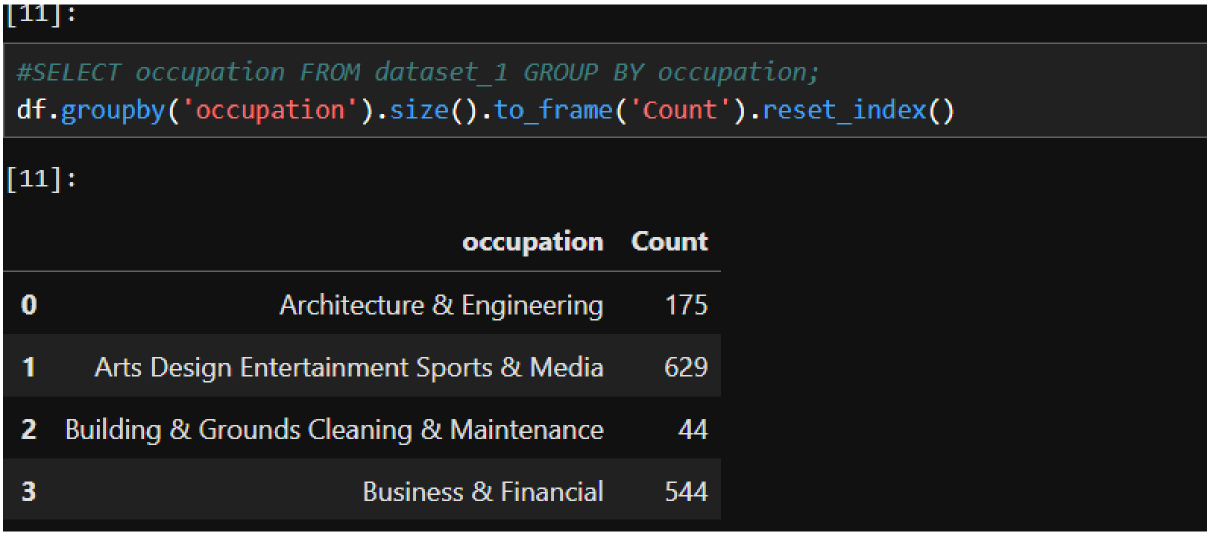
```sql
SELECT occupation
FROM dataset_1
GROUP BY occupation;
```

SELECT occupation

	occupation
1	Architecture & Engineering
2	Arts Design Entertainment Sports & Media
3	Building & Grounds Cleaning & Maintenance
4	Business & Financial

In Python:

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



```
[11]:
#SELECT occupation FROM dataset_1 GROUP BY occupation;
df.groupby('occupation').size().to_frame('Count').reset_index()
```

[11]:

	occupation	Count
0	Architecture & Engineering	175
1	Arts Design Entertainment Sports & Media	629
2	Building & Grounds Cleaning & Maintenance	44
3	Business & Financial	544

9) In SQL:

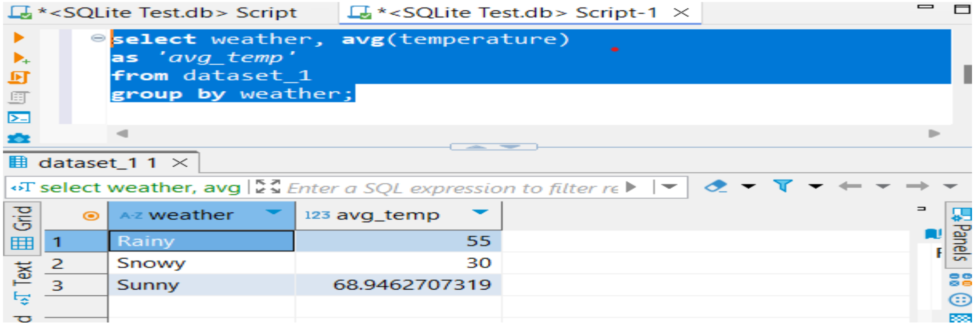
Using group by, for finding the average/mean temp.

select weather, **avg**(temperature)

as *'avg_temp'*

from dataset_1

group by weather;

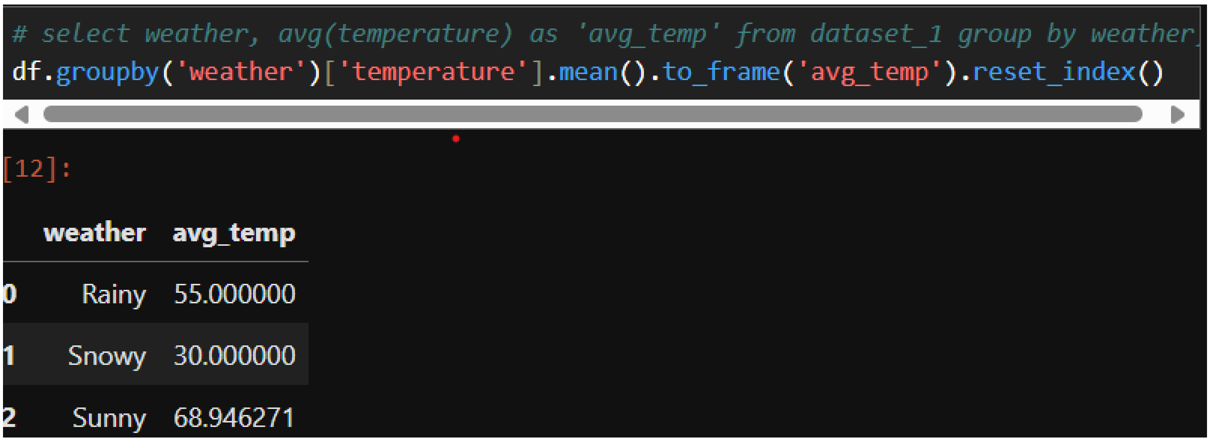


```
select weather, avg(temperature)
as 'avg_temp'
from dataset_1
group by weather;
```

	weather	avg_temp
1	Rainy	55
2	Snowy	30
3	Sunny	68.9462707319

In Python:

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



```
# select weather, avg(temperature) as 'avg_temp' from dataset_1 group by weather
df.groupby('weather')['temperature'].mean().to_frame('avg_temp').reset_index()
```

[12]:

	weather	avg_temp
0	Rainy	55.000000
1	Snowy	30.000000
2	Sunny	68.946271

10) In SQL:

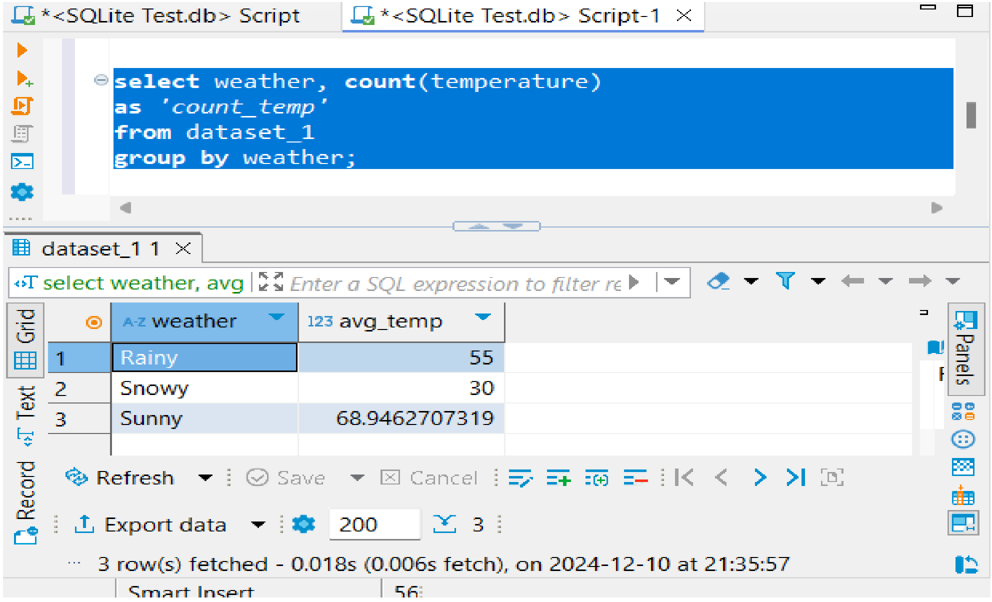
Using group by finding the count value of the attribute

select weather, **count**(temperature)

as *'count_temp'*

from dataset_1

group by weather;



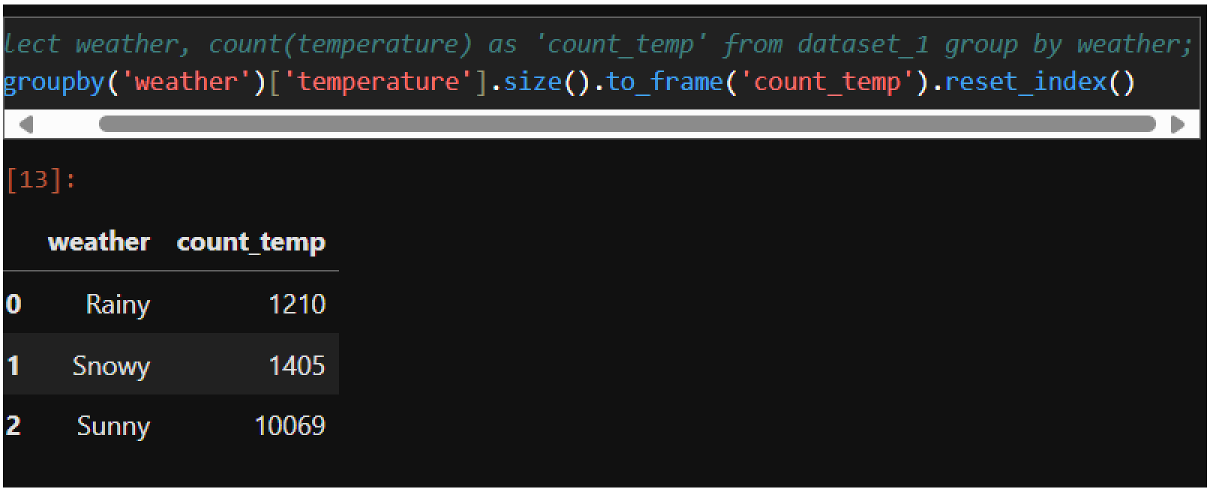
```
select weather, count(temperature)
as 'count_temp'
from dataset_1
group by weather;
```

	weather	avg_temp
1	Rainy	55
2	Snowy	30
3	Sunny	68.9462707319

3 row(s) fetched - 0.018s (0.006s fetch), on 2024-12-10 at 21:35:57

In Python:

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



```
lect weather, count(temperature) as 'count_temp' from dataset_1 group by weather;
groupby('weather')['temperature'].size().to_frame('count_temp').reset_index()
```

[13]:

	weather	count_temp
0	Rainy	1210
1	Snowy	1405
2	Sunny	10069

11) In SQL:

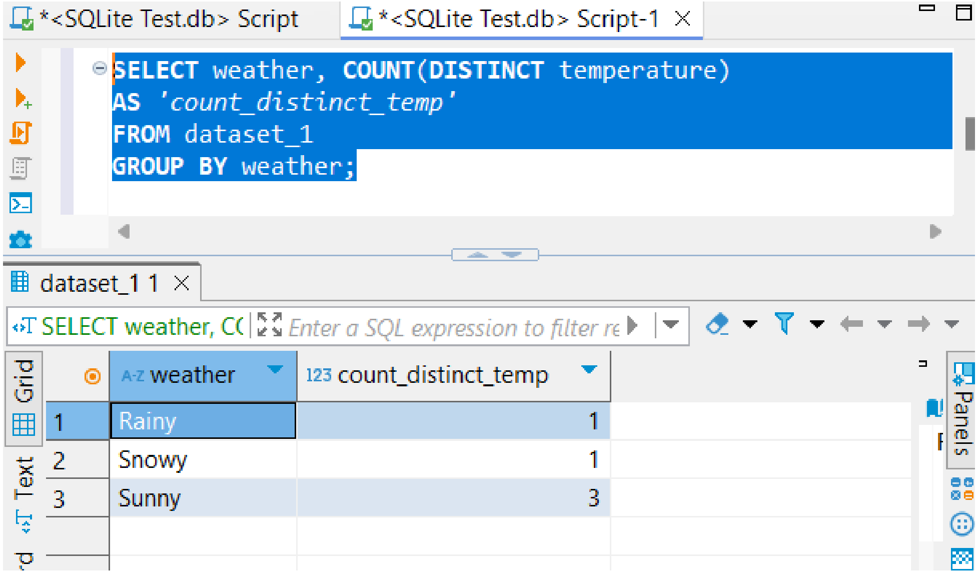
Finding Distinct/unique count of temperature attribute

SELECT weather, **COUNT**(**DISTINCT** temperature)

AS *'count_distinct_temp'*

FROM dataset_1

GROUP BY weather;

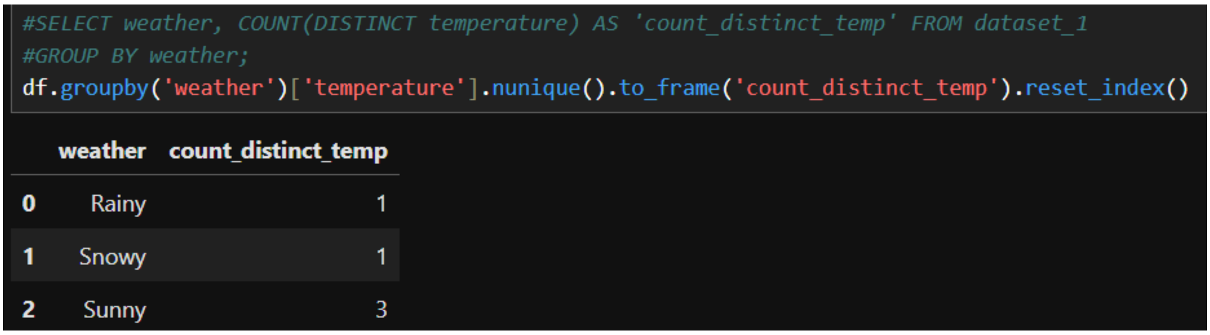


```
SELECT weather, COUNT(DISTINCT temperature)
AS 'count_distinct_temp'
FROM dataset_1
GROUP BY weather;
```

```
   weather  count_distinct_temp
1  Rainy                      1
2  Snowy                      1
3  Sunny                      3
```

In Python:

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;
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
```

12) In SQL:

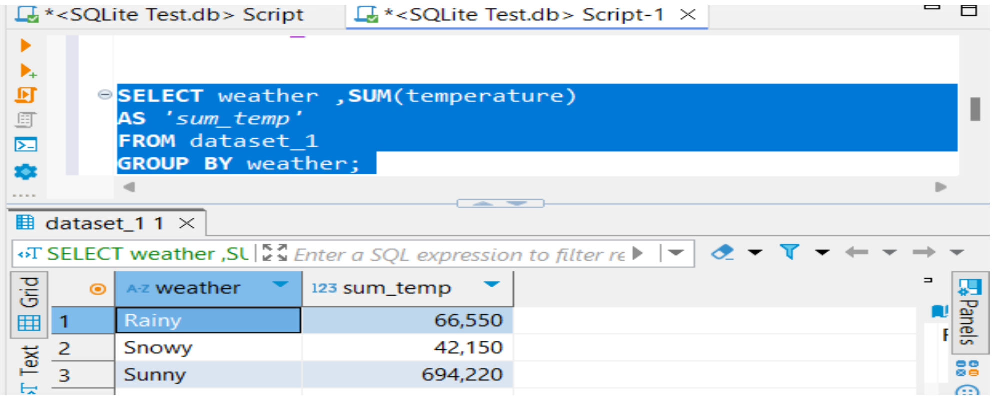
Retrieving the sum of temperature attribute

SELECT weather ,**SUM**(temperature)

AS *'sum_temp'*

FROM dataset_1

GROUP BY weather;

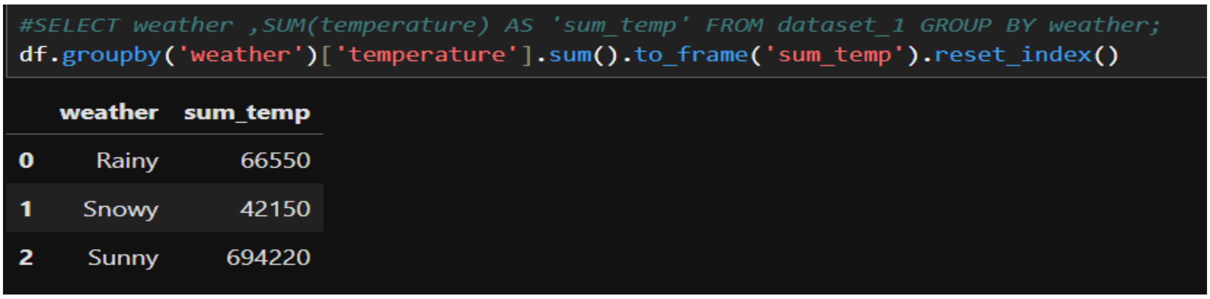


```sql
SELECT weather ,SUM(temperature)
AS 'sum_temp'
FROM dataset_1
GROUP BY weather;
```

	weather	sum_temp
1	Rainy	66,550
2	Snowy	42,150
3	Sunny	694,220

In Python:

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



```python
#SELECT weather ,SUM(temperature) AS 'sum_temp' FROM dataset_1 GROUP BY weather;
df.groupby('weather')['temperature'].sum().to_frame('sum_temp').reset_index()
```

	weather	sum_temp
0	Rainy	66550
1	Snowy	42150
2	Sunny	694220

13) In SQL:

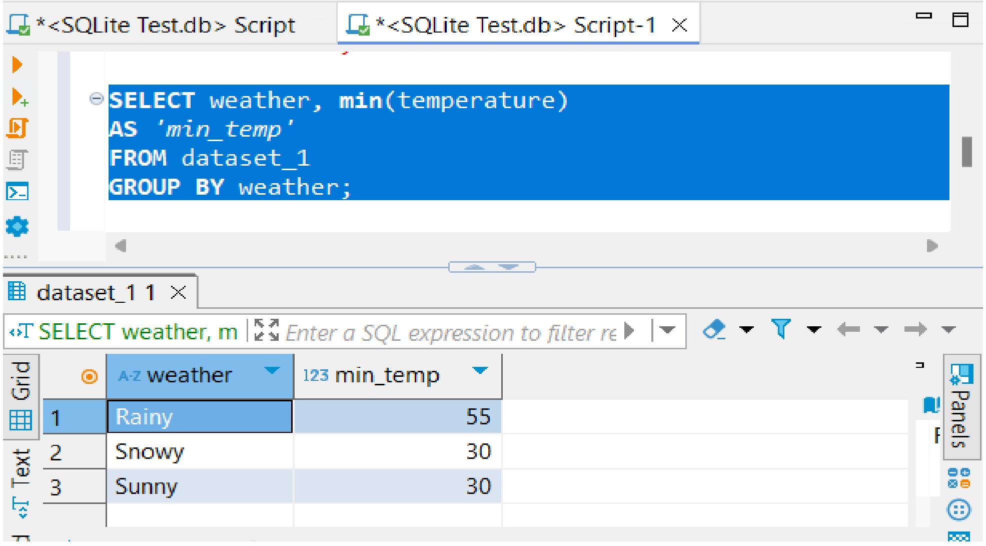
Retrieving the minimum temperature

SELECT weather, **min**(temperature)

AS *'min_temp'*

FROM dataset_1

GROUP BY weather;

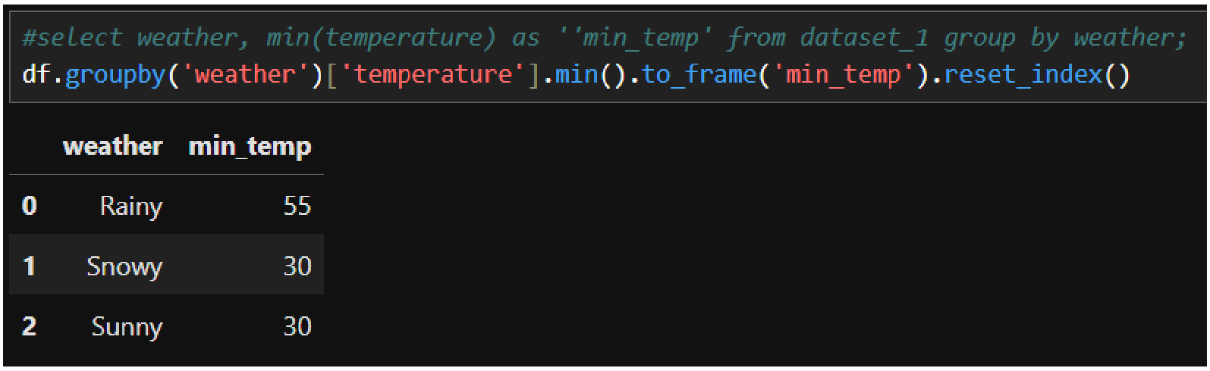


```sql
SELECT weather, min(temperature)
AS 'min_temp'
FROM dataset_1
GROUP BY weather;
```

	weather	min_temp
1	Rainy	55
2	Snowy	30
3	Sunny	30

In Python:

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



```
#select weather, min(temperature) as ''min_temp' from dataset_1 group by weather;
df.groupby('weather')['temperature'].min().to_frame('min_temp').reset_index()
```

```
   weather  min_temp
0    Rainy        55
1    Snowy        30
2    Sunny        30
```

14) In SQL:

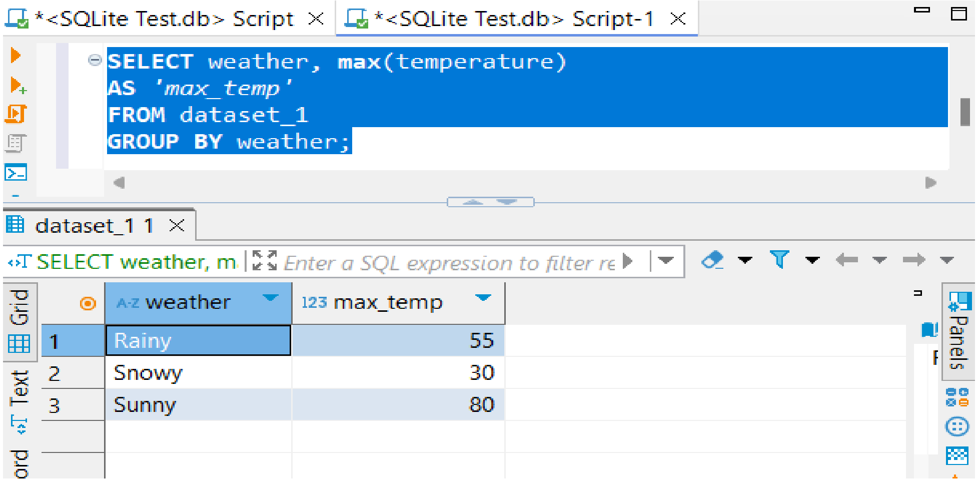
Retrieving the maximum temperature

SELECT weather, **max**(temperature)

AS *'max_temp'*

FROM dataset_1

GROUP BY weather;



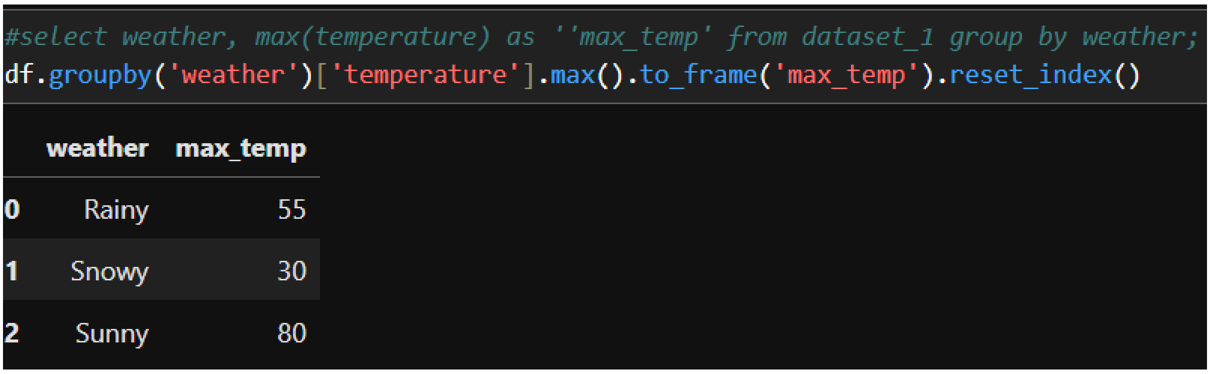
*<SQLite Test.db> Script ×
*<SQLite Test.db> Script-1 ×
dataset_1 1 ×
SELECT weather, m
Enter a SQL expression to filter re
Grid
Text
Panels

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

	weather	max_temp
1	Rainy	55
2	Snowy	30
3	Sunny	80

In Python:

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



```
#select weather, max(temperature) as ''max_temp' from dataset_1 group by weather;
df.groupby('weather')['temperature'].max().to_frame('max_temp').reset_index()
```

```
   weather  max_temp
0    Rainy        55
1    Snowy        30
2    Sunny        80
```

15) In SQL:

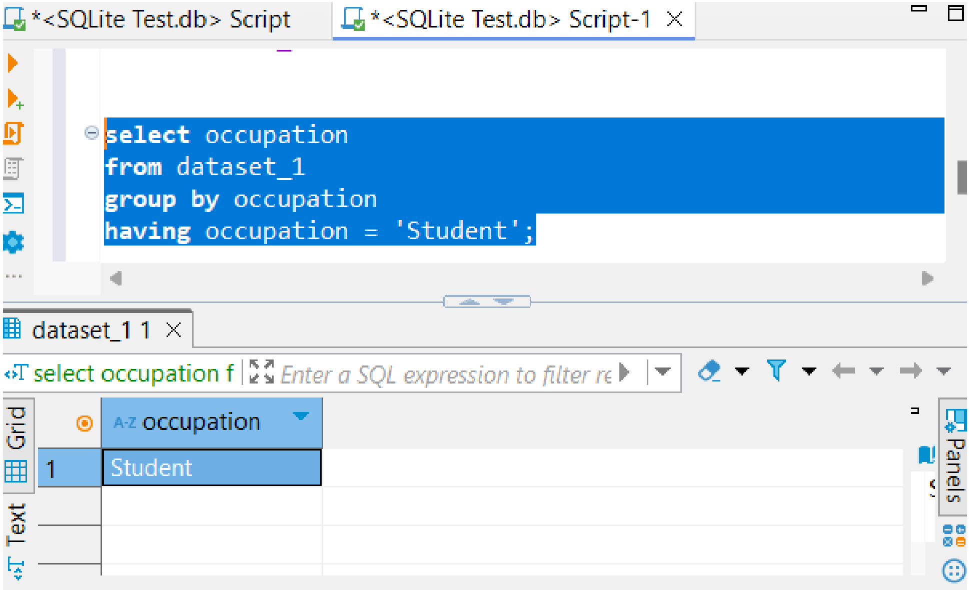
Using groupby and having clause

select occupation

from dataset_1

group by occupation

having occupation = 'Student';

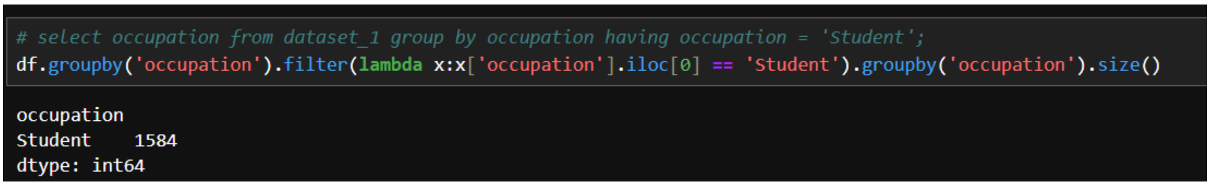


```
select occupation
from dataset_1
group by occupation
having occupation = 'Student';
```

	occupation
1	Student

In Python:

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

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

16) In SQL:

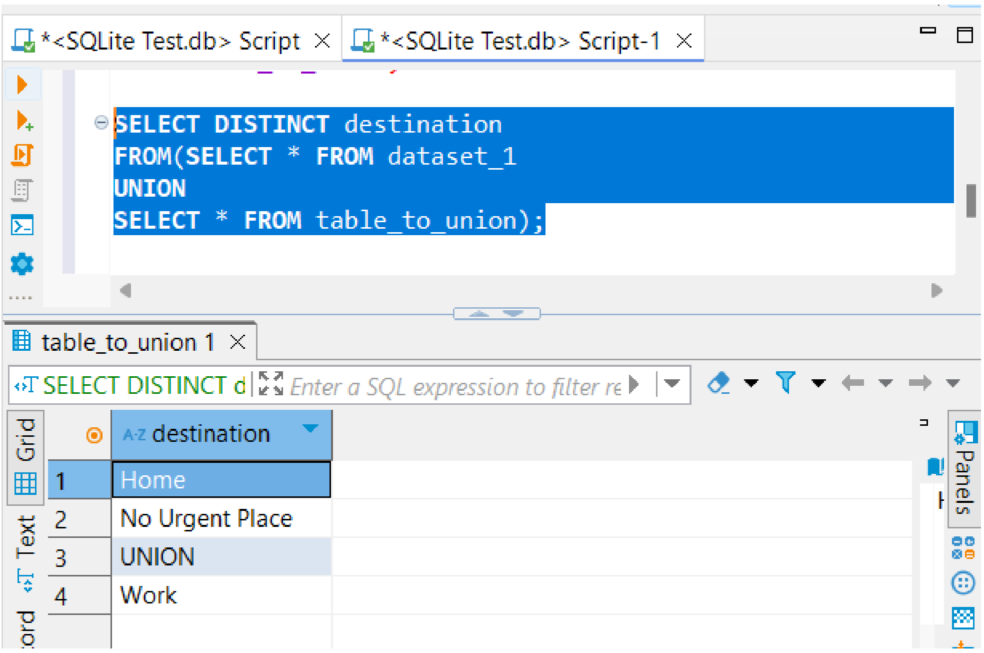
Selecting distinct destination attribute values from 2 tables using UNION

SELECT DISTINCT destination

FROM(SELECT * FROM dataset_1

UNION

SELECT * FROM table_to_union);



```
SELECT DISTINCT destination
FROM(SELECT * FROM dataset_1
UNION
SELECT * FROM table_to_union);
```

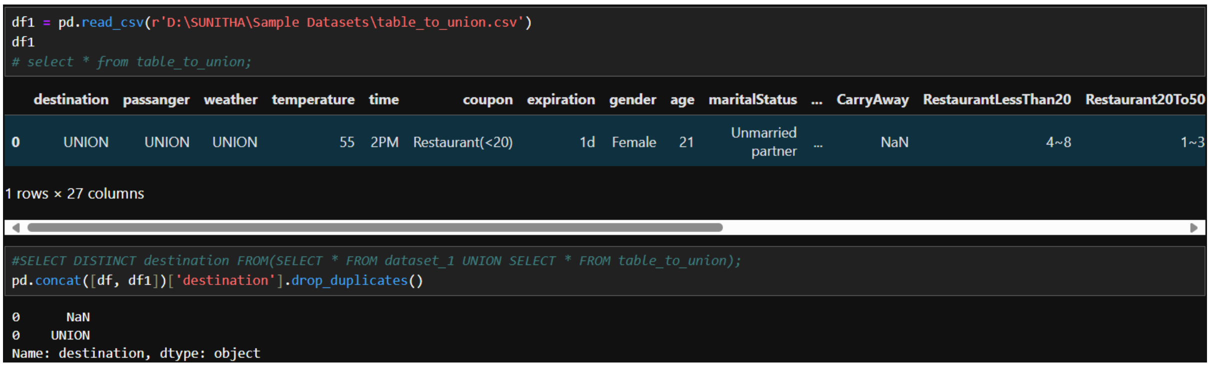
	destination
1	Home
2	No Urgent Place
3	UNION
4	Work

In Python:

df1 = pd.read_csv(r'D:\SUNITHA\Sample Datasets\table_to_union.csv')

df1

pd.concat([df, df1])['destination'].drop_duplicates()



```
df1 = pd.read_csv(r'D:\SUNITHA\Sample Datasets\table_to_union.csv')
df1
# select * from table_to_union;
```

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CarryAway	RestaurantLessThan20	Restaurant20To50
0	UNION	UNION	UNION	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	NaN	4~8	1~3

1 rows × 27 columns

```
#SELECT DISTINCT destination FROM(SELECT * FROM dataset_1 UNION SELECT * FROM table_to_union);
pd.concat([df, df1])['destination'].drop_duplicates()
```

```
0      NaN
0    UNION
Name: destination, dtype: object
```

17) In SQL:

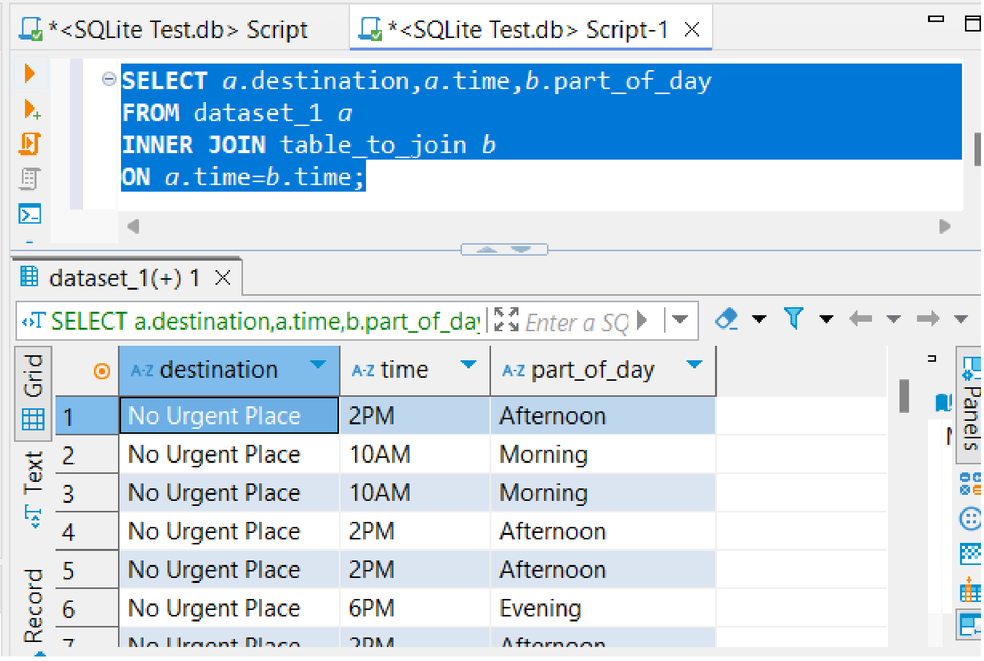
Inner Join

SELECT *a*.destination,*a*.time,*b*.part_of_day

FROM dataset_1 *a*

INNER JOIN table_to_join *b*

ON *a*.time=*b*.time;



```
SELECT a.destination,a.time,b.part_of_day
FROM dataset_1 a
INNER JOIN table_to_join b
ON a.time=b.time;
```

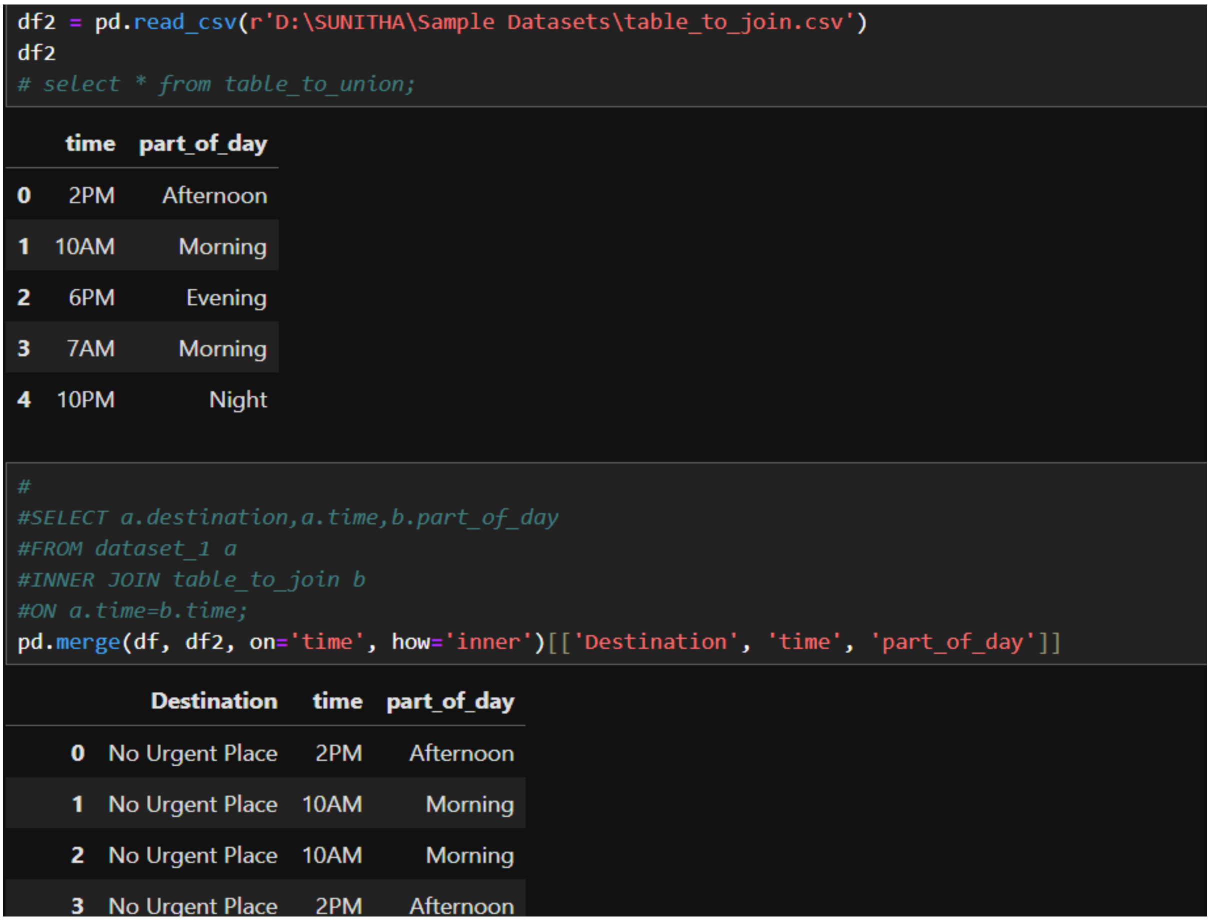
	destination	time	part_of_day
1	No Urgent Place	2PM	Afternoon
2	No Urgent Place	10AM	Morning
3	No Urgent Place	10AM	Morning
4	No Urgent Place	2PM	Afternoon
5	No Urgent Place	2PM	Afternoon
6	No Urgent Place	6PM	Evening
7	No Urgent Place	2PM	Afternoon

In Python:

df2 = pd.read_csv(r'D:\SUNITHA\Sample Datasets\table_to_join.csv')

df2

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



```
df2 = pd.read_csv(r'D:\SUNITHA\Sample Datasets\table_to_join.csv')
df2
# select * from table_to_union;
```

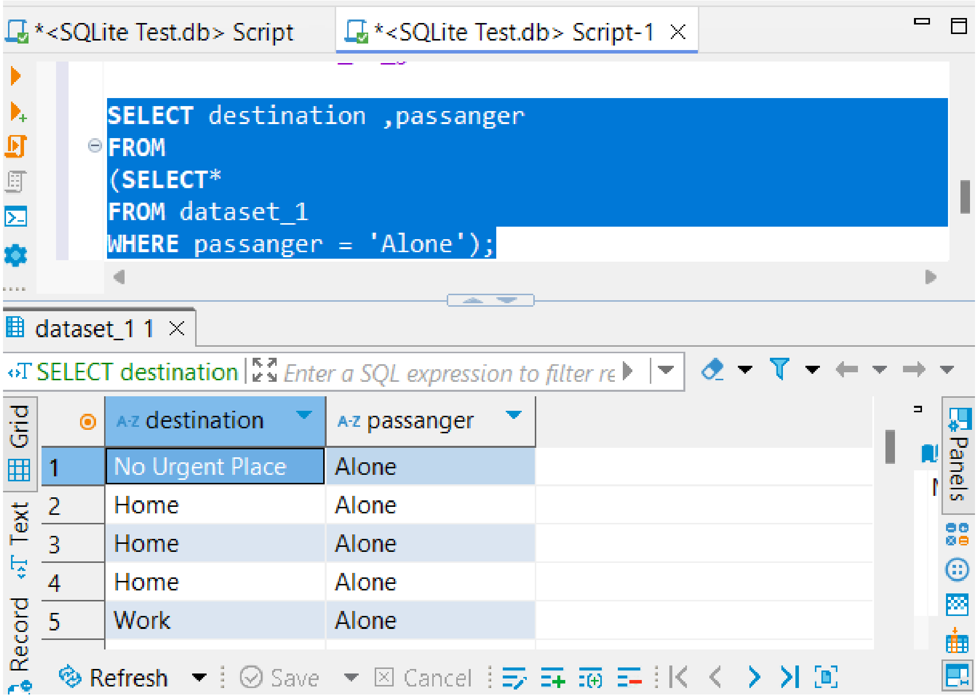
```
   time  part_of_day
0   2PM    Afternoon
1  10AM      Morning
2   6PM      Evening
3   7AM      Morning
4  10PM        Night
```

```
#
#SELECT a.destination,a.time,b.part_of_day
#FROM dataset_1 a
#INNER JOIN table_to_join b
#ON a.time=b.time;
pd.merge(df, df2, on='time', how='inner')[['Destination', 'time', 'part_of_day']]
```

```
       Destination  time  part_of_day
0  No Urgent Place   2PM    Afternoon
1  No Urgent Place  10AM      Morning
2  No Urgent Place  10AM      Morning
3  No Urgent Place   2PM    Afternoon
```

18) In SQL:

```
    SELECT destination ,passanger
    FROM
    (SELECT*
    FROM dataset_1
    WHERE passanger = 'Alone');
```

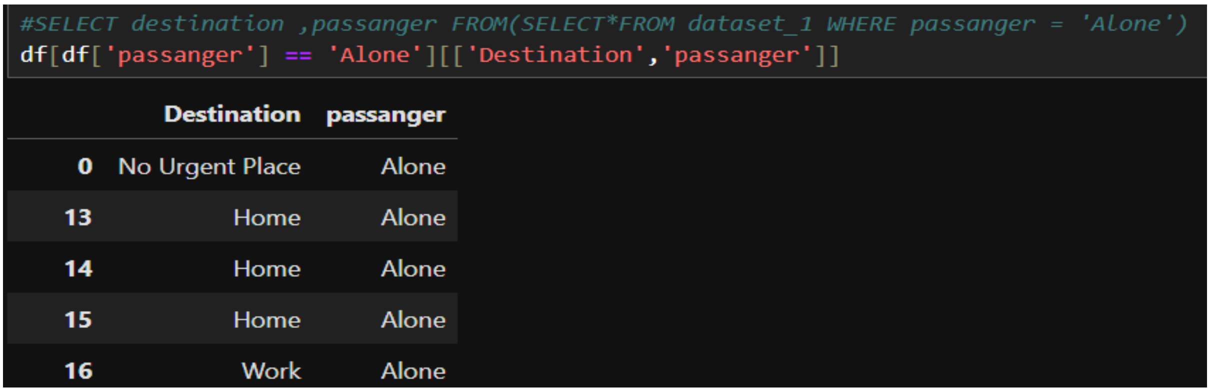


```
SELECT destination ,passanger
FROM
(SELECT*
FROM dataset_1
WHERE passanger = 'Alone');
```

	destination	passanger
1	No Urgent Place	Alone
2	Home	Alone
3	Home	Alone
4	Home	Alone
5	Work	Alone

In Python:

df[df['passanger'] == 'Alone'][['Destination','passanger']]



```
#SELECT destination ,passanger FROM(SELECT*FROM dataset_1 WHERE passanger = 'Alone')
df[df['passanger'] == 'Alone'][['Destination','passanger']]
```

```
        Destination    passanger
 0  No Urgent Place        Alone
13             Home        Alone
14             Home        Alone
15             Home        Alone
16             Work        Alone
```

19) In SQL:

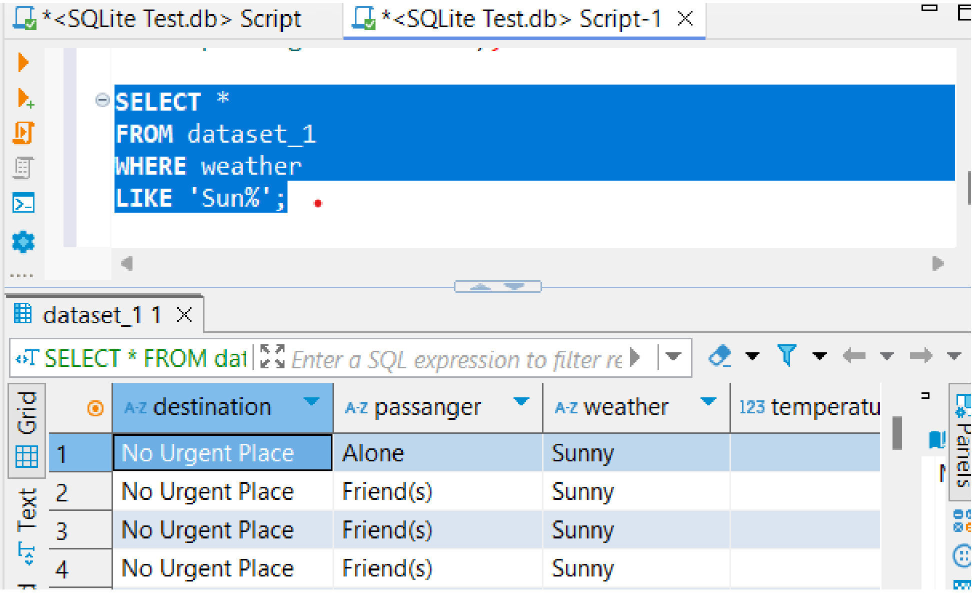
Retrieving the values where weather attribute has starting value with ‘Sun’

SELECT *

FROM dataset_1

WHERE weather

LIKE 'Sun%';

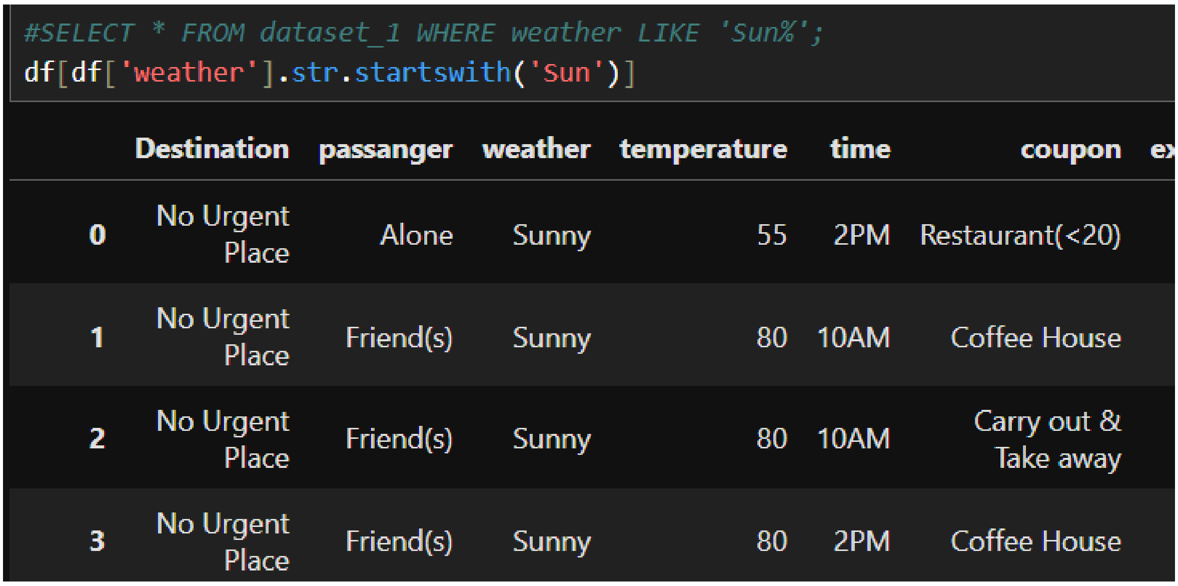


```sql
SELECT *
FROM dataset_1
WHERE weather
LIKE 'Sun%';
```

	destination	passanger	weather	temperatu
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	

In Python:

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



```python
#SELECT * FROM dataset_1 WHERE weather LIKE 'Sun%';
df[df['weather'].str.startswith('Sun')]
```

	Destination	passanger	weather	temperature	time	coupon	ex
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	

20) In SQL:

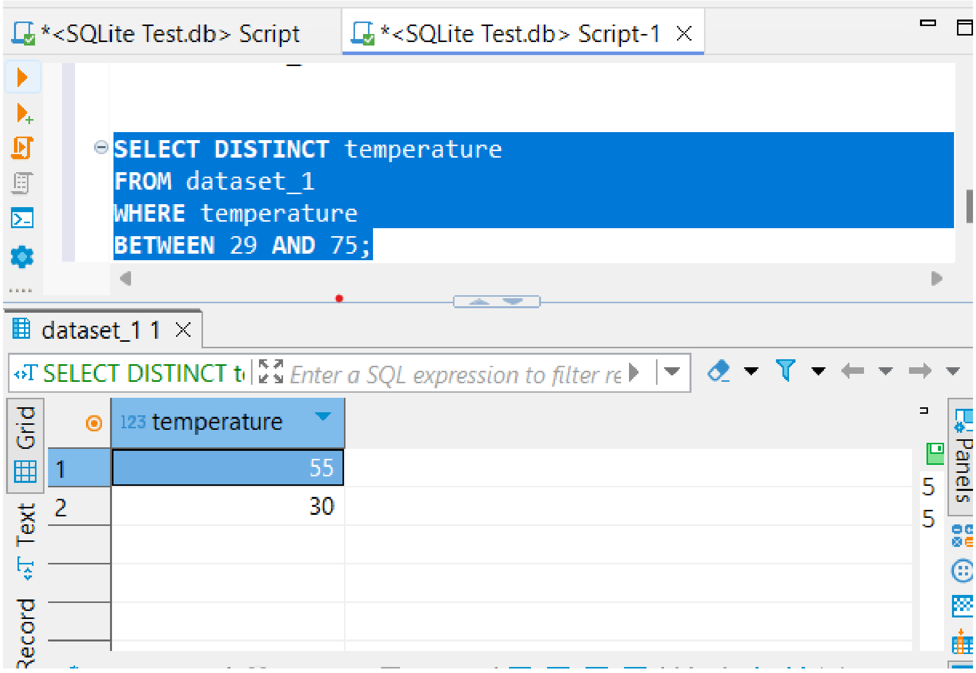
Retrieving distinct temperature values between 29 and 75

SELECT DISTINCT temperature

FROM dataset_1

WHERE temperature

BETWEEN 29 **AND** 75;


*<SQLite Test.db> Script
*<SQLite Test.db> Script-1
dataset_1 1
Enter a SQL expression to filter re
Grid
Text
Record
Panels

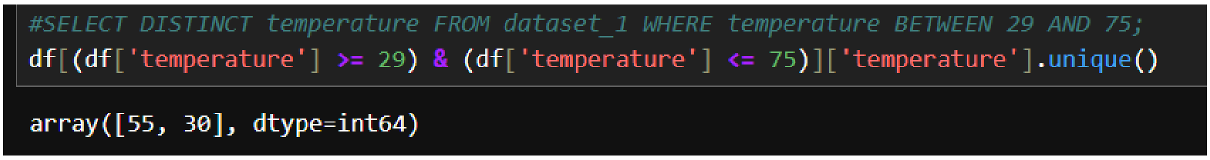
```
SELECT DISTINCT temperature
FROM dataset_1
WHERE temperature
BETWEEN 29 AND 75;
```

SELECT DISTINCT t

	temperature
1	55
2	30

In Python:

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

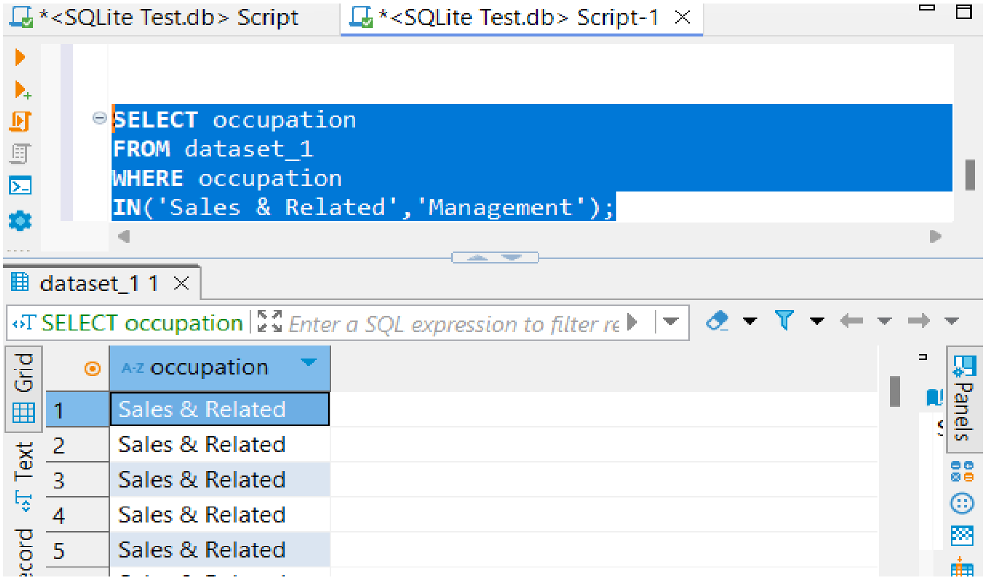


```
#SELECT DISTINCT temperature FROM dataset_1 WHERE temperature BETWEEN 29 AND 75;
df[(df['temperature'] >= 29) & (df['temperature'] <= 75)]['temperature'].unique()
```

```
array([55, 30], dtype=int64)
```

21) In SQL:

```
SELECT occupation
FROM dataset_1
WHERE occupation
IN('Sales & Related','Management');
```


*<SQLite Test.db> Script
*<SQLite Test.db> Script-1
dataset_1 1
Enter a SQL expression to filter re
Grid
Text
Panels

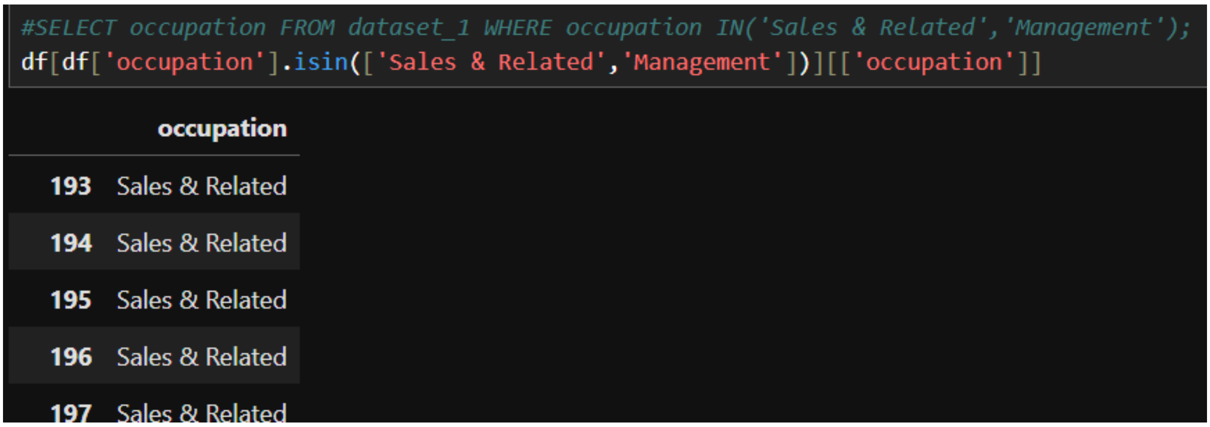
```
SELECT occupation
FROM dataset_1
WHERE occupation
IN('Sales & Related','Management');
```

SELECT occupation

	occupation
1	Sales & Related
2	Sales & Related
3	Sales & Related
4	Sales & Related
5	Sales & Related

In Python:

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



```
#SELECT occupation FROM dataset_1 WHERE occupation IN('Sales & Related','Management');
df[df['occupation'].isin(['Sales & Related','Management'])][['occupation']]
```

```
          occupation
193  Sales & Related
194  Sales & Related
195  Sales & Related
196  Sales & Related
197  Sales & Related
```