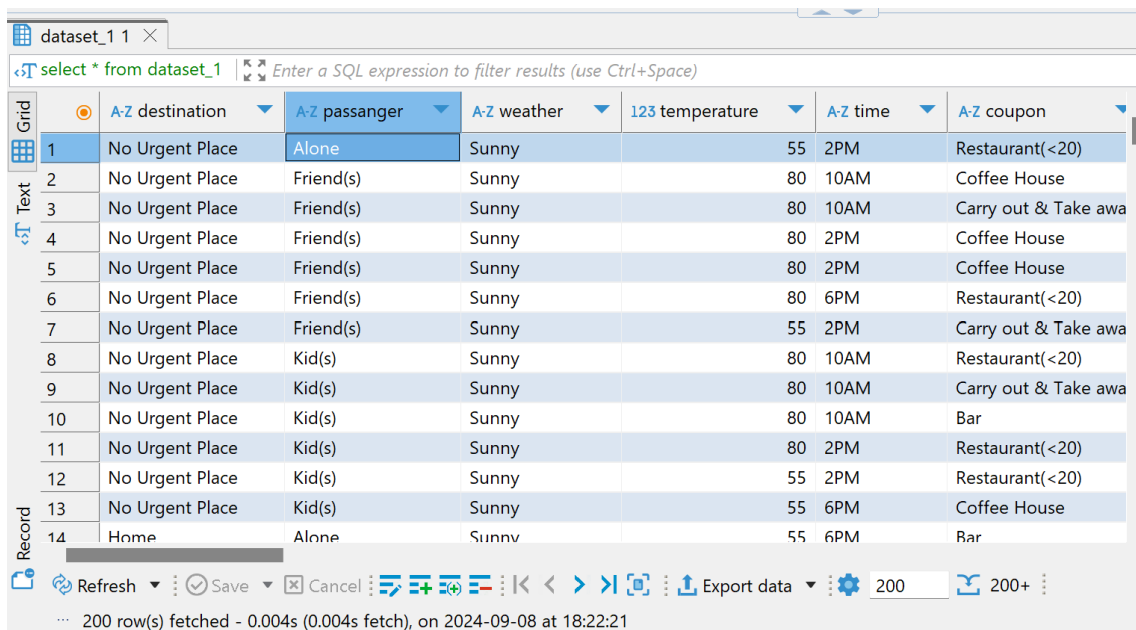


Data Analysis using SQL vs

Data Analysis using Python

SQL Query

```
select * from dataset_1 ;
```



	A-Z destination	A-Z passanger	A-Z weather	123 temperature	A-Z time	A-Z coupon
1	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House
3	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take awa
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House
5	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House
6	No Urgent Place	Friend(s)	Sunny	80	6PM	Restaurant(<20)
7	No Urgent Place	Friend(s)	Sunny	55	2PM	Carry out & Take awa
8	No Urgent Place	Kid(s)	Sunny	80	10AM	Restaurant(<20)
9	No Urgent Place	Kid(s)	Sunny	80	10AM	Carry out & Take awa
10	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar
11	No Urgent Place	Kid(s)	Sunny	80	2PM	Restaurant(<20)
12	No Urgent Place	Kid(s)	Sunny	55	2PM	Restaurant(<20)
13	No Urgent Place	Kid(s)	Sunny	55	6PM	Coffee House
14	Home	Alone	Sunny	55	6PM	Bar

Python Code

df

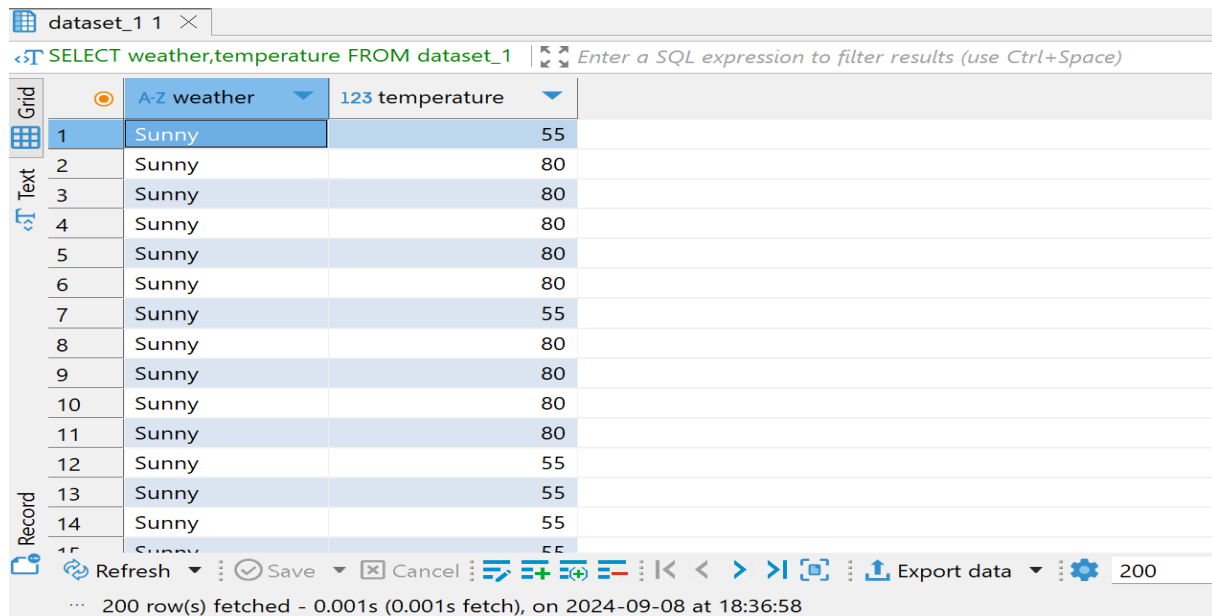
```
[1]: #To read the file
[2]: import pandas as pd
[5]: df=pd.read_csv(r'C:\Users\arati\OneDrive\Desktop\SQL\data.csv')
[6]: df
```

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CarryAway	RestaurantLessThan20	Restauran
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	NaN	4~8	
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Unmarried partner	...	NaN	4~8	
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	Unmarried partner	...	NaN	4~8	
...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	1~3	4~8	
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d	Male	26	Single	...	1~3	4~8	
12681	Work	Alone	Snowy	30	7AM	Coffee House	1d	Male	26	Single	...	1~3	4~8	
12682	Work	Alone	Snowy	30	7AM	Bar	1d	Male	26	Single	...	1~3	4~8	
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)	2h	Male	26	Single	...	1~3	4~8	

12684 rows x 27 columns

SQL Query

SELECT weather,temperature FROM dataset_1



The screenshot shows a data table interface with a tab labeled 'dataset_1 1'. The SQL query 'SELECT weather,temperature FROM dataset_1' is entered in the top bar. The table has two columns: 'weather' and 'temperature'. The first 14 rows are visible, all with 'Sunny' weather and temperatures ranging from 55 to 80. The interface includes a 'Grid' view selector, a 'Text' input area, and a 'Record' view selector. The bottom status bar indicates '200 row(s) fetched - 0.001s (0.001s fetch), on 2024-09-08 at 18:36:58'.

	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
14	Sunny	55

Python Code

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

```
[7]: df[['weather','temperature']] #Select wether, temperature from dataset_1
```

```
[7]:
```

	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 Query

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

dataset_1

<T SELECT * FROM dataset_1 LIMIT 10 Enter a SQL expression to filter results (use Ctrl+Space)

A-Z destination	A-Z passanger	A-Z weather	123 temperature	A-Z time	A-Z coupon
No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)
No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House
No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away
No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House
No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House
No Urgent Place	Friend(s)	Sunny	80	6PM	Restaurant(<20)
No Urgent Place	Friend(s)	Sunny	55	2PM	Carry out & Take away
No Urgent Place	Kid(s)	Sunny	80	10AM	Restaurant(<20)
No Urgent Place	Kid(s)	Sunny	80	10AM	Carry out & Take away
No Urgent Place	Kid(s)	Sunny	80	10AM	Bar

Python Code

```
Df[['weather','temperature']]
```

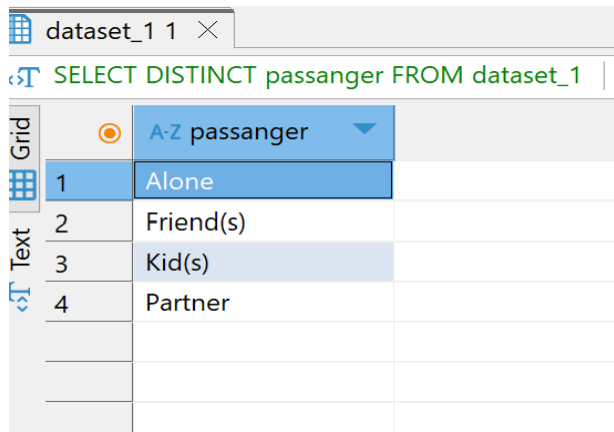
df.head(10) #To see first 10 rows

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CarryAway	RestaurantLessThan20	Restaurant20To50
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	NaN	4~8	1~
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	1~
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Unmarried partner	...	NaN	4~8	1~
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	1~
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	Unmarried partner	...	NaN	4~8	1~
5	No Urgent Place	Friend(s)	Sunny	80	6PM	Restaurant(<20)	2h	Female	21	Unmarried partner	...	NaN	4~8	1~
6	No Urgent Place	Friend(s)	Sunny	55	2PM	Carry out & Take away	1d	Female	21	Unmarried partner	...	NaN	4~8	1~
7	No Urgent Place	Kid(s)	Sunny	80	10AM	Restaurant(<20)	2h	Female	21	Unmarried partner	...	NaN	4~8	1~
8	No Urgent Place	Kid(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Unmarried partner	...	NaN	4~8	1~
9	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar	1d	Female	21	Unmarried partner	...	NaN	4~8	1~

10 rows × 27 columns

SQL Query

```
SELECT DISTINCT passenger FROM dataset_1
```



The screenshot shows a data visualization tool interface. At the top, a tab is labeled 'dataset_1 1'. Below it, a SQL query is entered: 'SELECT DISTINCT passenger FROM dataset_1'. The results are displayed in a 'Grid' view. On the left, there is a sidebar with icons for 'Grid' (selected) and 'Text'. The main grid has a header row with a radio button and the text 'A-Z passenger'. Below the header, there are four rows of data, each with a number in the first column and a passenger status in the second column.

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

Python Code

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

```
[9]: df['passanger'].unique() #To DISTINCT passengers
```

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

SQL Query

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

dataset_1 1 X

SELECT * FROM dataset_1 WHERE destination | Enter a SQL expression to filter results (use Ctrl+Space)

	A-Z destination	A-Z passanger	A-Z weather	123 temperature	A-Z time	A-Z coupon	A-Z
1	Home	Alone	Sunny	55	6PM	Bar	1
2	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1
3	Home	Alone	Sunny	80	6PM	Coffee House	2
4	Home	Alone	Sunny	55	6PM	Bar	1
5	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1
6	Home	Alone	Sunny	80	6PM	Coffee House	2
7	Home	Alone	Sunny	55	6PM	Bar	1
8	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1
9	Home	Alone	Sunny	80	6PM	Coffee House	2
10	Home	Alone	Sunny	55	6PM	Bar	1
11	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1
12	Home	Alone	Sunny	80	6PM	Coffee House	2
13	Home	Alone	Sunny	55	6PM	Bar	1
14	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1
15	Home	Alone	Sunny	80	6PM	Coffee House	2

Python Code

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

```
[10]: df[df['destination']=='Home'] #To see where destination == Home
```

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CarryAway	RestaurantLessThan20	Restaurant
13	Home	Alone	Sunny	55	6PM	Bar	1d	Female	21	Unmarried partner	...	NaN	4~8	
14	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1d	Female	21	Unmarried partner	...	NaN	4~8	
15	Home	Alone	Sunny	80	6PM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	
35	Home	Alone	Sunny	55	6PM	Bar	1d	Male	21	Single	...	4~8	4~8	
36	Home	Alone	Sunny	55	6PM	Restaurant(20-50)	1d	Male	21	Single	...	4~8	4~8	
...	
12675	Home	Alone	Snowy	30	10PM	Coffee House	2h	Male	26	Single	...	1~3	4~8	
12676	Home	Alone	Sunny	80	6PM	Restaurant(20-50)	1d	Male	26	Single	...	1~3	4~8	
12677	Home	Partner	Sunny	30	6PM	Restaurant(<20)	1d	Male	26	Single	...	1~3	4~8	
12678	Home	Partner	Sunny	30	10PM	Restaurant(<20)	2h	Male	26	Single	...	1~3	4~8	
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	1~3	4~8	

3237 rows x 27 columns

SQL Query

SELECT *FROM dataset_1 ORDER BY coupon

dataset_1 1								
SELECT *FROM dataset_1 ORDER BY coupon								
Grid		A-Z destination	A-Z passanger	A-Z weather	123 temperature	A-Z time	A-Z coupon	A
Text	1	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar	1
	2	Home	Alone	Sunny	55	6PM	Bar	1
	3	Work	Alone	Sunny	55	7AM	Bar	1
	4	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar	1
	5	Home	Alone	Sunny	55	6PM	Bar	1
	6	Work	Alone	Sunny	55	7AM	Bar	1
	7	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar	1
	8	Home	Alone	Sunny	55	6PM	Bar	1
	9	Work	Alone	Sunny	55	7AM	Bar	1
	10	No Urgent Place	Kid(s)	Sunny	80	10AM	Bar	1
	11	Home	Alone	Sunny	55	6PM	Bar	1
	12	Work	Alone	Sunny	55	7AM	Bar	1
	13	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar	1
	14	Home	Alone	Sunny	55	6PM	Bar	1
	15	Work	Alone	Sunny	55	7AM	Bar	1
Record								
Refresh Save Cancel Export data 200 200+								

Python Code

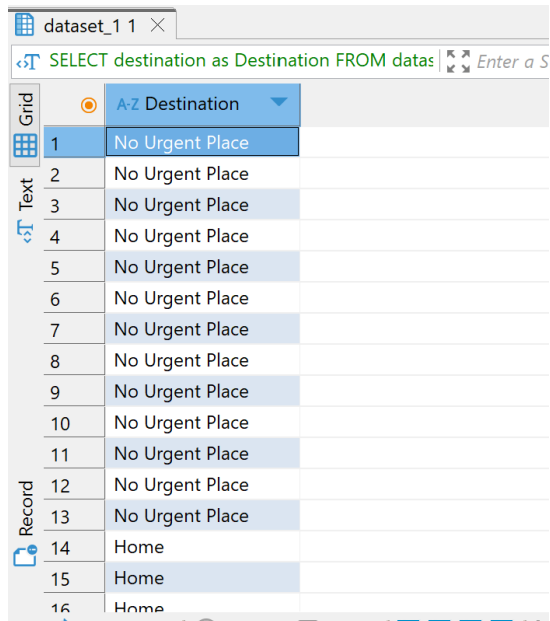
df.sort_values('coupon')

[11]:	df.sort_values('coupon') #Shorting the COUPON value in ascending order													
[11]:	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CarryAway	RestaurantLessThan20	Restaura
	11702	Home	Partner	Sunny	30	10PM	Bar	2h	Female	50plus	Married partner	...	4~8	1~3
	9930	No Urgent Place	Alone	Snowy	30	2PM	Bar	1d	Female	21	Single	...	gt8	gt8
	10632	Home	Alone	Rainy	55	6PM	Bar	1d	Male	21	Single	...	gt8	less1
	7997	No Urgent Place	Friend(s)	Rainy	55	10PM	Bar	2h	Male	26	Unmarried partner	...	4~8	never
	11166	Work	Alone	Snowy	30	7AM	Bar	1d	Female	41	Married partner	...	gt8	1~3

	10476	Home	Alone	Sunny	80	6PM	Restaurant(<20)	1d	Female	31	Unmarried partner	...	1~3	1~3
	5447	Home	Alone	Sunny	80	10PM	Restaurant(<20)	2h	Female	50plus	Single	...	less1	less1
	10478	Home	Alone	Snowy	30	10PM	Restaurant(<20)	2h	Female	31	Unmarried partner	...	1~3	1~3
	5440	No Urgent Place	Alone	Sunny	80	2PM	Restaurant(<20)	2h	Female	50plus	Single	...	less1	less1
	0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	NaN	4~8
12684 rows x 27 columns														

SQL Query

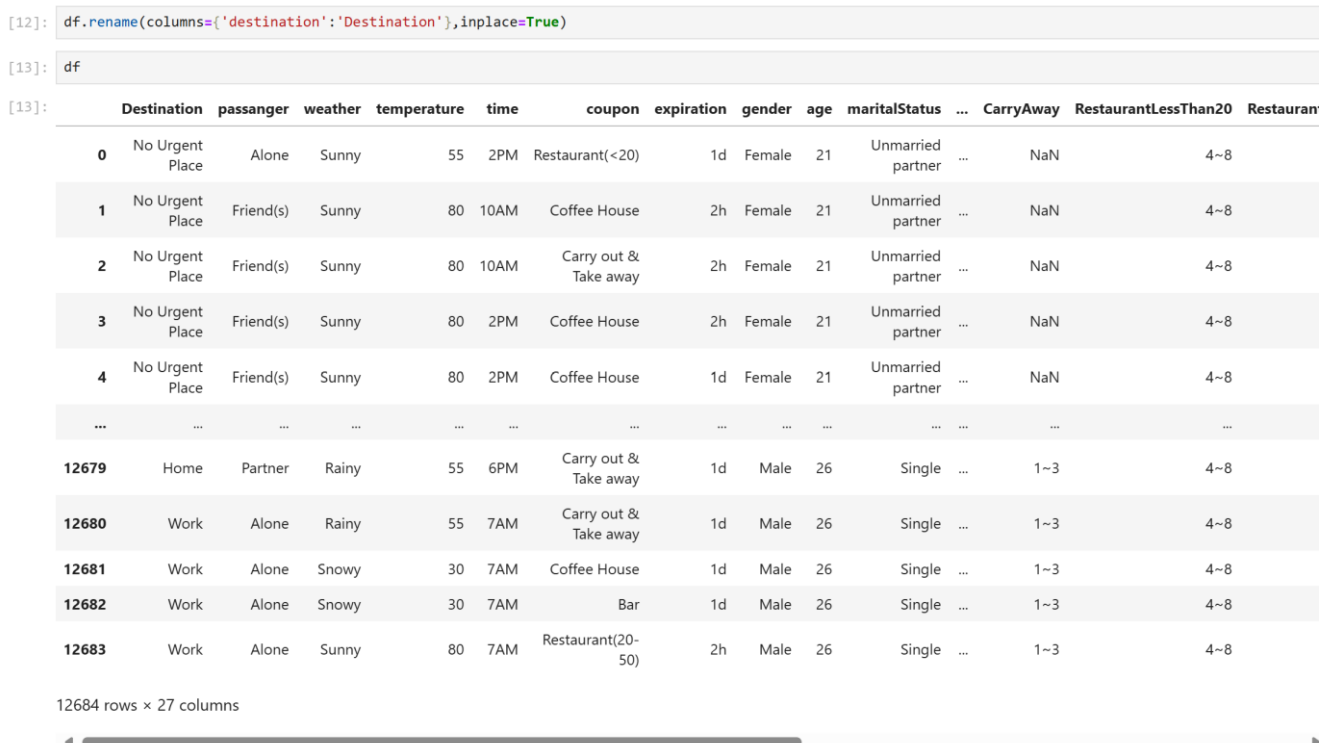
SELECT destination **as** *Destination* **FROM** dataset_1



	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

Python Code

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



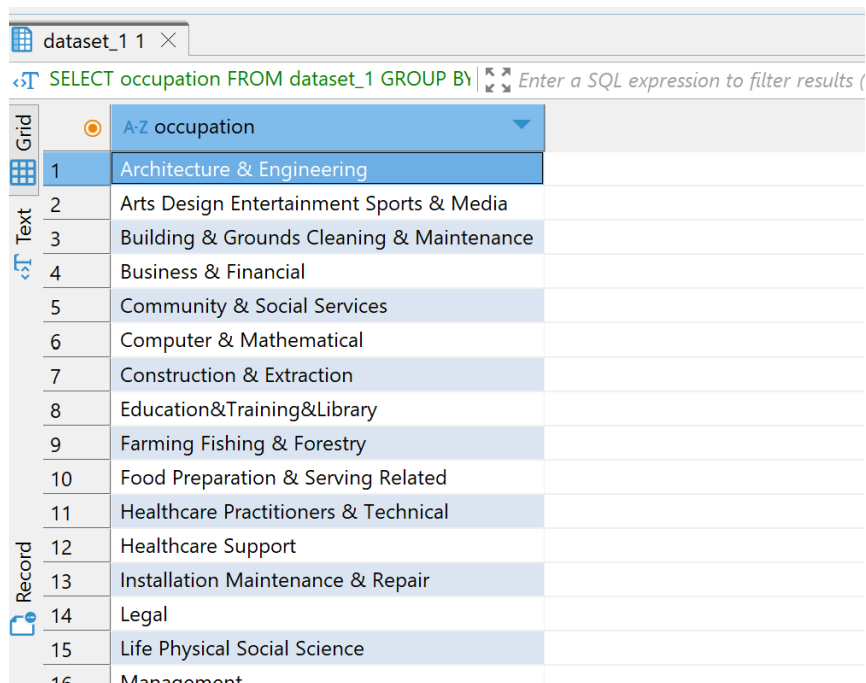
```
[12]: df.rename(columns={'destination':'Destination'},inplace=True)
[13]: df
```

	Destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CarryAway	RestaurantLessThan20	Restaurant
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	NaN	4~8	
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Unmarried partner	...	NaN	4~8	
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	Unmarried partner	...	NaN	4~8	
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	Unmarried partner	...	NaN	4~8	
...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	1~3	4~8	
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d	Male	26	Single	...	1~3	4~8	
12681	Work	Alone	Snowy	30	7AM	Coffee House	1d	Male	26	Single	...	1~3	4~8	
12682	Work	Alone	Snowy	30	7AM	Bar	1d	Male	26	Single	...	1~3	4~8	
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)	2h	Male	26	Single	...	1~3	4~8	

12684 rows × 27 columns

SQL Query

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 Code

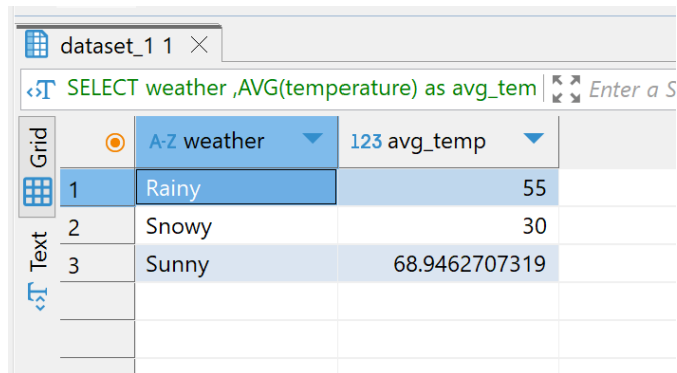
```
df.groupby('occupation').size().to_frame('Count').reset_index()
```

```
[16]: df.groupby('occupation').size().to_frame('Count').reset_index() # Select occupation from dataset_1 GROUPBY occupation
```

	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

SQL Query

SELECT weather ,AVG(temperature) as avg_temp FROM dataset_1 GROUP BY



The screenshot shows a SQL query editor with a tab labeled 'dataset_1 1'. The query entered is 'SELECT weather ,AVG(temperature) as avg_tem'. Below the query, there is a 'Grid' view of the results. The grid has two columns: 'A-Z weather' and '123 avg_temp'. The results are as follows:

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

Python Code

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

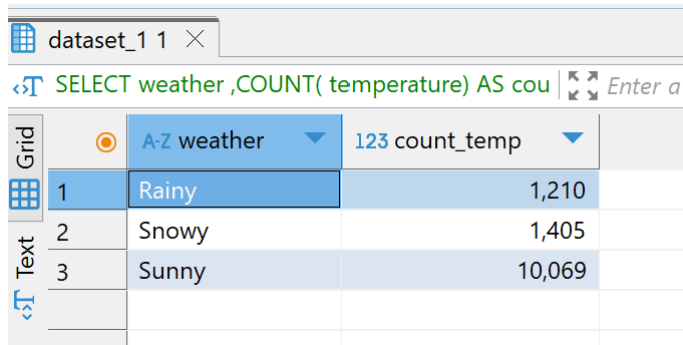
```
[17]: df.groupby('weather')['temperature'].mean().to_frame('avg_temp').reset_index() #To see avg(temperature) as avg_temp from dataset_1 groupby weather
```

```
[17]:
```

	weather	avg_temp
0	Rainy	55.000000
1	Snowy	30.000000
2	Sunny	68.946271

SQL Query

SELECT weather ,**COUNT**(temperature) **AS** count_temp **FROM** dataset_1
GROUP BY weather



The screenshot shows a SQL query editor with a tab labeled 'dataset_1 1'. The query entered is `SELECT weather ,COUNT(temperature) AS cou`. Below the query, there is a table with 3 columns: an index, 'weather', and 'count_temp'. The table contains 3 rows of data: 'Rainy' with count 1,210, 'Snowy' with count 1,405, and 'Sunny' with count 10,069.

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

Python Code

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

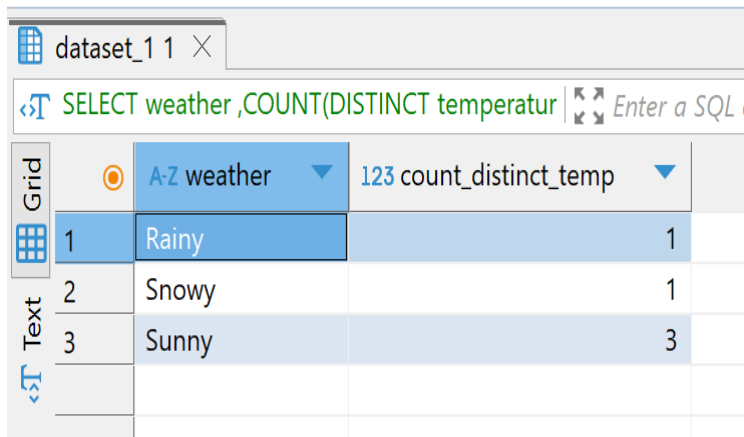
```
[18]: #Select weather count(temperature) as Count_temp from dataset_1 groupby weather
df.groupby('weather')['temperature'].size().to_frame('Count_temp').reset_index()
```

```
[18]:
```

	weather	Count_temp
0	Rainy	1210
1	Snowy	1405
2	Sunny	10069

SQL Query

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



The screenshot shows a SQL query editor with a tab labeled 'dataset_1 1'. The query entered is `SELECT weather ,COUNT(DISTINCT temperature) AS count_distinct_temp`. Below the query, there is a table view with a 'Grid' button and a 'Text' button. The table has two columns: 'A-Z weather' and 'count_distinct_temp'. The data is as follows:

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

Python Code

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

```
[19]: #Select weather count( distinct temperature) as count_distinct_temp from dataset_1 groupby weather  
df.groupby('weather')['temperature'].nunique().to_frame('count_distinct_temp').reset_index()
```

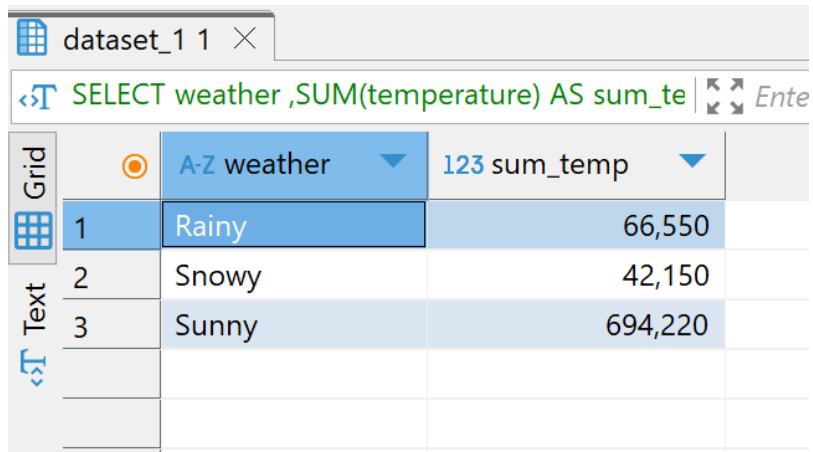
```
[19]:
```

	weather	count_distinct_temp
--	---------	---------------------

0	Rainy	1
1	Snowy	1
2	Sunny	3

SQL Query

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



The screenshot shows a database query interface. At the top, there's a tab labeled 'dataset_1 1'. Below it, the SQL query is entered: `SELECT weather ,SUM(temperature) AS sum_te`. The interface has a 'Grid' view selected, showing a table with 3 columns: an index, 'weather', and 'sum_temp'. The results are as follows:

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

Python Code

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

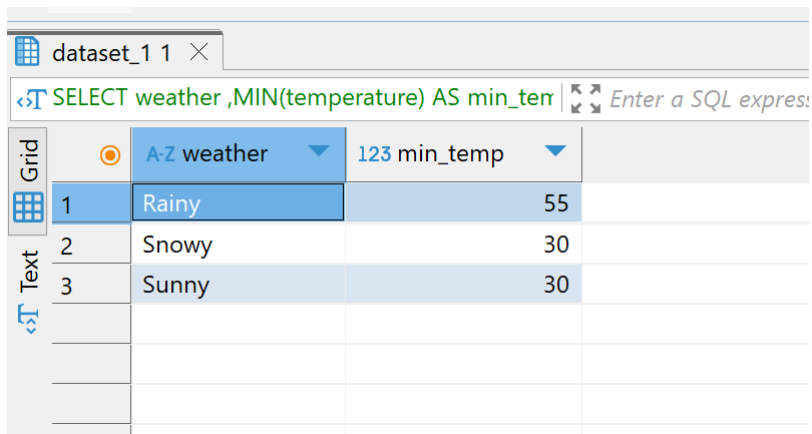
```
[20]: #SELECT weather from SUM(temperature) as sum_temp from database_1 groupby weather
df.groupby('weather')['temperature'].sum().to_frame('sum_temp').reset_index()
```

```
[20]:
```

	weather	sum_temp
0	Rainy	66550
1	Snowy	42150
2	Sunny	694220

SQL Query

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



The screenshot shows a SQL query editor with a tab labeled 'dataset_1 1'. The query entered is `SELECT weather ,MIN(temperature) AS min_temp`. Below the query bar, there are two views: 'Grid' and 'Text'. The 'Grid' view is selected, showing a table with 3 rows and 2 columns. The first row is highlighted in blue and shows 'Rainy' and '55'. The second row shows 'Snowy' and '30'. The third row shows 'Sunny' and '30'.

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

Python Code

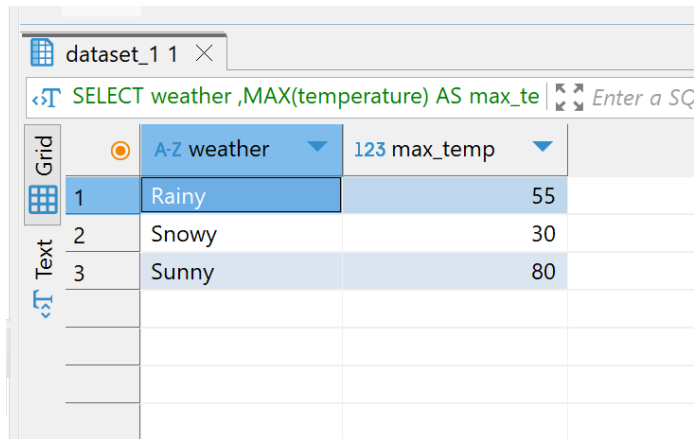
```
df.groupby('weather')['temperature'].min().to_frame('min_temp').reset_index()
```

```
#SELECT weather from MIN(temperature) as min_temp from database_1 groupby weather
df.groupby('weather')['temperature'].min().to_frame('min_temp').reset_index()
```

	weather	min_temp
0	Rainy	55
1	Snowy	30
2	Sunny	30

SQL Query

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



The screenshot shows a SQL query editor with a tab labeled 'dataset_1 1'. The query entered is `SELECT weather ,MAX(temperature) AS max_te`. Below the query, there is a table with 4 columns. The first column is a grid icon, the second is a radio button, the third is 'A-Z weather', and the fourth is '123 max_temp'. The table contains three rows of data: 'Rainy' with a value of 55, 'Snowy' with a value of 30, and 'Sunny' with a value of 80.

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

Python Code

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

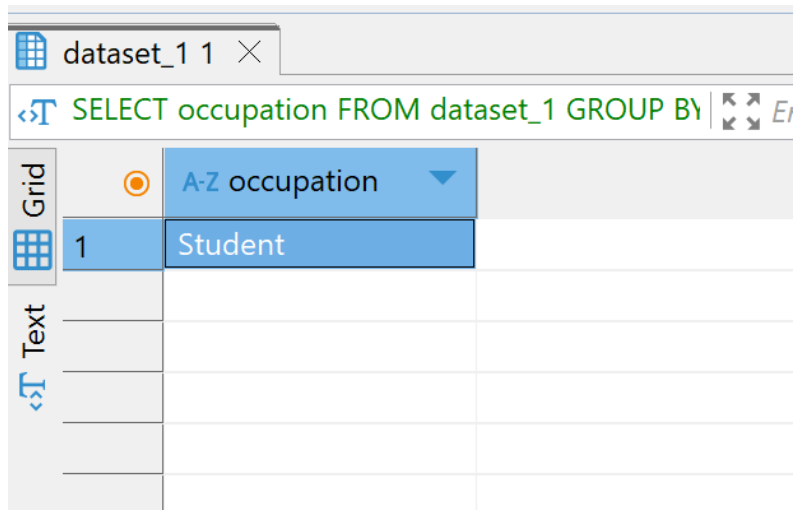
```
[22]: #SELECT weather from MAX(temperature) as max_temp from database_1 groupby weather
df.groupby('weather')['temperature'].max().to_frame('max_temp').reset_index()
```

```
[22]:
```

	weather	max_temp
0	Rainy	55
1	Snowy	30
2	Sunny	80

SQL Query

SELECT occupation **FROM** dataset_1 **GROUP BY** occupation **HAVING** occupation='Student'



The screenshot shows a SQL query editor window titled 'dataset_1 1'. The query bar contains the text 'SELECT occupation FROM dataset_1 GROUP BY'. Below the query bar, there is a 'Grid' view of the results. The grid has a column header 'A-Z occupation' and a single row with the value 'Student'. The 'Text' view is also visible on the left side of the grid.

	A-Z occupation
1	Student

Python Code

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

```
•[25]: #SELECT occupation from dataset_1 groupby OCCUPATION HAVING occupation = student  
df.groupby('occupation').filter(lambda x: x['occupation'].iloc[0] == 'Student').groupby('occupation').size()
```

```
[25]: occupation  
Student    1584  
dtype: int64
```

SQL Query

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

The screenshot shows a data table interface with a tab labeled 'table_to_union 1'. The SQL query 'SELECT DISTINCT destination FROM(SELECT * FROM dataset_1 UNION SELECT * FROM table_to_union)' is entered in the query bar. Below the query bar, there is a grid view of the data. The grid has a column header 'A-Z destination' and four rows of data: 'Home', 'No Urgent Place', 'UNION', and 'Work'. The 'UNION' row is highlighted. A tooltip is visible over the 'UNION' row, displaying 'Column: ma' and 'Read-only: N'.

	A-Z destination
1	Home
2	No Urgent Place
3	UNION
4	Work

Python Code

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

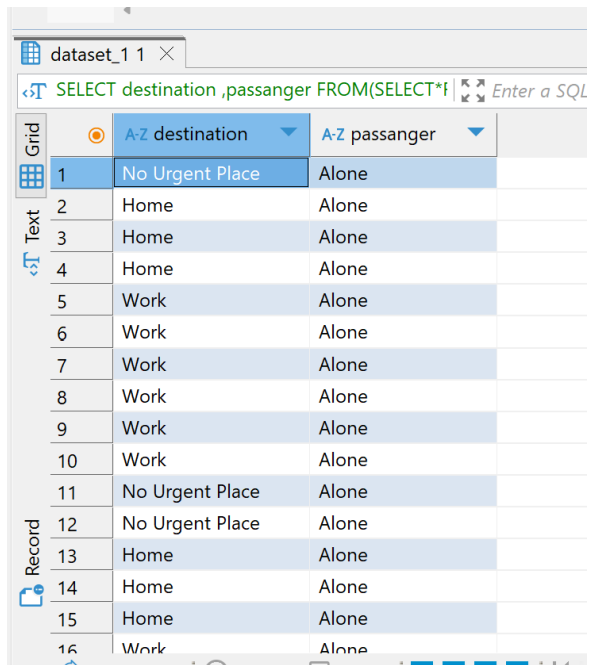
```
[30]: pd.concat([df, df1])['Destination'].drop_duplicates()
```

```
[30]: 0      No Urgent Place
      13             Home
      16             Work
      Name: Destination, dtype: object
```

```
[ 1]:
```


SQL Query

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



The screenshot shows a SQL query interface. At the top, a tab labeled 'dataset_1 1' is open. Below it, a SQL query is entered: `SELECT destination ,passanger FROM(SELECT*f`. Below the query editor, there is a data grid with two columns: 'A-Z destination' and 'A-Z passanger'. The grid displays 16 rows of data, with the first row highlighted in blue. The data is as follows:

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

Python Code

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

```
[42]: #SELECT destination,passangers FROM dataset_1 where passanger=Alone
df[df['passanger'] == 'Alone'][['Destination', 'passanger']]
```

```
[42]:
```

	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

SQL Query

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

dataset_1 1

SELECT * FROM dataset_1 WHERE weather LIKE

Enter a SQL expression to filter results (use C

Grid

A-Z destination

A-Z passanger

A-Z weather

123

temperature

1

No Urgent Place

Alone

Sunny

55

2

No Urgent Place

Friend(s)

Sunny

80

3

No Urgent Place

Friend(s)

Sunny

80

4

No Urgent Place

Friend(s)

Sunny

80

5

No Urgent Place

Friend(s)

Sunny

80

6

No Urgent Place

Friend(s)

Sunny

80

7

No Urgent Place

Friend(s)

Sunny

55

8

No Urgent Place

Kid(s)

Sunny

80

9

No Urgent Place

Kid(s)

Sunny

80

10

No Urgent Place

Kid(s)

Sunny

80

11

No Urgent Place

Kid(s)

Sunny

80

12

No Urgent Place

Kid(s)

Sunny

55

13

No Urgent Place

Kid(s)

Sunny

55

14

Home

Alone

Sunny

55

15

Home

Alone

Sunny

55

Text

SQL

Record

6

Python Code

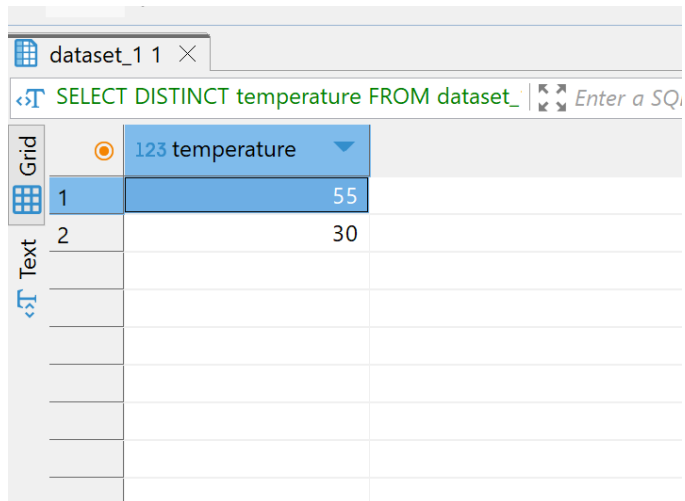
```
df[df['weather'].str.startswith('Sun')]
```

```
3]: #SELECT weather from dataset_1 where weather is sunny|
df[df['weather'].str.startswith('Sun')]
```

	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
...
12673	Home	Alone	Sunny	30	6PM
12676	Home	Alone	Sunny	80	6PM
12677	Home	Partner	Sunny	30	6PM
12678	Home	Partner	Sunny	30	10PM
12683	Work	Alone	Sunny	80	7AM

SQL Query

SELECT DISTINCT temperature **FROM** dataset_1 **WHERE** temperature **BETWEEN** 29 **AND** 75



The screenshot shows a SQL query interface. At the top, there's a tab labeled 'dataset_1 1'. Below it, the query 'SELECT DISTINCT temperature FROM dataset_1' is entered. On the left, there's a sidebar with 'Grid' and 'Text' views. The 'Grid' view is selected, showing a table with two rows. The first row has a value of 55, and the second row has a value of 30. The column header is '123 temperature'.

	123 temperature
1	55
2	30

Python Code

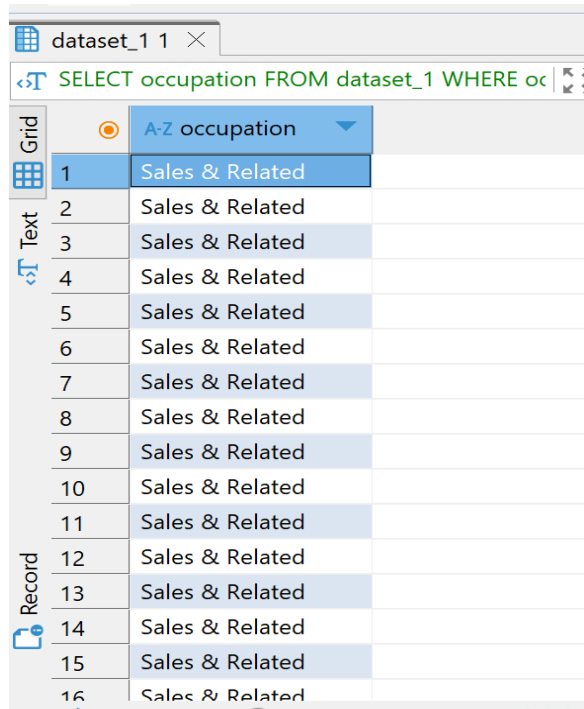
```
df[(df['temperature'] >= 29) & (df['temperature'] <= 75)]['temperature'].unique()
```

```
[44]: #SELECT distinct temperature from dataset_1 where temperature is between 29 to 75
      df[(df['temperature'] >= 29) & (df['temperature'] <= 75)]['temperature'].unique()
```

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

SQL Query

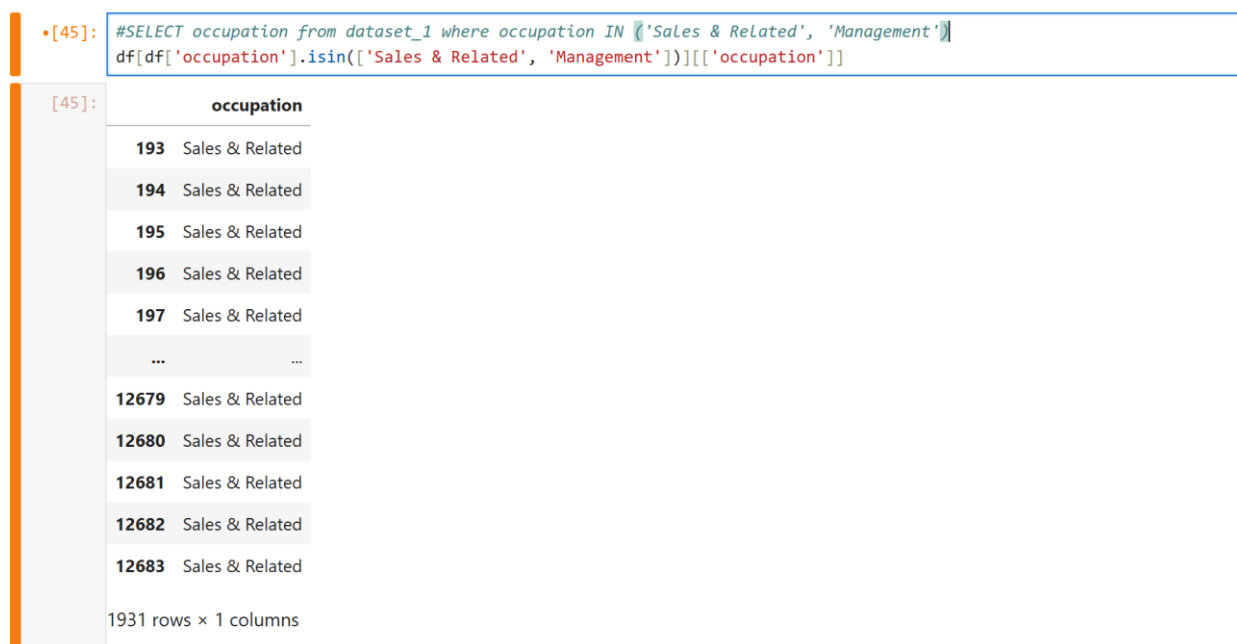
SELECT occupation FROM dataset_1 WHERE occupation IN('Sales & Related','Management')



	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 Code

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



```
[45]: #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
...	...
12679	Sales & Related
12680	Sales & Related
12681	Sales & Related
12682	Sales & Related
12683	Sales & Related

1931 rows × 1 columns