# INFO 2950: Intro to Data Science

## Whiteboards: draw your best avocado 🥑

Lecture 4
2023-08-29

# Agenda

1. Numpy and SQL stats
2. Grouping
3. Plotting in Python
4. More SQL Joins
5. Admin

# Numpy stats in 1-D

```
>>> a = np.array([[1, 2], [3, 4]])
```

Draw *a*!

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

```
>>> a = np.array([[1, 2], [3, 4]])
```

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

```
array([[1, 2],
       [3, 4]])
```

```
>>> a = np.array([[1, 2], [3, 4]])

>>> np.mean(a)

>>> np.var(a)

>>> np.std(a)

>>> np.median(a)
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

```
array([[1, 2],
       [3, 4]])
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)                          2.5
>>> np.var(a)                           1.25
>>> np.std(a)                           1.118033988749895
>>> np.median(a)                        2.5
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

```
array([[1, 2],
       [3, 4]])
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> a.shape
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

```
array([[1, 2],
       [3, 4]])
```

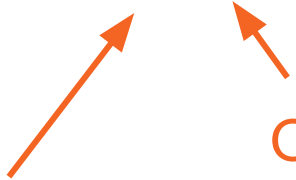
```
>>> a = np.array([[1, 2], [3, 4]])
>>> a.shape                          (2, 2)
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

```
array([[1, 2],
       [3, 4]])
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> a.shape
(2, 2)
```

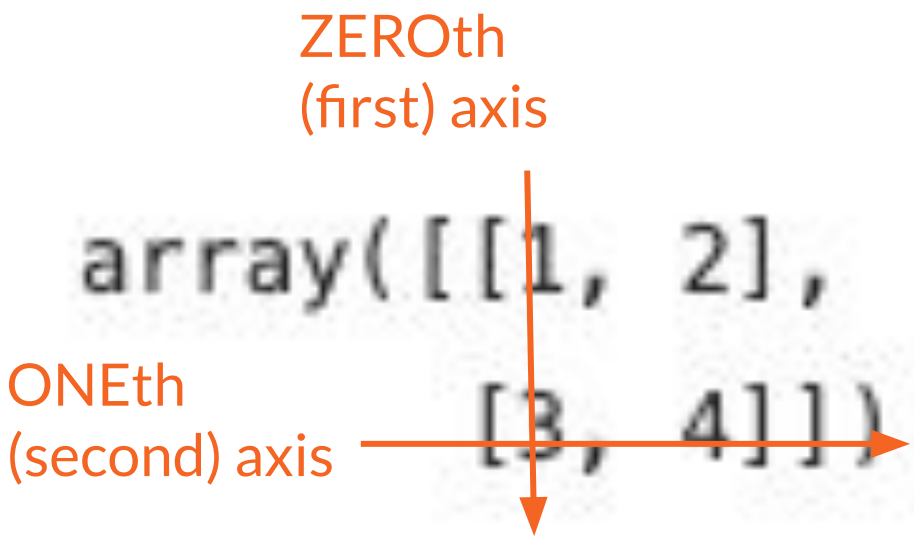


ZEROth
(first) axis

ONEth
(second) axis

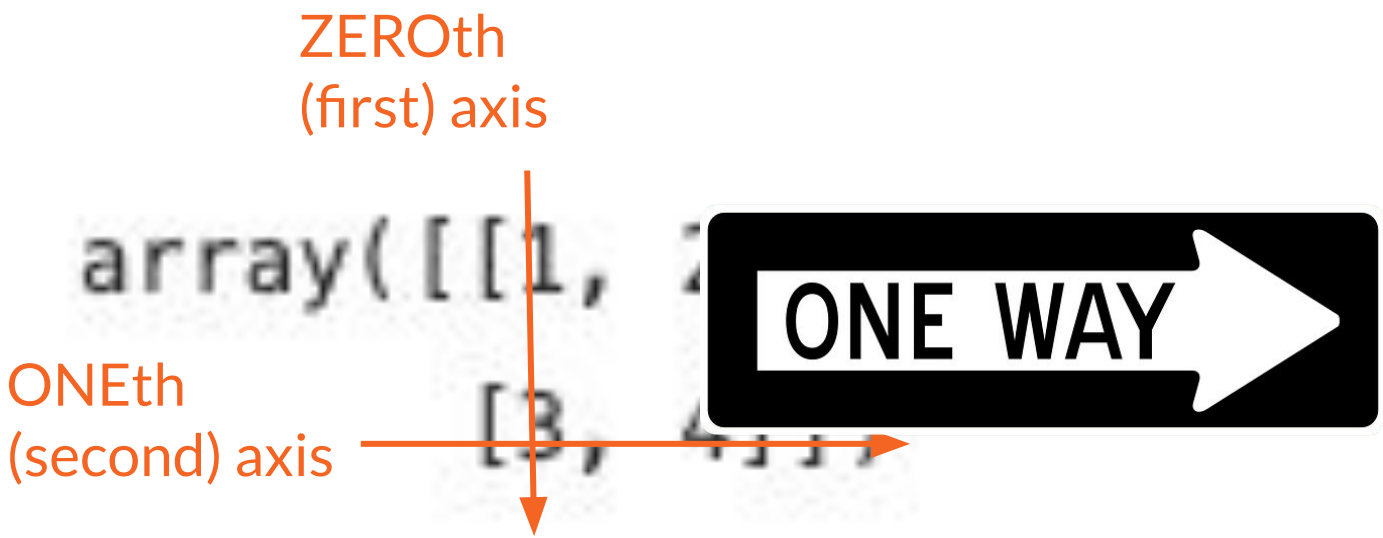
https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D

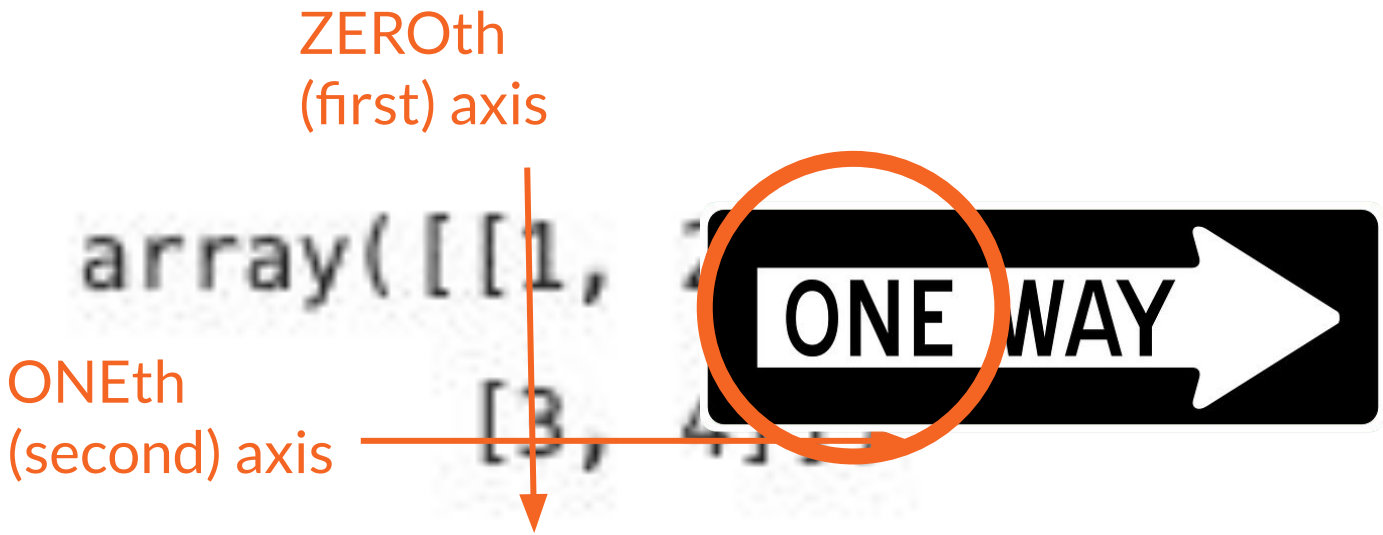



https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D



# Numpy stats in 1-D



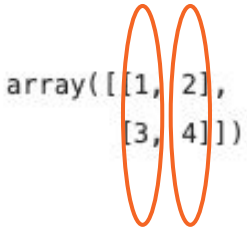
# Numpy stats in 1-D, axis 0

```
array([[1, 2],
       [3, 4]])
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a, axis=0)
>>> np.var(a, axis=0)
>>> np.std(a, axis=0)
>>> np.median(a, axis=0)
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D, axis 0




```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a, axis=0)                array([2., 3.])
>>> np.var(a, axis=0)                 array([1., 1.])
>>> np.std(a, axis=0)                 array([1., 1.])
>>> np.median(a, axis=0)              array([2., 3.])
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

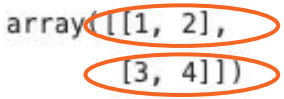
# Numpy stats in 1-D, axis 1

```
array([[1, 2],
       [3, 4]])
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a, axis=1)
>>> np.var(a, axis=1)
>>> np.std(a, axis=1)
>>> np.median(a, axis=1)
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D, axis 1

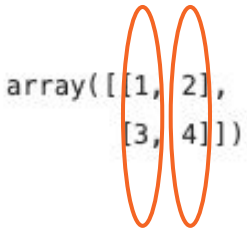



```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a, axis=1)          array([1.5, 3.5])
>>> np.var(a, axis=1)           array([0.25, 0.25])
>>> np.std(a, axis=1)           array([0.5, 0.5])
>>> np.median(a, axis=1)        array([1.5, 3.5])
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

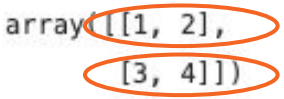

https://andrewm4894.com/2020/10/15/numpy-feature-engineering-2x-speed-up-over-pandas/

# Numpy stats in 1-D




```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a, axis=0)          array([2, 3])
```

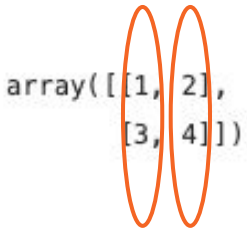
**`axis`** specifies the dimension you want to **get rid of**




```
>>> np.mean(a, axis=1)          array([1.5, 3.5])
```

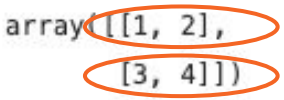
https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Numpy stats in 1-D




```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a, axis=0)          array([2, 3])
```

**`axis`** specifies the dimension you want to **get rid of**




```
>>> np.mean(a, axis=1)          array([1.5, 3.5])
```

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

# Practice: Numpy stats in 1-D

```
>>> a = np.array([[2, 1, 0], [4, 2, 6]])
>>> a.shape                          (___,____)
>>> np.mean(a, axis=0)               array([_____])
>>> np.mean(a, axis=1)               array([_____])
```

# Practice: Numpy stats in 1-D

```
>>> a = np.array([[2, 1, 0], [4, 2, 6]])
>>> a.shape                          (2,3)
>>> np.mean(a, axis=0)               array([3., 1.5, 3.])
>>> np.mean(a, axis=1)               array([1., 4.])
```

| | | |
|---|---|---|
| 2 | 1 | 0 |
| 4 | 2 | 6 |

# SQL stats in 1-D

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT VARIANCE(column_name)
FROM table_name
WHERE condition;
```

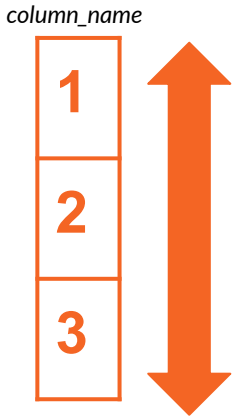
# Practice: On which axis does this SQL code take an average?

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
np.mean(column, axis=?)
```

# Practice: On which axis does this SQL code take an average?

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```




```
np.mean(column, axis=0)
```

# Practice: On which axis does this SQL code take an average?

```
SELECT AVG(column_name)

FROM table_name

WHERE condition;
```


`np.mean(column)` ← We actually don't need to specify an axis here since *column* is one-dimensional!

```
food_df = pd.DataFrame({'Restaurant': ['Four Seasons', 'Plum Tree', 'CTB'], 'Rating': [1, 2, 3]})
food_df
```

✓ 0.6s

| | Restaurant | Rating |
|---|---|---|
| 0 | Four Seasons | 1 |
| 1 | Plum Tree | 2 |
| 2 | CTB | 3 |

# How do we get the average rating across all restaurants?

```
food_df = pd.DataFrame({'Restaurant': ['Four Seasons', 'Plum Tree', 'CTB'], 'Rating': [1, 2, 3]})
food_df
```

✓ 0.6s

| | Restaurant | Rating |
|---|---|---|
| 0 | Four Seasons | 1 |
| 1 | Plum Tree | 2 |
| 2 | CTB | 3 |

**How do we get the average rating across all restaurants? (answer = 2.0)**

**SQL:** SELECT _____ FROM food_df

**Numpy:** np.mean(_____)

```
food_df = pd.DataFrame({'Restaurant': ['Four Seasons', 'Plum Tree', 'CTB'], 'Rating': [1, 2, 3]})
food_df
```

✓ 0.6s

| | Restaurant | Rating |
|---|---|---|
| 0 | Four Seasons | 1 |
| 1 | Plum Tree | 2 |
| 2 | CTB | 3 |

**How do we get the average rating across all restaurants? (answer = 2.0)**

**SQL:** SELECT AVG(Rating) FROM food_df

**Numpy:** np.mean(food_df['Rating'])

```
food_df = pd.DataFrame({'Restaurant': ['Four Seasons', 'Plum Tree', 'CTB'], 'Rating': [1, 2, 3]})
food_df
```

✓ 0.6s

| | Restaurant | Rating |
|---|---|---|
| 0 | Four Seasons | 1 |
| 1 | Plum Tree | 2 |
| 2 | CTB | 3 |

```
duckdb.sql("SELECT AVG(Rating) FROM food_df;").df()
```

✓ 0.3s

| | avg(Rating) |
|---|---|
| 0 | 2.0 |

```
np.mean(food_df['Rating'])
```

✓ 0.2s

```
2.0
```

```
np.mean(food_df['Rating'], axis=0)
```

✓ 0.2s

```
2.0
```

# But what if we want to know average ratings… based on *type* of restaurant?

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

# What if we want to know the average food rating in Collegetown grouped by food?

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

| Food | Avg_Food_Rating |
|---|---|
| Noodles | 2.333 |
| Sandwich | 2.5 |
| Tacos | 1.5 |

# Common concept, called a "GROUP BY"

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

| Food | Avg_Food_Rating |
|---|---|
| Noodles | 2.333 |
| Sandwich | 2.5 |
| Tacos | 1.5 |

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

# Group By's in Pandas

.groupby() of a column gives you a GroupBy object, but it doesn't seem to really do anything yet…

```
food_df.groupby('Food')
```

✓ 0.3s

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff0f1215360>
```

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

# Group By's in Pandas

**But you can do things *on* a pandas GroupBy object!**

```
food_df.groupby('Food')['Rating'].mean()
```

✓ 0.3s

```
Food
Noodles     2.333333
Sandwich    2.500000
Tacos       1.500000
Name: Rating, dtype: float64
```

# Group By's in SQL

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

```
duckdb.sql("SELECT Food, AVG(Rating) FROM food_df GROUP BY Food;").df()
```

✓ 0.5s

| | Food | avg(Rating) |
|---|---|---|
| 0 | Noodles | 2.333333 |
| 1 | Sandwich | 2.500000 |
| 2 | Tacos | 1.500000 |

# Think, Pair, Share: When would it not make sense to use a Group By?

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Sandwich | 2 |
| Gorgers | Sandwich | 3 |
| Dos Amigos | Tacos | 2 |
| Luna Inspired Street Food | Tacos | 1 |

(A good group by)

| Food | Avg_Food_Rating |
|---|---|
| Noodles | 2.333 |
| Sandwich | 2.5 |
| Tacos | 1.5 |

# When would it not make sense to use a Group By?

| Restaurant | Food | Rating |
|---|---|---|
| Shi Miao Dao | Noodles | 2 |
| Pho Time | Noodles | 2 |
| De Tasty | Noodles | 3 |
| CTB | Noodles | 2 |
| Gorgers | Noodles | 3 |
| Dos Amigos | Noodles | 2 |
| Luna Inspired Street Food | Noodles | 1 |

- If all the food in Ithaca were entirely noodles, grouping by Food wouldn't be informative
- We want multiple values in the group by! Otherwise it's the same as just doing Avg(Rating)

# When would it not make sense to use a Group By?

| Restaurant | Food |
|---|---|
| Shi Miao Dao | Noodles |
| Pho Time | Noodles |
| De Tasty | Noodles |
| CTB | Sandwich |
| Gorgers | Sandwich |
| Dos Amigos | Tacos |
| Luna Inspired Street Food | Tacos |

- If the food were diverse but we didn't have any corresponding data (no ratings), then grouping doesn't do much!
- Using a Group By, we could still report the count (# restaurants per Food type), but that's it

# 1 min break & attendance!

tinyurl.com/4a8kfjz5

# The pandas library helps us read CSV files

```
avocados_df = pd.read_csv("avocado.csv")

avocados_df.head()
```

| | Unnamed: 0 | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | Total Bags | Small Bags | Large Bags |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 2015-12-27 | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | 8696.87 | 8603.62 | 93.25 |
| **1** | 1 | 2015-12-20 | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | 9505.56 | 9408.07 | 97.49 |
| **2** | 2 | 2015-12-13 | 0.93 | 118220.22 | 794.70 | 109149.67 | 130.50 | 8145.35 | 8042.21 | 103.14 |
| **3** | 3 | 2015-12-06 | 1.08 | 78992.15 | 1132.00 | 71976.41 | 72.58 | 5811.16 | 5677.40 | 133.76 |
| **4** | 4 | 2015-11-29 | 1.28 | 51039.60 | 941.48 | 43838.39 | 75.78 | 6183.95 | 5986.26 | 197.69 |

https://www.kaggle.com/datasets/neuromusic/avocado-prices

# What's going on here?

```
avocados_df = pd.read_csv("avocado.csv")

avocados_df.head()
```

| | Unnamed: 0 | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | Total Bags | Small Bags | Large Bags |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 2015-12-27 | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | 8696.87 | 8603.62 | 93.25 |
| **1** | 1 | 2015-12-20 | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | 9505.56 | 9408.07 | 97.49 |
| **2** | 2 | 2015-12-13 | 0.93 | 118220.22 | 794.70 | 109149.67 | 130.50 | 8145.35 | 8042.21 | 103.14 |
| **3** | 3 | 2015-12-06 | 1.08 | 78992.15 | 1132.00 | 71976.41 | 72.58 | 5811.16 | 5677.40 | 133.76 |
| **4** | 4 | 2015-11-29 | 1.28 | 51039.60 | 941.48 | 43838.39 | 75.78 | 6183.95 | 5986.26 | 197.69 |

https://www.kaggle.com/datasets/neuromusic/avocado-prices

# The pandas library helps us read CSV files

```
avocados_df = pd.read_csv("avocado.csv")

avocados_df.head()
```

| | Unnamed: 0 | Date | Small Bags | Large Bags |
|---|---|---|---|---|
| 0 | 0 | 2015-12-27 | 8603.62 | 93.25 |
| 1 | 1 | 2015-12-20 | 9408.07 | 97.49 |
| 2 | 2 | 2015-12-13 | 8042.21 | 103.14 |
| 3 | 3 | 2015-12-06 | 5677.40 | 133.76 |
| 4 | 4 | 2015-11-29 | 5986.26 | 197.69 |

If we don't specify an index column, pandas creates one with line numbers by default

This dataset was previously loaded in pandas, and then saved to a file *with the new line number column*

When we loaded it just now, pandas didn't know that the first column is the index, so it made a new one

# Did we read the values correctly?

```
avocados_df.dtypes
```

```
Unnamed: 0        int64
Date             object
AveragePrice    float64
Total Volume    float64
4046            float64
4225            float64
4770            float64
Total Bags      float64
Small Bags      float64
Large Bags      float64
XLarge Bags     float64
type             object
year              int64
region           object
dtype: object
```

# Did we read the values correctly?

```
avocados_df.dtypes
```

```
Unnamed: 0        int64
Date             object
AveragePrice    float64
Total Volume    float64
4046            float64
4225            float64
4770            float64
Total Bags      float64
Small Bags      float64
Large Bags      float64
XLarge Bags     float64
type             object
year              int64
region           object
dtype: object
```

pandas uses *object* for columns it doesn't know what to do with. We won't need the date field, so looks good to me!

# duckdb runs SQL on pandas data frames

```
duckdb.sql("SELECT Date, AveragePrice FROM avocados_df WHERE
region == 'Syracuse' LIMIT 5")
```

| Date<br>varchar | AveragePrice<br>double |
|---|---|
| 2015-12-27 | 1.36 |
| 2015-12-20 | 1.36 |
| 2015-12-13 | 1.32 |
| 2015-12-06 | 1.17 |
| 2015-11-29 | 1.41 |

# duckdb runs SQL on pandas data frames

```
duckdb.sql("SELECT Date, AveragePrice FROM avocados_df WHERE
region == 'Syracuse' LIMIT 5")
```

| Date<br>varchar | AveragePrice<br>double |
|---|---|
| 2015-12-27 | 1.36 |
| 2015-12-20 | 1.36 |
| 2015-12-13 | 1.32 |
| 2015-12-06 | 1.17 |
| 2015-11-29 | 1.41 |

**how did it know** that avocados_df is a thing?

duckdb interprets SQL table names as variable names, and checks if that value is a pandas dataframe

# duckdb runs SQL on pandas data frames

```
duckdb.sql("SELECT Date, AveragePrice FROM avocados_df WHERE
region == 'Syracuse' LIMIT 5")
```

| Date<br>varchar | AveragePrice<br>double |
|---|---|
| 2015-12-27 | 1.36 |
| 2015-12-20 | 1.36 |
| 2015-12-13 | 1.32 |
| 2015-12-06 | 1.17 |
| 2015-11-29 | 1.41 |

Each output column lists the name of the variable ("Date", "AveragePrice") and the data type of the variable

varchar is a string ("variable number of characters")

# duckdb runs SQL on pandas data frames

```
duckdb.sql("SELECT COUNT(*) FROM avocados_df")
```

| count_star()<br>int64 |
| --- |
| 18249 |

The count function returns the total number of rows selected

The output is a new integer variable called "count_star()"

# GROUP BY divides data by facets

```
duckdb.sql("""SELECT year, COUNT(*) AS total_rows_per_year
FROM avocados_df GROUP BY year""")
```

here I'm using AS to give the output of count(*) a name

I'm using """ quotes for a multi-line string

# GROUP BY divides data by facets

```
duckdb.sql("""SELECT year, COUNT(*) AS total_rows_per_year
FROM avocados_df GROUP BY year""")
```

```
duckdb.sql("SELECT year, \
                  COUNT(*) AS total_rows_per_year \
            FROM avocados_df \
            GROUP BY year")
```

here I'm using AS to give the output of count(*) a name

I'm using """ quotes for a multi-line string

You can also use \ to break up a multi-line string

# GROUP BY divides data by facets

```
duckdb.sql("""SELECT year, COUNT(*) AS total_rows_per_year
FROM avocados_df GROUP BY year""")
```

| year<br>int64 | total_rows_per_year<br>int64 |
|---|---|
| 2015 | 5615 |
| 2016 | 5616 |
| 2017 | 5722 |
| 2018 | 1296 |

Instead of one count we are grouping by year and counting each group.

This only makes sense if we return at least one more column, like year.

# GROUP BY divides data by facets

```
duckdb.sql("""SELECT year, COUNT(*) AS total_rows_per_year
FROM avocados_df GROUP BY year""").df()
```

| | year | total_rows_per_year |
|---|---|---|
| 0 | 2015 | 5615 |
| 1 | 2016 | 5616 |
| 2 | 2017 | 5722 |
| 3 | 2018 | 1296 |

The previous examples returned duckdb objects, here I'm adding **.df()** to convert the output to a pandas dataframe

# Fill in the blank!

```
duckdb.sql("""SELECT year, AVG(average_price)
FROM avocados_df GROUP BY year""").df()
```

annual_avg_price

| | year | ~~avg("AveragePrice")~~ |
|---|---|---|
| 0 | 2015 | 1.375590 |
| 1 | 2016 | 1.338640 |
| 2 | 2017 | 1.515128 |
| 3 | 2018 | 1.347531 |

# Fill in the blank!

```
duckdb.sql("""SELECT year, AVG(average_price) AS annual_avg_price
FROM avocados_df GROUP BY year""").df()
```

| | year | ~~avg("AveragePrice")~~ annual_avg_price |
|---|---|---|
| 0 | 2015 | 1.375590 |
| 1 | 2016 | 1.338640 |
| 2 | 2017 | 1.515128 |
| 3 | 2018 | 1.347531 |

# You can save queries as a variable!

```
query = """SELECT region, AVG(AveragePrice) AS region_price,
STDDEV(AveragePrice) AS region_std
FROM avocados_df GROUP BY region"""

duckdb.sql(query).df()
```

# And, you can save dataframe output!

```
query = """SELECT region, AVG(AveragePrice) AS region_price,
STDDEV(AveragePrice) AS region_std
FROM avocados_df GROUP BY region"""

region_mean_sd = duckdb.sql(query).df()
```

Rather than display the returned data frame, save it as a variable

SQL turns one df into a different df!

# Think, Pair, Share: What will this do?

```
SELECT region, AVG(AveragePrice) AS region_price FROM
avocados_df GROUP BY region ORDER BY region_price DESC LIMIT 10
```

# What will this do?

```
SELECT region, AVG(AveragePrice) AS region_price FROM
avocados_df GROUP BY region ORDER BY region_price DESC LIMIT 10
```

For each region

Get the average price

Sort the regions from highest to lowest average price

Show only the first 10

```
SELECT region, AVG(AveragePrice) AS region_price FROM
avocados_df GROUP BY region ORDER BY region_price DESC LIMIT 10
```

# Where do you guess has the most expensive avocados? (on avg, from ~2015-2018)

For each region

Get the average price

Sort the regions from highest to lowest average price

Show only the first 10

# What will this do?

```
SELECT region, AVG(AveragePrice) AS region_price FROM
avocados_df GROUP BY region ORDER BY region_price DESC LIMIT 10
```

| region<br>varchar | region_price<br>double |
|---|---|
| HartfordSpringfield | 1.8186390532544363 |
| SanFrancisco | 1.8042011834319525 |
| NewYork | 1.727573964497041 |
| Philadelphia | 1.6321301775147927 |
| Sacramento | 1.6215680473372784 |
| Charlotte | 1.6060355029585796 |
| Northeast | 1.6019230769230774 |
| Albany | 1.5610355029585792 |
| Chicago | 1.5567751479289942 |
| RaleighGreensboro | 1.555118343195266 |
| 10 rows | 2 columns |

For each region

Get the average price

Sort the regions from highest to lowest average price

Show only the first 10

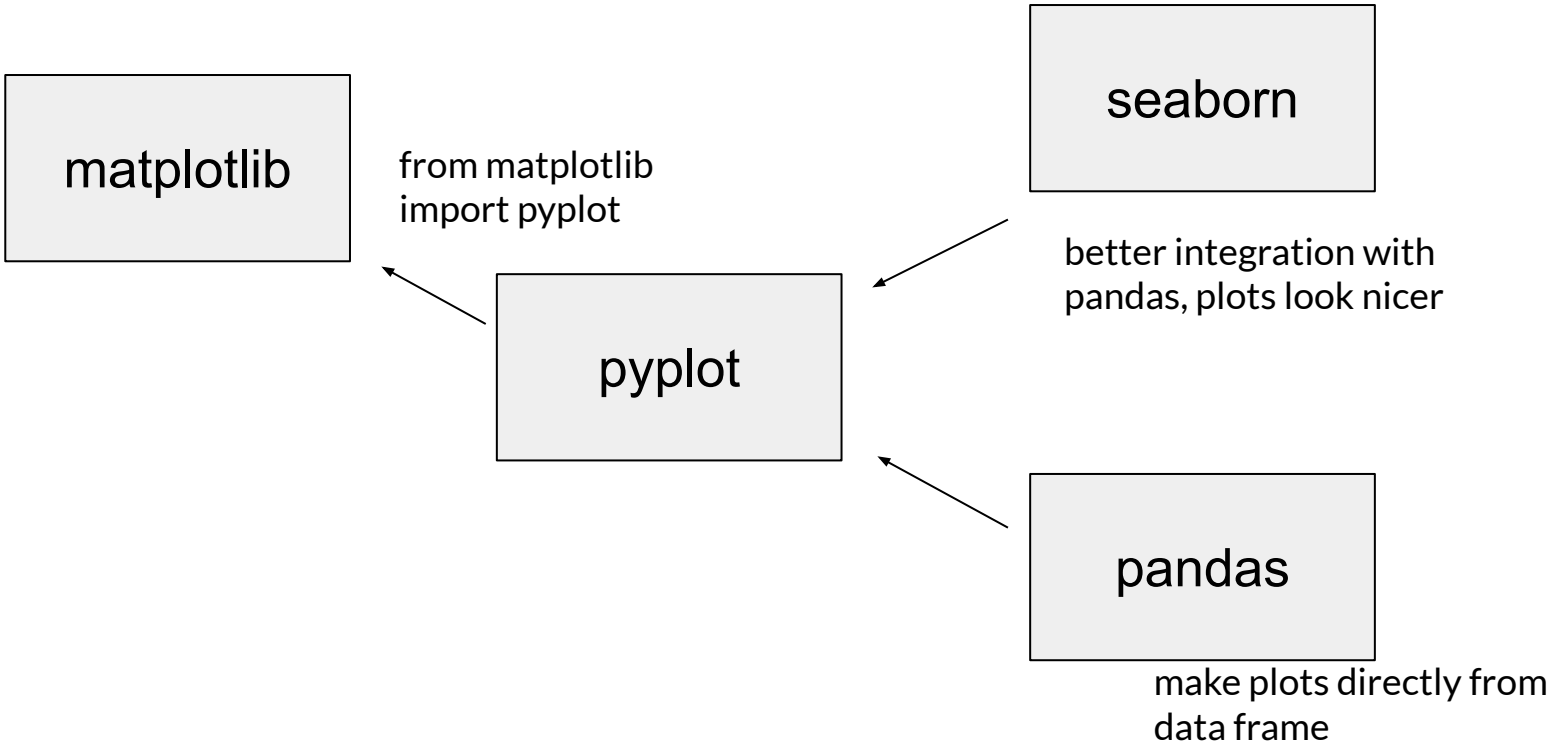
# What will this do?

```
SELECT region, AVG(AveragePrice) AS region_price FROM
avocados_df GROUP BY region ORDER BY region_price DESC LIMIT 10
```

| region<br>varchar | region_price<br>double |
|---|---|
| HartfordSpringfield | 1.8186390532544363 |
| SanFrancisco | 1.8042011834319525 |
| NewYork | 1.727573964497041 |
| Philadelphia | 1.6321301775147927 |
| Sacramento | 1.6215680473372784 |
| Charlotte | 1.6060355029585796 |
| Northeast | 1.6019230769230774 |
| Albany | 1.5610355029585792 |
| Chicago | 1.5567751479289942 |
| RaleighGreensboro | 1.555118343195266 |
| 10 rows | 2 columns |

**Tip: you always want to SELECT on the column(s) you GROUP BY. Otherwise you won't know what the other stats correspond to!**

# Making plots in python

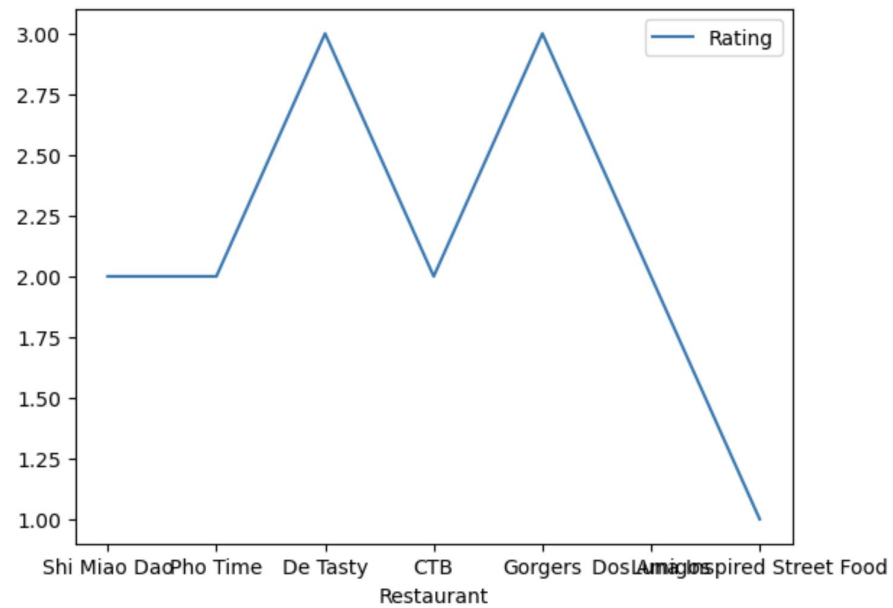


# pandas.DataFrame.plot(*x*,*y*)

```
food_df.plot("Restaurant","Rating")
```

✓ 0.6s

```
<Axes: xlabel='Restaurant'>
```



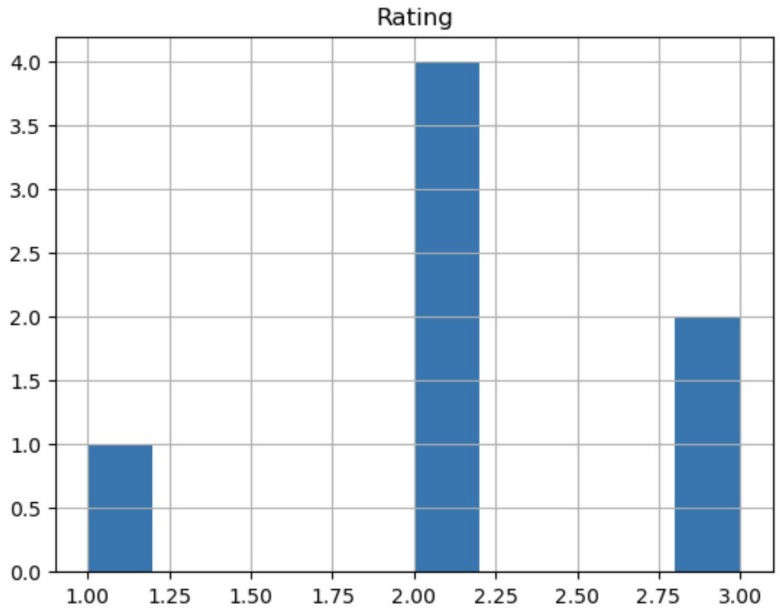

pandas

make plots directly from data frame

# pandas.DataFrame.hist(*x*)

```
food_df.hist("Rating")
```
✓ 0.5s

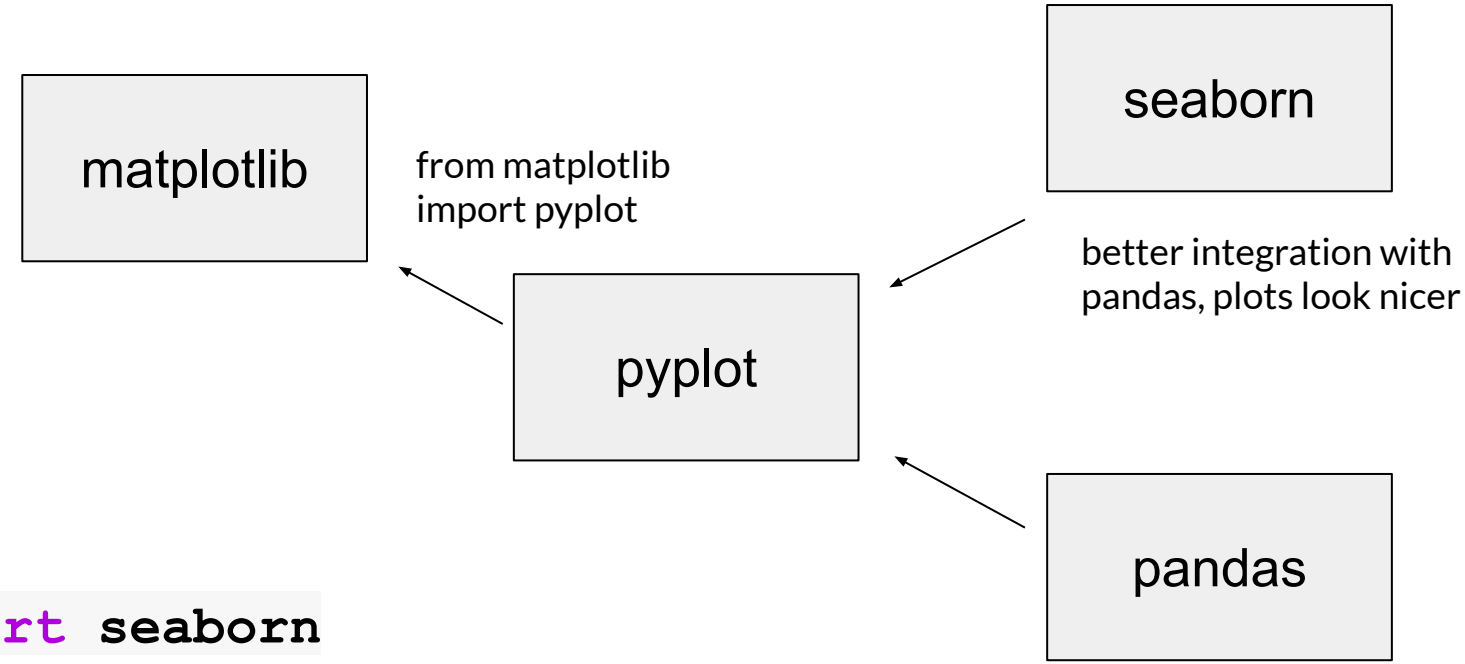
```
array([[<Axes: title={'center': 'Rating'}>]], dtype=object)
```




pandas

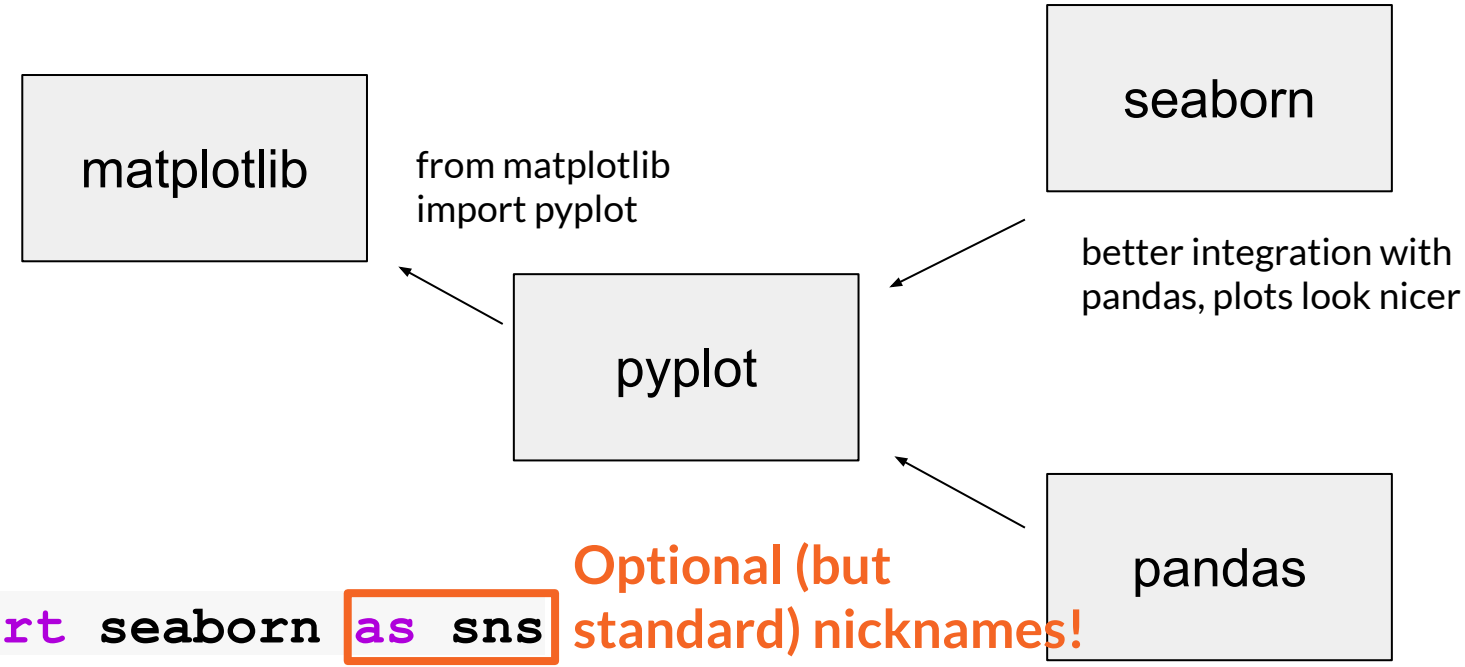
make plots directly from data frame

# Making plots in python




```
import seaborn
from matplotlib import pyplot
```

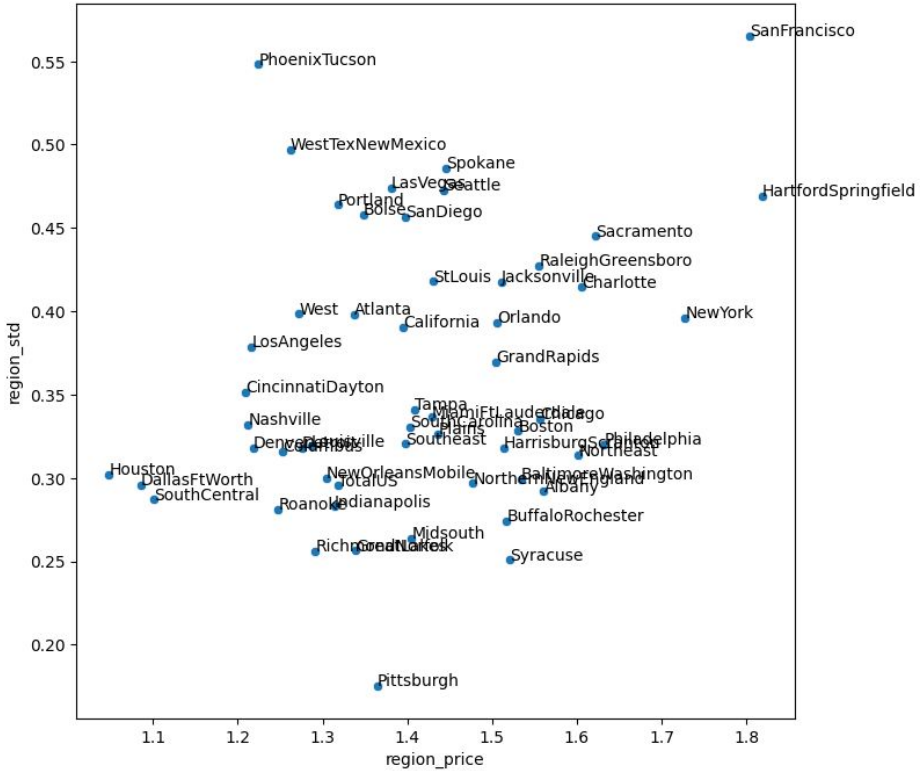
# Making plots in python



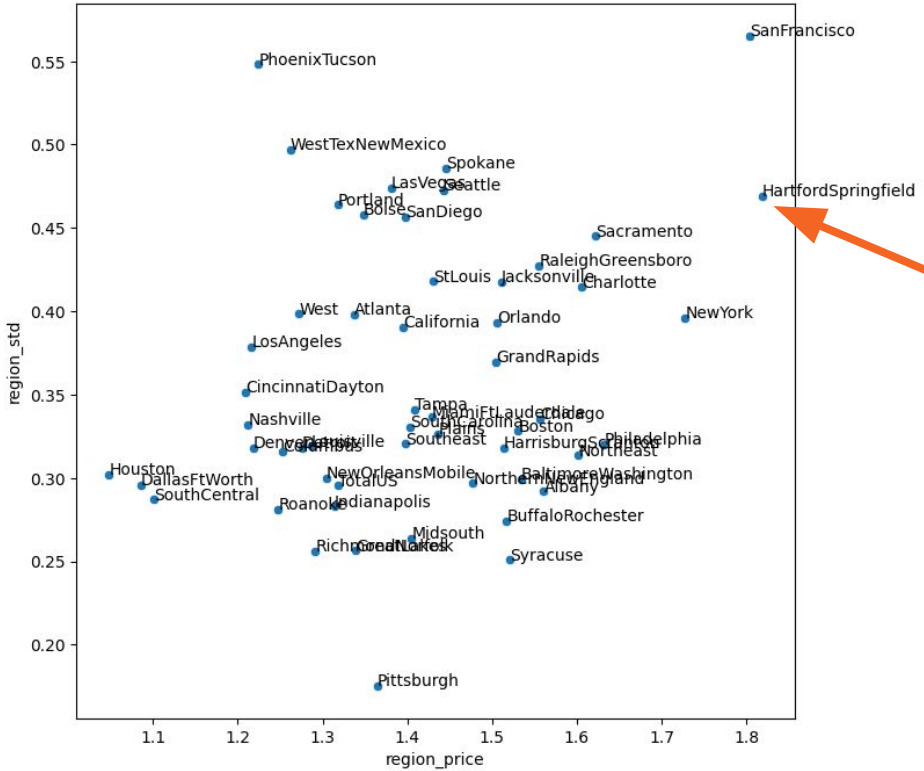

```
import seaborn as sns
from matplotlib import pyplot as plt
```

Optional (but standard) nicknames!

```
seaborn.scatterplot(x="region_price", y="region_std", data=region_mean_sd)
for i, row in region_mean_sd.iterrows():
    pyplot.text(x=row["region_price"], y=row["region_std"], s=row["region"] )
```



```
seaborn.scatterplot(x="region_price", y="region_std", data=region_mean_sd)
for i, row in region_mean_sd.iterrows():
    pyplot.text(x=row["region_price"], y=row["region_std"], s=row["region"] )
```




$$$

# 1 min break!

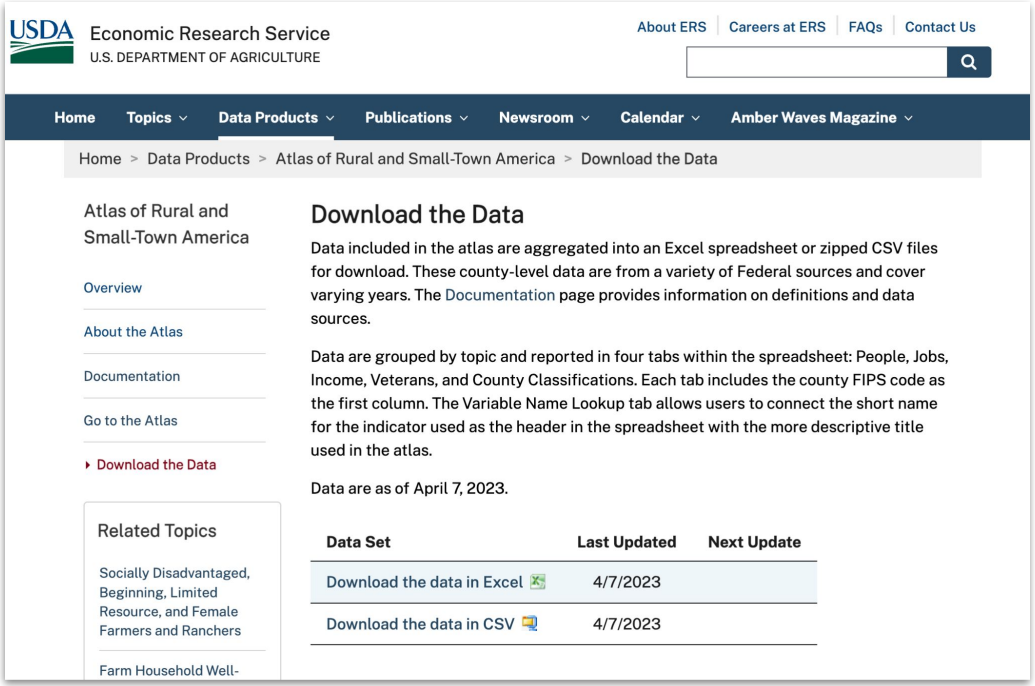

https://www.reddit.com/r/ProgrammerHumor/comments/obr4py/they_just_dont_understand/

# Why bother with SQL joins?

- Your data will come from multiple sources and you will need to figure out how to combine them

- Classic **interview** question format: “here are two data frames; tell me the SQL code to get *[description of resulting joined df]*”
    - You will need to choose the correct join statement for this!

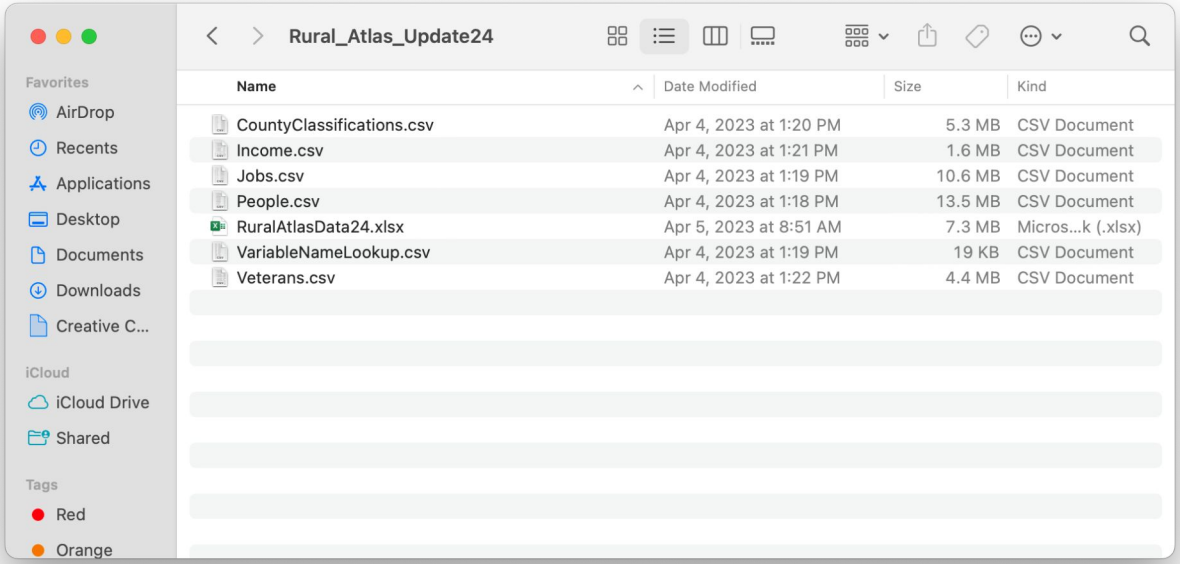
# Example: JOIN between two files




The USDA Atlas of Rural and Small-Town America actually contains information about the entire country.

Which has recently grown faster, urban counties or rural counties?

# Data spread across two files




The data comes in six CSV files

Urban/Rural classification is in **CountyClassifications.csv**

Population growth is in **People.csv**

# What would you Google to resolve this?

```
county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
```

---

UnicodeDecodeError Traceback (most recent call last)
Cell In[43], line 1
----> 1 county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
2 county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
...

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 77183: invalid continuation byte

[39]:

# What would you Google to resolve this?

```
county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
Cell In[43], line 1
----> 1 county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
      2 county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in
position 77183: invalid continuation byte
```

[39]:

**Try googling "UnicodeDecodeError utf-8" first; realize there are lots of different byte-related answers. So, append "invalid continuation byte" to search…**

# What would you Google to resolve this?

```
county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
```

---
UnicodeDecodeError
Cell In[43], line 1
----> 1 county_classifica
pd.read_csv("Rural_Atl
2 county_populatior
pd.read_csv("Rural_Atl
...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 77183: invalid continuation byte

532

I had the same error when I tried to open a CSV file by `pandas.read_csv` method.

The solution was change the encoding to `latin-1`:

```
pd.read_csv('ml-100k/u.item', sep='|', names=m_cols , encoding='latin-1')
```

[39]: https://stackoverflow.com/questions/5552555/unicodedecodeerror-invalid-continuation-byte

# Problem: character encodings

```
county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
Cell In[43], line 1
----> 1 county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv")
      2 county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv")
...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in
position 77183: invalid continuation byte

[39]:
```

Doña Ana county in New Mexico is not being read correctly!

The file is in an older, pre-Unicode format

# Problem: delimiters

```
county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv",
                                        encoding="LATIN-1")
county_classifications.head()
```

| | FIPStxt\tState\tCounty\tRuralUrbanContinuumCode2013\tUrbanInfluenceCode2013\tRu |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |

CSV means "comma separated values", but it looks like this file actually uses tabs **\t**

# Problem: data types

```
county_classifications =
pd.read_csv("Rural_Atlas_Update24/CountyClassifications.csv",
                          delimiter="\t", encoding="LATIN-1")
county_classifications.head()
```

| | FIPStxt | State | County | RuralUrbanContinuumCode2013 | UrbanInfluenceCode2013 |
|---|---|---|---|---|---|
| 0 | 1001 | AL | Autauga | 2.0 | 2.0 |
| 1 | 1003 | AL | Baldwin | 3.0 | 2.0 |
| 2 | 1005 | AL | Barbour | 6.0 | 6.0 |
| 3 | 1007 | AL | Bibb | 1.0 | 1.0 |
| 4 | 1009 | AL | Blount | 1.0 | 1.0 |

FIPS is a code like a zip code for US places

Autauga, AL should be **01001**

# Problem: data types

```
county_population =
pd.read_csv("Rural_Atlas_Update24/People.csv",
                          delimiter="\t", encoding="LATIN-1")
county_population.head()
```

| | FIPS | State | County | PopChangeRate1819 | PopChangeRate1019 | TotalPopEst2019 |
|---|---|---|---|---|---|---|
| 0 | 0 | US | United States | 0.475 | 6.116 | 328239523 |
| 1 | 1000 | AL | Alabama | 0.317 | 2.461 | 4903185 |
| 2 | 1001 | AL | Autauga | 0.605 | 2.001 | 55869 |
| 3 | 1003 | AL | Baldwin | 2.469 | 21.911 | 223234 |
| 4 | 1005 | AL | Barbour | -0.748 | -9.664 | 24686 |

This data frame has a different name (FIPS vs FIPStxt), but it seems to be the same data type, without the leading 0s

As long as they're both wrong in the same way...

# Can we match rows in two tables?

| | FIPStxt | State | County | RuralUrbanContinuumCode2013 | UrbanInfluenceCode2013 |
|---|---|---|---|---|---|
| 0 | 1001 | AL | Autauga | 2.0 | 2.0 |
| 1 | 1003 | AL | Baldwin | 3.0 | 2.0 |
| 2 | 1005 | AL | Barbour | 6.0 | 6.0 |
| 3 | 1007 | AL | Bibb | 1.0 | 1.0 |
| 4 | 1009 | AL | Blount | 1.0 | 1.0 |

| | FIPS | State | County | PopChangeRate1819 | PopChangeRate1019 | TotalPopEst2019 |
|---|---|---|---|---|---|---|
| 0 | 0 | US | United States | 0.475 | 6.116 | 328239523 |
| 1 | 1000 | AL | Alabama | 0.317 | 2.461 | 4903185 |
| 2 | 1001 | AL | Autauga | 0.605 | 2.001 | 55869 |
| 3 | 1003 | AL | Baldwin | 2.469 | 21.911 | 223234 |
| 4 | 1005 | AL | Barbour | -0.748 | -9.664 | 24686 |

**What fields in the two data frames would we use to find rows that match?**

# Can we match rows in two tables?

| | FIPStxt | State | County | RuralUrbanContinuumCode2013 | UrbanInfluenceCode2013 |
|---|---|---|---|---|---|
| 0 | 1001 | AL | Autauga | 2.0 | 2.0 |
| 1 | 1003 | AL | Baldwin | 3.0 | 2.0 |
| 2 | 1005 | AL | Barbour | 6.0 | 6.0 |
| 3 | 1007 | AL | Bibb | 1.0 | 1.0 |
| 4 | 1009 | AL | Blount | 1.0 | 1.0 |

| | FIPS | State | County | PopChangeRate1819 | PopChangeRate1019 | TotalPopEst2019 |
|---|---|---|---|---|---|---|
| 0 | 0 | US | United States | 0.475 | 6.116 | 328239523 |
| 1 | 1000 | AL | Alabama | 0.317 | 2.461 | 4903185 |
| 2 | 1001 | AL | Autauga | 0.605 | 2.001 | 55869 |
| 3 | 1003 | AL | Baldwin | 2.469 | 21.911 | 223234 |
| 4 | 1005 | AL | Barbour | -0.748 | -9.664 | 24686 |

Definitely FIPS (since it's the most granular); could additionally match on State and County

Ok to just match on State or only on County? NO, since they're not necessarily enough to identify the FIPS

# Can we match rows in two tables?

| | FIPStxt | State | County | RuralUrbanContinuumCode2013 | UrbanInfluenceCode2013 |
|---|---|---|---|---|---|
| 0 | 1001 | AL | Autauga | 2.0 | 2.0 |
| 1 | 1003 | AL | Baldwin | 3.0 | 2.0 |
| 2 | 1005 | AL | Barbour | 6.0 | 6.0 |
| 3 | 1007 | AL | Bibb | 1.0 | 1.0 |
| 4 | 1009 | AL | Blount | 1.0 | 1.0 |

| | FIPS | State | County | PopChangeRate1819 | PopChangeRate1019 | TotalPopEst2019 |
|---|---|---|---|---|---|---|
| 0 | 0 | US | United States | 0.475 | 6.116 | 328239523 |
| 1 | 1000 | AL | Alabama | 0.317 | 2.461 | 4903185 |
| 2 | 1001 | AL | Autauga | 0.605 | 2.001 | 55869 |
| 3 | 1003 | AL | Baldwin | 2.469 | 21.911 | 223234 |
| 4 | 1005 | AL | Barbour | -0.748 | -9.664 | 24686 |

These two columns have different names, is that okay for matching?

# SQL JOIN

```
county_df = duckdb.sql("""SELECT c.FIPStxt, c.State, c.County,
c.RuralUrbanContinuumCode2013, p.PopChangeRate1019
FROM county_classifications, county_population
WHERE county_classifications.FIPStxt = county_population.FIPS""").df()
```

| | FIPStxt | State | County | RuralUrbanContinuumCode2013 | PopChangeRate1019 |
|---|---|---|---|---|---|
| **0** | 1003 | AL | Baldwin | 3.0 | 21.911 |
| **1** | 1005 | AL | Barbour | 6.0 | -9.664 |
| **2** | 1007 | AL | Bibb | 1.0 | -2.081 |
| **3** | 1009 | AL | Blount | 1.0 | 0.784 |
| **4** | 1011 | AL | Bullock | 6.0 | -7.126 |

**Yes, we can match FIPStxt to FIPS as long as we identify which dataset it came from!**

# SQL JOIN

```
county_df = duckdb.sql("""SELECT c.FIPStxt, c.State, c.County,
c.RuralUrbanContinuumCode2013, p.PopChangeRate1019
FROM county_classifications c, county_population p
WHERE c.FIPStxt = p.FIPS""").df()
```

| | FIPStxt | State | County | RuralUrbanContinuumCode2013 | PopChangeRate1019 |
|---|---|---|---|---|---|
| 0 | 1003 | AL | Baldwin | 3.0 | 21.911 |
| 1 | 1005 | AL | Barbour | 6.0 | -9.664 |
| 2 | 1007 | AL | Bibb | 1.0 | -2.081 |
| 3 | 1009 | AL | Blount | 1.0 | 0.784 |
| 4 | 1011 | AL | Bullock | 6.0 | -7.126 |

More readable:

In the FROM clause we give each data frame a short name (c and p, respectively), and then in the WHERE clause we match the values

| | State | County | PopChangeRate1019 |
|---|---|---|---|
| **0** | ND | McKenzie | 134.311 |
| **1** | TX | Loving | 101.190 |
| **2** | ND | Williams | 66.404 |
| **3** | TX | Hays | 45.493 |
| **4** | UT | Wasatch | 44.185 |
| **...** | ... | ... | ... |
| **3215** | PR | Guánica | -20.587 |
| **3216** | PR | Lares | -20.781 |
| **3217** | TX | Terrell | -23.244 |
| **3218** | IL | Alexander | -29.795 |
| **3219** | TX | Concho | -33.545 |

3220 rows × 3 columns

```
county_df = duckdb.sql("""
SELECT State, County, PopChangeRate1019
FROM county_df
ORDER BY PopChangeRate1019 DESC""").df()
```

We can now run SQL queries on the joined data frame

| | State | County | PopChangeRate1019 |
|---|---|---|---|
| **0** | ND | McKenzie | 134.311 |
| **1** | TX | Loving | 101.190 |
| **2** | ND | Williams | 66.404 |
| **3** | TX | Hays | 45.493 |
| **4** | UT | Wasatch | 44.185 |
| **...** | ... | ... | ... |
| **3215** | PR | Guánica | -20.587 |
| **3216** | PR | Lares | -20.781 |
| **3217** | TX | Terrell | -23.244 |
| **3218** | IL | Alexander | -29.795 |
| **3219** | TX | Concho | -33.545 |

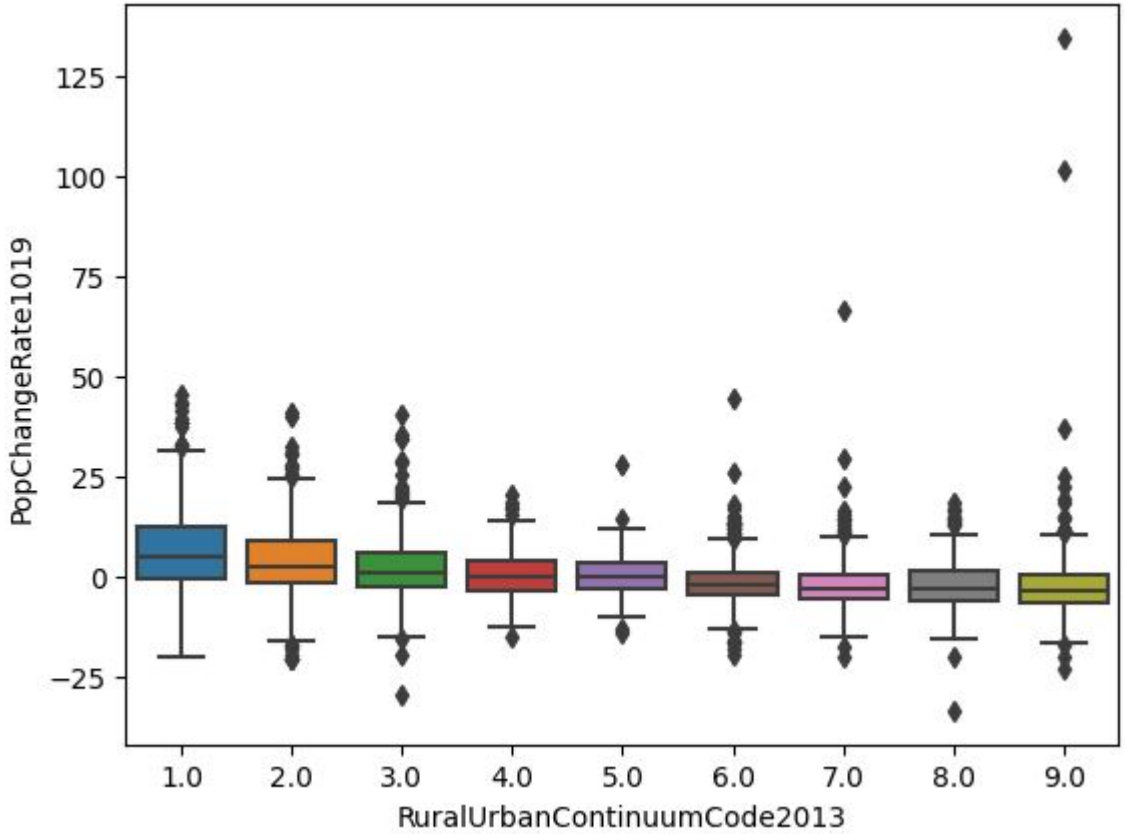
3220 rows × 3 columns

```
county_df = duckdb.sql("""
SELECT State, County, PopChangeRate1019
FROM county_df
ORDER BY PopChangeRate1019 DESC""").df()
```

We can now run SQL queries on the joined data frame

What is unusual about Loving county, TX?

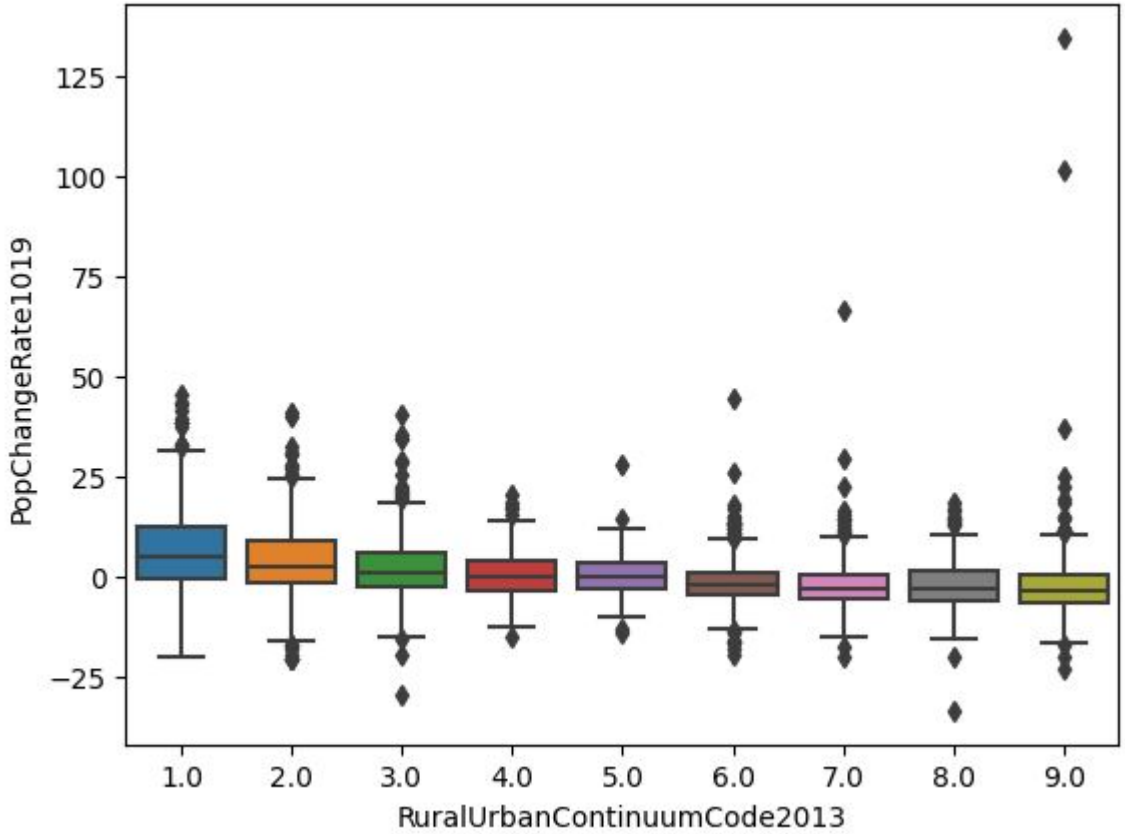
```
seaborn.boxplot(x="RuralUrbanContinuumCode2013",
                y="PopChangeRate1019", data=county_df)
pyplot.show()
```




**Is Code = 9 urban or rural?**

**Which grew faster on average, urban or rural counties?**

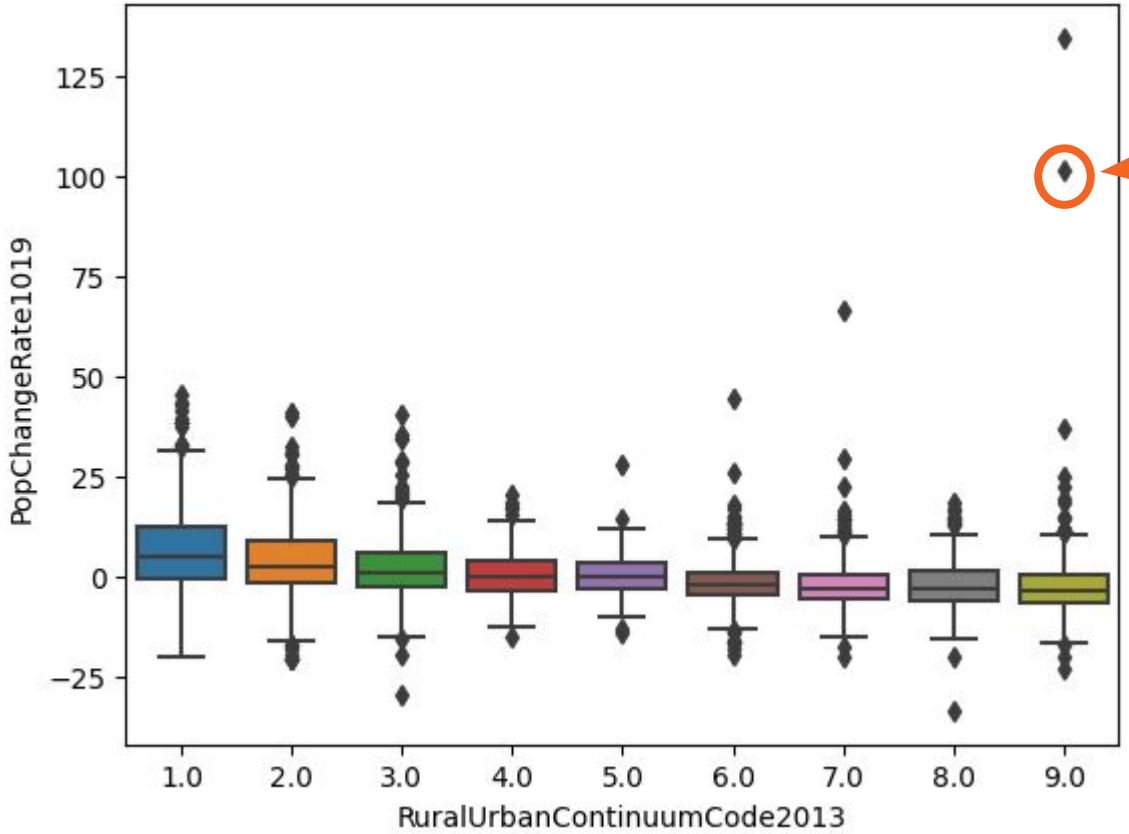
```
seaborn.boxplot(x="RuralUrbanContinuumCode2013",
                y="PopChangeRate1019", data=county_df)
pyplot.show()
```




Is Code = 9 is rural.

Urban counties (closer to code = 1) had higher population change rates than rural ones.

```
seaborn.boxplot(x="RuralUrbanContinuumCode2013",
                y="PopChangeRate1019", data=county_df)
pyplot.show()
```




This is Loving, TX!

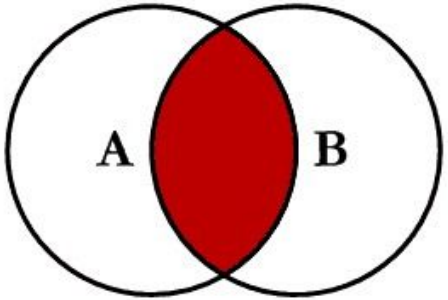
```
seaborn.boxplot(x="RuralUrbanContinuumCode2013",
                y="PopChangeRate1019", data=county_df)
pyplot.show()
```

Loving County is a county in the U.S. state of Texas. With a population of 64 per the 2020 census, it is the least-populous county in the United States with a permanent population. Its county seat and only community is Mentone.

Wikipedia
https://en.wikipedia.org › wiki › Loving_County,_Texas

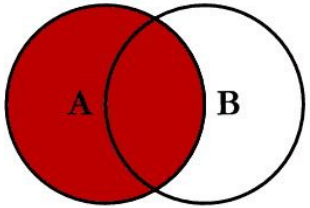
Loving County, Texas - Wikipedia

# Last time: INNER JOIN

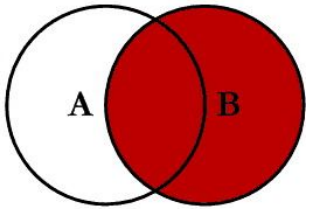



```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

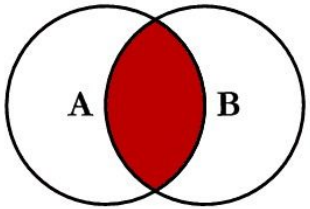
# SQL JOINS



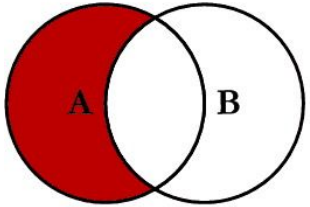

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



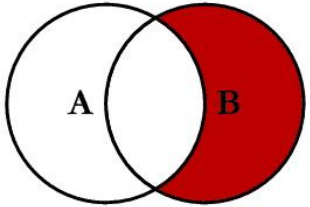

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



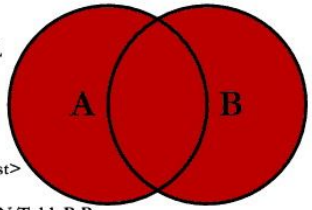

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



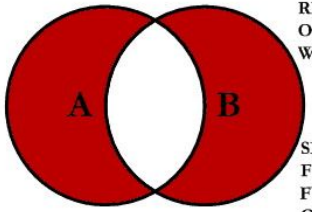

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# df1 and df2: when to skip class?

```
weather = {'day_of_week': ["Monday","Tuesday","Wednesday","Thursday","Friday"],
'weather': [80, 90, 50, -10, 70]}
df1 = pd.DataFrame(weather)
df1
```

```
classes = {
    'course': ["Data Science", "Underwater Basketweaving", "Stats", "LinAlg", "Wines"],
    'weekday': ["Monday","Thursday","Wednesday","Wednesday","Saturday"]}
df2 = pd.DataFrame(classes)
df2
```

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

# What classes do I have when I also know about the weather?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

# What classes do I have when I also know about the weather?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 INNER JOIN df2 \
ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80 | Data Science | Monday |
| 1 | Wednesday | 50 | LinAlg | Wednesday |
| 2 | Thursday | -10 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50 | Stats | Wednesday |

# What classes do I have when I also know about the weather AND I'm willing to go outside?

**Table: `df1`**

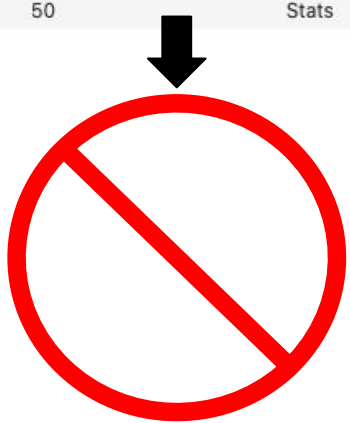
| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 INNER JOIN df2 \
ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80 | Data Science | Monday |
| 1 | Wednesday | 50 | LinAlg | Wednesday |
| 2 | Thursday | -10 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50 | Stats | Wednesday |



# What classes do I have when I also know about the weather AND I'm willing to go outside?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 INNER JOIN df2 \
ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80 | Data Science | Monday |
| 1 | Wednesday | 50 | LinAlg | Wednesday |
| 2 | Thursday | -10 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50 | Stats | Wednesday |

```
duckdb.sql("SELECT day_of_week, weather, course \
            FROM df1 INNER JOIN df2 \
            ON df1.day_of_week = df2.weekday \
            WHERE weather>0").df()
```

| | day_of_week | weather | course |
|---|---|---|---|
| 0 | Monday | 80 | Data Science |
| 1 | Wednesday | 50 | LinAlg |
| 2 | Wednesday | 50 | Stats |

# I care about the 5-day weather forecast, and want to know when of those days I have classes

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

# I care about the 5-day weather forecast, and want to know when of those days I have classes

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT day_of_week, weather, course \
           FROM df1 LEFT JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course |
|---|---|---|---|
| 0 | Monday | 80 | Data Science |
| 1 | Wednesday | 50 | LinAlg |
| 2 | Thursday | -10 | Underwater Basketweaving |
| 3 | Wednesday | 50 | Stats |
| 4 | Tuesday | 90 | None |
| 5 | Friday | 70 | None |

# I care about the 5-day weather forecast, and want to know when of those days I have classes

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT day_of_week, weather, course \
        FROM df1 LEFT JOIN df2 \
        ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course |
|---|---|---|---|
| 0 | Monday | 80 | Data Science |
| 1 | Wednesday | 50 | LinAlg |
| 2 | Thursday | -10 | Underwater Basketweaving |
| 3 | Wednesday | 50 | Stats |
| 4 | Tuesday | 90 | None |
| 5 | Friday | 70 | None |

**How do we sort this?**

# ORDER BY on str is alphabetical!

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT day_of_week, weather, course \
           FROM df1 LEFT JOIN df2 \
           ON df1.day_of_week = df2.weekday \
           ORDER BY day_of_week").df()
```

| | day_of_week | weather | course |
|---|---|---|---|
| 0 | Friday | 70 | None |
| 1 | Monday | 80 | Data Science |
| 2 | Thursday | -10 | Underwater Basketweaving |
| 3 | Tuesday | 90 | None |
| 4 | Wednesday | 50 | LinAlg |
| 5 | Wednesday | 50 | Stats |

# I only care about the days I have classes. What's the weather like then?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

# I only care about the days I have classes. What's the weather like then?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT weekday, course, weather \
           FROM df1 RIGHT JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | weekday | course | weather |
|---|---|---|---|
| 0 | Monday | Data Science | 80.0 |
| 1 | Thursday | Underwater Basketweaving | -10.0 |
| 2 | Wednesday | Stats | 50.0 |
| 3 | Wednesday | LinAlg | 50.0 |
| 4 | Saturday | Wines | NaN |

# Right join: what if I SELECT day_of_week?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT weekday, course, weather \
           FROM df1 RIGHT JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | weekday | course | weather |
|---|---|---|---|
| 0 | Monday | Data Science | 80.0 |
| 1 | Thursday | Underwater Basketweaving | -10.0 |
| 2 | Wednesday | Stats | 50.0 |
| 3 | Wednesday | LinAlg | 50.0 |
| 4 | Saturday | Wines | NaN |

```
duckdb.sql("SELECT day_of_week, course, weather \
           FROM df1 RIGHT JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | course | weather |
|---|---|---|---|
| 0 | Monday | Data Science | 80.0 |
| 1 | Thursday | Underwater Basketweaving | -10.0 |
| 2 | Wednesday | Stats | 50.0 |
| 3 | Wednesday | LinAlg | 50.0 |
| 4 | None | Wines | NaN |

# Same or different output?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

**Code (a):**

```
duckdb.sql("SELECT weekday, course, weather \
           FROM df1 RIGHT JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

**Code (b):**

```
duckdb.sql("SELECT weekday, course, weather \
           FROM df2 LEFT JOIN df1 \
           ON df1.day_of_week = df2.weekday").df()
```

# Same output!

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT weekday, course, weather \
          FROM df1 RIGHT JOIN df2 \
          ON df1.day_of_week = df2.weekday").df()
```

| | weekday | course | weather |
|---|---|---|---|
| 0 | Monday | Data Science | 80.0 |
| 1 | Thursday | Underwater Basketweaving | -10.0 |
| 2 | Wednesday | Stats | 50.0 |
| 3 | Wednesday | LinAlg | 50.0 |
| 4 | Saturday | Wines | NaN |

```
duckdb.sql("SELECT weekday, course, weather \
          FROM df2 LEFT JOIN df1 \
          ON df1.day_of_week = df2.weekday").df()
```

# Left join vs. right join?

- `[FROM a LEFT JOIN b]` **is not** always the same as `[FROM a RIGHT JOIN b]`
- `[FROM a LEFT JOIN b]` **is** always the same as `[FROM b RIGHT JOIN a]`

# Left join vs. right join?

- `[FROM a LEFT JOIN b]` is not always the same as `[FROM a RIGHT JOIN b]`
- `[FROM a LEFT JOIN b]` is always the same as `[FROM b RIGHT JOIN a]`
- In practice… no one uses right joins (except in interviews!)


https://www.reddit.com/r/ProgrammerHumor/comments/10kzn98/right_join_sqls_neglected_stepchild/

# I want to combine all of my data to get the most information I can about both weather and classes.

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

# I want to combine all of my data to get the most information I can about both weather and classes.

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

(also sometimes called an 'outer join')

```
duckdb.sql("SELECT * \
           FROM df1 FULL JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80.0 | Data Science | Monday |
| 1 | Wednesday | 50.0 | LinAlg | Wednesday |
| 2 | Thursday | -10.0 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50.0 | Stats | Wednesday |
| 4 | Tuesday | 90.0 | None | None |
| 5 | Friday | 70.0 | None | None |
| 6 | None | NaN | Wines | Saturday |

# Full join + WHERE no None in ON variables?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 FULL JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80.0 | Data Science | Monday |
| 1 | Wednesday | 50.0 | LinAlg | Wednesday |
| 2 | Thursday | -10.0 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50.0 | Stats | Wednesday |
| 4 | Tuesday | 90.0 | None | None |
| 5 | Friday | 70.0 | None | None |
| 6 | None | NaN | Wines | Saturday |

# Full join + WHERE no None in ON variables?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 FULL JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80.0 | Data Science | Monday |
| 1 | Wednesday | 50.0 | LinAlg | Wednesday |
| 2 | Thursday | -10.0 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50.0 | Stats | Wednesday |
| 4 | Tuesday | 90.0 | None | None |
| 5 | Friday | 70.0 | None | None |
| 6 | None | NaN | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 FULL JOIN df2 \
           ON df1.day_of_week = df2.weekday \
           WHERE day_of_week IS NOT NULL AND \
                 weekday IS NOT NULL").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80 | Data Science | Monday |
| 1 | Wednesday | 50 | LinAlg | Wednesday |
| 2 | Thursday | -10 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50 | Stats | Wednesday |

# Full join + WHERE… look familiar?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 FULL JOIN df2 \
           ON df1.day_of_week = df2.weekday \
           WHERE day_of_week IS NOT NULL AND \
                 weekday IS NOT NULL").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80 | Data Science | Monday |
| 1 | Wednesday | 50 | LinAlg | Wednesday |
| 2 | Thursday | -10 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50 | Stats | Wednesday |

# Full join + WHERE… look familiar?

**Table: `df1`**

| | day_of_week | weather |
|---|---|---|
| 0 | Monday | 80 |
| 1 | Tuesday | 90 |
| 2 | Wednesday | 50 |
| 3 | Thursday | -10 |
| 4 | Friday | 70 |

**Table: `df2`**

| | course | weekday |
|---|---|---|
| 0 | Data Science | Monday |
| 1 | Underwater Basketweaving | Thursday |
| 2 | Stats | Wednesday |
| 3 | LinAlg | Wednesday |
| 4 | Wines | Saturday |

```
duckdb.sql("SELECT * FROM df1 INNER JOIN df2 \
           ON df1.day_of_week = df2.weekday").df()
```

=

```
duckdb.sql("SELECT * FROM df1 FULL JOIN df2 \
           ON df1.day_of_week = df2.weekday \
           WHERE day_of_week IS NOT NULL AND \
                 weekday IS NOT NULL").df()
```

| | day_of_week | weather | course | weekday |
|---|---|---|---|---|
| 0 | Monday | 80 | Data Science | Monday |
| 1 | Wednesday | 50 | LinAlg | Wednesday |
| 2 | Thursday | -10 | Underwater Basketweaving | Thursday |
| 3 | Wednesday | 50 | Stats | Wednesday |

# Joins in the real world

- Your data may not always be clean & matching
  - If you have no duplicate values for your ON variable, and the same values for both tables, then all joins will be the same
  - But this rarely happens in the wild!
- You may need to join more than 2 tables
  - Daisy-chain join statements to merge multiple times

# Bonus interview question: how to get the median in SQL?

- SQL has no built-in median() function!
- But, you can write your own functions to get the median… how?

https://stackoverflow.com/questions/1291152/simple-way-to-calculate-median-with-mysql

# SQL medians without Group By

```
SELECT AVG(dd.val) as median_val

FROM (

SELECT d.val, @rownum:=@rownum+1 as `row_number`, @total_rows:=@rownum

  FROM data d, (SELECT @rownum:=0) r

  WHERE d.val is NOT NULL

  ORDER BY d.val

) as dd

WHERE dd.row_number IN ( FLOOR((@total_rows+1)/2), FLOOR((@total_rows+2)/2) );
```

https://stackoverflow.com/questions/1291152/simple-way-to-calculate-median-with-mysql

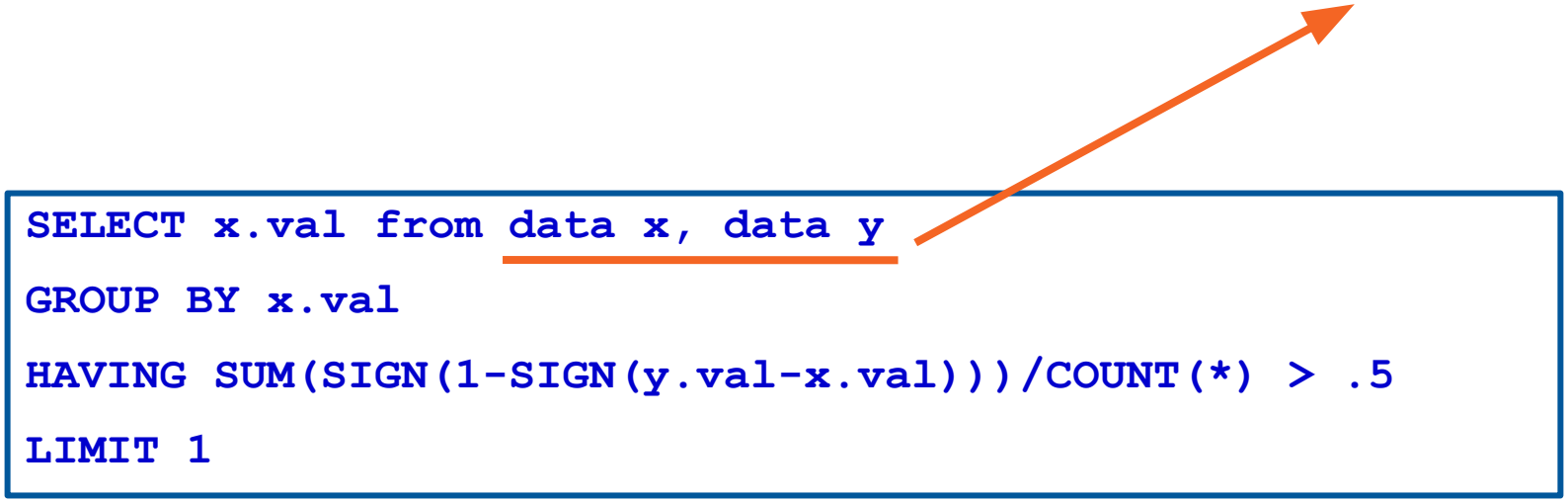
# SQL medians with Group By and Cross Join

Code runs faster & looks cleaner!

```
SELECT x.val from data x, data y
GROUP BY x.val
HAVING SUM(SIGN(1-SIGN(y.val-x.val)))/COUNT(*) > .5
LIMIT 1
```

https://stackoverflow.com/questions/1291152/simple-way-to-calculate-median-with-mysql

# SQL medians with Group By and Cross Join



```
SELECT x.val from data x, data y
GROUP BY x.val
HAVING SUM(SIGN(1-SIGN(y.val-x.val)))/COUNT(*) > .5
LIMIT 1
```

https://stackoverflow.com/questions/1291152/simple-way-to-calculate-median-with-mysql

Please don’t let markers and whiteboards walk away with you!