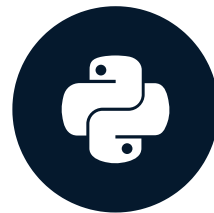# Filtering joins

JOINING DATA WITH PANDAS

Aaren Stubberfield

Instructor

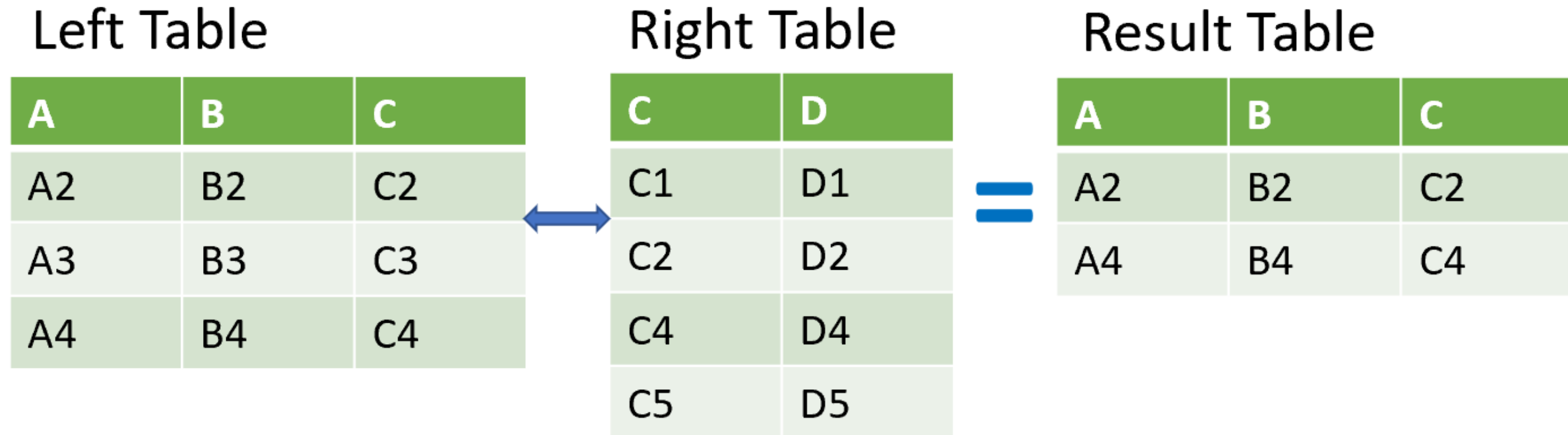# Mutating versus filtering joins

Mutating joins:

- Combines data from two tables based on matching observations in both tables

Filtering joins:

- Filter observations from table based on whether or not they match an observation in another table

What is a semi join?

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

Result Table

A	B	C
A2	B2	C2
A4	B4	C4

Semi joins

- Returns the intersection, similar to an inner join

- Returns only columns from the left table and *not* the right

- No duplicates

Example datasets

```
   gid   name
0  1     Rock
1  2     Jazz
2  3     Metal
3  4     Alternative ...
4  5     Rock And Roll
```

```
   tid   name              aid   mtid   gid   composer          u_price
0  1     For Those Ab...   1     1      1     Angus Young,...   0.99
1  2     Balls to the...   2     2      1     nan               0.99
2  3     Fast As a Shark   3     2      1     F. Baltes, S...   0.99
3  4     Restless and...   3     2      1     F. Baltes, R...   0.99
4  5     Princess of ...   3     2      1     Deaffy & R.A...   0.99
```
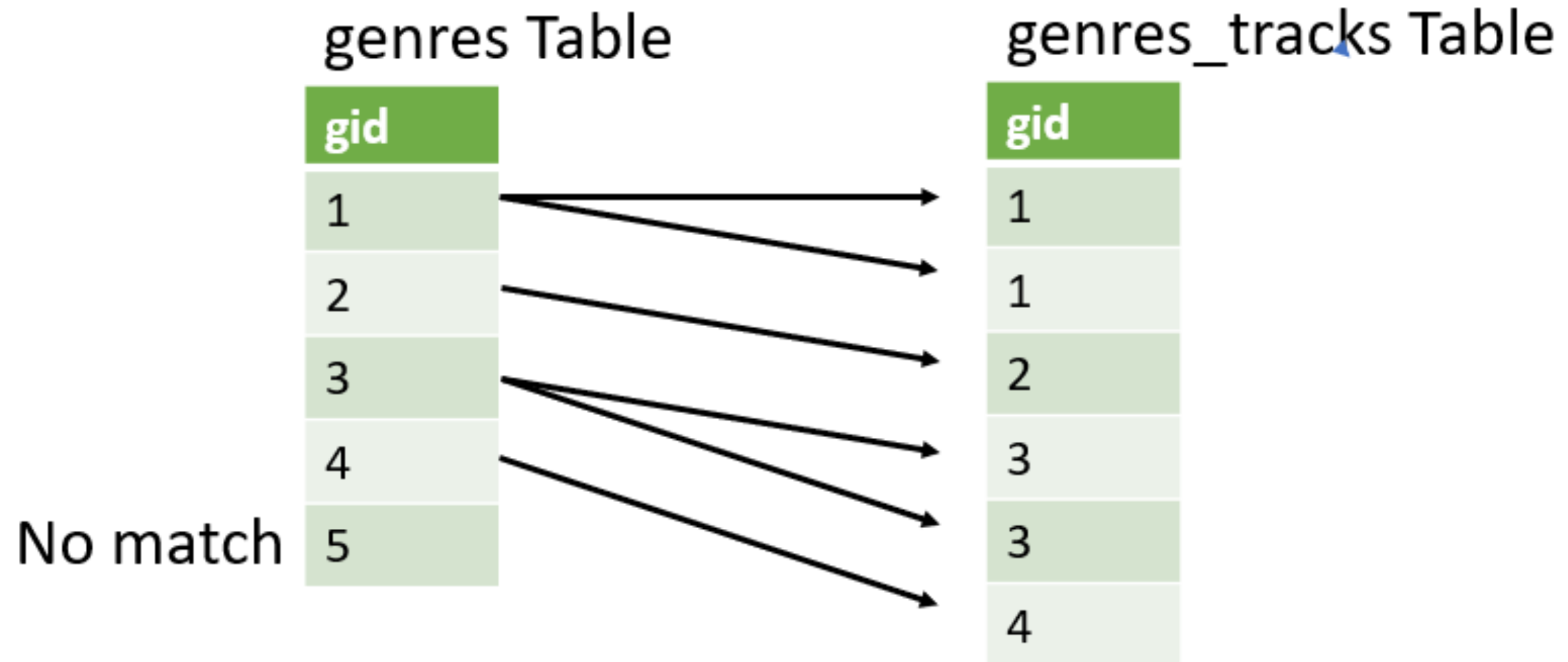
Step 1 - semi join

```python
genres_tracks = genres.merge(top_tracks, on='gid')
print(genres_tracks.head())
```

```
   gid  name_x  tid   name_y           aid  mtid  composer          u_price
0  1    Rock    2260  Don't Stop M...  185  1     Mercury, Fre...   0.99
1  1    Rock    2933  Mysterious Ways  232  1     U2                0.99
2  1    Rock    2618  Speed Of Light   212  1     Billy Duffy/...   0.99
3  1    Rock    2998  When Love Co...  237  1     Bono/Clayton...   0.99
4  1    Rock    685   Who'll Stop ...  54   1     J. C. Fogerty     0.99
```

Step 2 - semi join

```
genres['gid'].isin(genres_tracks['gid'])
```

Step 2 - semi join

```
genres['gid'].isin(genres_tracks['gid'])
```

```
0     True
1     True
2     True
3     True
4    False
Name: gid, dtype: bool
```

Step 3 - semi join

```python
genres_tracks = genres.merge(top_tracks, on='gid')
top_genres = genres[genres['gid'].isin(genres_tracks['gid'])]
print(top_genres.head())
```

```
   gid  name
0  1    Rock
1  2    Jazz
2  3    Metal
3  4    Alternative & Punk
4  6    Blues
```

What is an anti join?

Anti join:

- Returns the left table, excluding the intersection

- Returns only columns from the left table and *not* the right

Step 1 - anti join

```python
genres_tracks = genres.merge(top_tracks, on='gid', how='left', indicator=True)
print(genres_tracks.head())
```

	gid	name_x	tid	name_y	aid	mtid	composer	u_price	_merge
0	1	Rock	2260.0	Don't Stop M...	185.0	1.0	Mercury, Fre...	0.99	both
1	1	Rock	2933.0	Mysterious Ways	232.0	1.0	U2	0.99	both
2	1	Rock	2618.0	Speed Of Light	212.0	1.0	Billy Duffy/...	0.99	both
3	1	Rock	2998.0	When Love Co...	237.0	1.0	Bono/Clayton...	0.99	both
4	5	Rock And Roll	NaN	NaN	NaN	NaN	NaN	NaN	left_only

Step 2 - anti join

```python
gid_list = genres_tracks.loc[genres_tracks['_merge'] == 'left_only', 'gid']
print(gid_list.head())
```

```
23     5
34     9
36    11
37    12
38    13
Name: gid, dtype: int64
```

Step 3 - anti join

```python
genres_tracks = genres.merge(top_tracks, on='gid', how='left', indicator=True)
gid_list = genres_tracks.loc[genres_tracks['_merge'] == 'left_only','gid']
non_top_genres = genres[genres['gid'].isin(gid_list)]
print(non_top_genres.head())
```

```
   gid   name
0  5     Rock And Roll
1  9     Pop
2  11    Bossa Nova
3  12    Easy Listening
4  13    Heavy Metal
```

Concatenate DataFrames together vertically

JOINING DATA WITH PANDAS
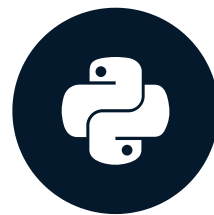
Aaren Stubberfield
Instructor

Concatenate two tables vertically

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

A	B	C
A4	B4	C4
A5	B5	C5
A6	B6	C6

- pandas `.concat()` method can concatenate both vertical and horizontal.
  - `axis=0`, vertical

Basic concatenation

- 3 different tables

- Same column names

- Table variable names:
  - `inv_jan` *(top)*
  - `inv_feb` *(middle)*
  - `inv_mar` *(bottom)*

```
   iid  cid  invoice_date  total
0  1    2    2009-01-01    1.98
1  2    4    2009-01-02    3.96
2  3    8    2009-01-03    5.94
```

```
   iid  cid  invoice_date  total
0  7    38   2009-02-01    1.98
1  8    40   2009-02-01    1.98
2  9    42   2009-02-02    3.96
```

```
   iid  cid  invoice_date  total
0  14   17   2009-03-04    1.98
1  15   19   2009-03-04    1.98
2  16   21   2009-03-05    3.96
```

Basic concatenation

```
pd.concat([inv_jan, inv_feb, inv_mar])
```

	iid	cid	invoice_date	total
0	1	2	2009-01-01	1.98
1	2	4	2009-01-02	3.96
2	3	8	2009-01-03	5.94
0	7	38	2009-02-01	1.98
1	8	40	2009-02-01	1.98
2	9	42	2009-02-02	3.96
0	14	17	2009-03-04	1.98
1	15	19	2009-03-04	1.98
2	16	21	2009-03-05	3.96

Ignoring the index

```
pd.concat([inv_jan, inv_feb, inv_mar],
          ignore_index=True)
```

```
    iid   cid   invoice_date   total
0   1     2     2009-01-01     1.98
1   2     4     2009-01-02     3.96
2   3     8     2009-01-03     5.94
3   7     38    2009-02-01     1.98
4   8     40    2009-02-01     1.98
5   9     42    2009-02-02     3.96
6   14    17    2009-03-04     1.98
7   15    19    2009-03-04     1.98
8   16    21    2009-03-05     3.96
```

Setting labels to original tables

```
pd.concat([inv_jan, inv_feb, inv_mar],
          ignore_index=False,
          keys=['jan','feb','mar'])
```

		iid	cid	invoice_date	total
jan	0	1	2	2009-01-01	1.98
	1	2	4	2009-01-02	3.96
	2	3	8	2009-01-03	5.94
feb	0	7	38	2009-02-01	1.98
	1	8	40	2009-02-01	1.98
	2	9	42	2009-02-02	3.96
mar	0	14	17	2009-03-04	1.98
	1	15	19	2009-03-04	1.98
	2	16	21	2009-03-05	3.96

Concatenate tables with different column names

Table: `inv_jan`

```
   iid   cid   invoice_date   total
0  1     2     2009-01-01     1.98
1  2     4     2009-01-02     3.96
2  3     8     2009-01-03     5.94
```

Table: `inv_feb`

```
   iid   cid   invoice_date   total   bill_ctry
0  7     38    2009-02-01     1.98    Germany
1  8     40    2009-02-01     1.98    France
2  9     42    2009-02-02     3.96    France
```

Concatenate tables with different column names

```
pd.concat([inv_jan, inv_feb],
          sort=True)
```

|	bill_ctry	cid	iid	invoice_date	total
0	NaN	2	1	2009-01-01	1.98
1	NaN	4	2	2009-01-02	3.96
2	NaN	8	3	2009-01-03	5.94
0	Germany	38	7	2009-02-01	1.98
1	France	40	8	2009-02-01	1.98
2	France	42	9	2009-02-02	3.96

Concatenate tables with different column names

```
pd.concat([inv_jan, inv_feb],
          join='inner')
```

```
iid   cid   invoice_date   total
1     2     2009-01-01     1.98
2     4     2009-01-02     3.96
3     8     2009-01-03     5.94
7     38    2009-02-01     1.98
8     40    2009-02-01     1.98
9     42    2009-02-02     3.96
```

Verifying integrity

JOINING DATA WITH PANDAS
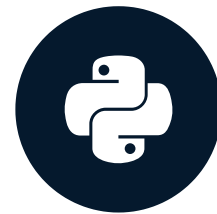
Aaren Stubberfield
Instructor

Let's check our data

Possible merging issue:

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

↔

C	D
C1	D1
C1	D2
C1	D3
C2	D4

- Unintentional one-to-many relationship

- Unintentional many-to-many relationship

Possible concatenating issue:

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

↕

A	B	C
A3 (duplicate)	B3 (duplicate)	C3 (duplicate)
A4	B4	C4
A5	B5	C5

- Duplicate records possibly unintentionally introduced

Validating merges

`.merge(validate=None)` :

- Checks if merge is of specified type

- `'one_to_one'`

- `'one_to_many'`

- `'many_to_one'`

- `'many_to_many'`

Merge dataset for example

Table Name: `tracks`

```
   tid  name              aid  mtid  gid  u_price
0  2    Balls to the...   2    2     1    0.99
1  3    Fast As a Shark   3    2     1    0.99
2  4    Restless and...   3    2     1    0.99
```

Table Name: `specs`

```
   tid  milliseconds  bytes
0  2    342562        5510424
1  3    230619        3990994
2  2    252051        4331779
```

Merge validate: one_to_one

```
tracks.merge(specs, on='tid',
             validate='one_to_one')
```

```
Traceback (most recent call last):
MergeError: Merge keys are not unique in right dataset; not a one-to-one merge
```

Merge validate: one_to_many

```
albums.merge(tracks, on='aid',
             validate='one_to_many')
```

	aid	title	artid	tid	name	mtid	gid	u_price
0	2	Balls to the...	2	2	Balls to the...	2	1	0.99
1	3	Restless and...	2	3	Fast As a Shark	2	1	0.99
2	3	Restless and...	2	4	Restless and...	2	1	0.99

Verifying concatenations

`.concat(verify_integrity=False)`:

- Check whether the new concatenated index contains duplicates

- Default value is `False`

Dataset for .concat() example

Table Name: `inv_feb`

```
       cid   invoice_date   total
iid
7      38    2009-02-01     1.98
8      40    2009-02-01     1.98
9      42    2009-02-02     3.96
```

Table Name: `inv_mar`

```
       cid   invoice_date   total
iid
9      17    2009-03-04     1.98
15     19    2009-03-04     1.98
16     21    2009-03-05     3.96
```

Verifying concatenation: example

```
pd.concat([inv_feb, inv_mar],
          verify_integrity=True)
```

```
pd.concat([inv_feb, inv_mar],
          verify_integrity=False)
```

```
Traceback (most recent call last):
ValueError: Indexes have overlapping
values: Int64Index([9], dtype='int64',
name='iid')
```

```
       cid  invoice_date  total
iid
7      38   2009-02-01    1.98
8      40   2009-02-01    1.98
9      42   2009-02-02    3.96
9      17   2009-03-04    1.98
15     19   2009-03-04    1.98
16     21   2009-03-05    3.96
```

Why verify integrity and what to do

Why:

- Real world data is often *NOT* clean

What to do:

- Fix incorrect data

- Drop duplicate rows