



INTRODUCTION TO DATABASES IN PYTHON

Filtering and Targeting Data



Where Clauses

```
In [1]: stmt = select([census])
```

```
In [2]: stmt = stmt.where(census.columns.state ==  
                          'California')
```

```
In [3]: results = connection.execute(stmt).fetchall()
```

```
In [4]: for result in results:  
    ...:     print(result.state, result.age)
```

```
Out[4]:
```

```
California 0
```

```
California 1
```

```
California 2
```

```
California 3
```

```
California 4
```

```
California 5
```

```
...
```



Where Clauses

- Restrict data returned by a query based on boolean conditions
- Compare a column against a value or another column
- Often used comparisons: '==', '<=', '>=', or '!='

Expressions

- Provide more complex conditions than simple operators
- Eg. `in_()`, `like()`, `between()`
- Many more in documentation
- Available as method on a Column



Expressions

```
In [1]: stmt = select([census])
```

```
In [2]: stmt = stmt.where(  
        census.columns.state.startswith('New'))
```

```
In [3]: for result in connection.execute(stmt):  
        ....:     print(result.state, result.pop2000)
```

```
Out[3]:
```

```
New Jersey 56983
```

```
New Jersey 56686
```

```
New Jersey 57011
```

```
...
```



Conjunctions

- Allow us to have multiple criteria in a where clause
- Eg. `and_()`, `not_()`, `or_()`



Conjunctions

```
In [1]: from sqlalchemy import or_
```

```
In [2]: stmt = select([census])
```

```
In [3]: stmt = stmt.where(  
...:     or_(census.columns.state == 'California',  
...:         census.columns.state == 'New York')  
...: )  
...: )
```

```
In [4]: for result in connection.execute(stmt):  
...:     print(result.state, result.sex)
```

```
Out[4]:  
New York M  
...  
California F
```



INTRODUCTION TO DATABASES IN PYTHON

Let's practice!



INTRODUCTION TO DATABASES IN PYTHON

Ordering Query Results



Order by Clauses

- Allows us to control the order in which records are returned in the query results
- Available as a method on statements `order_by()`



Order by Ascending

```
In [1]: print(results[:10])
```

```
Out[1]: [('Illinois',), ...]
```

```
In [3]: stmt = select([census.columns.state])
```

```
In [4]: stmt = stmt.order_by(census.columns.state)
```

```
In [5]: results = connection.execute(stmt).fetchall()
```

```
In [6]: print(results[:10])
```

```
Out[6]: [('Alabama',), ...]
```



Order by Descending

- Wrap the column with `desc()` in the `order_by()` clause



Order by Multiple

- Just separate multiple columns with a comma
- Orders completely by the first column
- Then if there are duplicates in the first column, orders by the second column
- repeat until all columns are ordered



Order by Multiple

```
In [6]: print(results)
Out[6]: ('Alabama', 'M')

In [7]: stmt = select([census.columns.state,
....:                  census.columns.sex])

In [8]: stmt = stmt.order_by(census.columns.state,
....:                        census.columns.sex)

In [9]: results = connection.execute(stmt).first()

In [10]: print(results)
Out[10]: ('Alabama', 'F')
('Alabama', 'F')
...
('Alabama', 'M')
```



INTRODUCTION TO DATABASES IN PYTHON

Let's practice!



INTRODUCTION TO DATABASES IN PYTHON

Counting, Summing and Grouping Data



SQL Functions

- E.g. Count, Sum
- `from sqlalchemy import func`
- More efficient than processing in Python
- Aggregate data



Sum Example

```
In [1]: from sqlalchemy import func

In [2]: stmt = select([func.sum(census.columns.pop2008)])

In [3]: results = connection.execute(stmt).scalar()

In [4]: print(results)
Out[4]: 302876613
```



Group by

- Allows us to group row by common values



Group by

```
In [1]: stmt = select([census.columns.sex,
...:                  func.sum(census.columns.pop2008)
...:                  ])

In [2]: stmt = stmt.group_by(census.columns.sex)

In [3]: results = connection.execute(stmt).fetchall()

In [4]: print(results)
Out[4]: [('F', 153959198), ('M', 148917415)]
```



Group by

- Supports multiple columns to group by with a pattern similar to `order_by()`
- Requires all selected columns to be grouped or aggregated by a function



Group by Multiple

```
In [1]: stmt = select([census.columns.sex,  
....:                  census.columns.age,  
....:                  func.sum(census.columns.pop2008)  
....: ])
```

```
In [2]: stmt = stmt.group_by(census.columns.sex,  
....:                        census.columns.age)
```

```
In [2]: results = connection.execute(stmt).fetchall()
```

```
In [3]: print(results)
```

```
Out[3]:
```

```
[('F', 0, 2105442), ('F', 1, 2087705), ('F', 2, 2037280), ('F', 3,  
2012742), ('F', 4, 2014825), ('F', 5, 1991082), ('F', 6, 1977923),  
('F', 7, 2005470), ('F', 8, 1925725), ...]
```



Handling ResultSets from Functions

- SQLAlchemy auto generates “column names” for functions in the ResultSet
- The column names are often `func_#` such as `count_1`
- Replace them with the `label()` method



Using label()

```
In [1]: print(results[0].keys())
```

```
Out[1]: ['sex', u'sum_1']
```

```
In [2]: stmt = select([census.columns.sex,  
...:                  func.sum(census.columns.pop2008).label(  
...:                        'pop2008_sum')  
...: ])
```

```
In [3]: stmt = stmt.group_by(census.columns.sex)
```

```
In [4]: results = connection.execute(stmt).fetchall()
```

```
In [5]: print(results[0].keys())
```

```
Out[5]: ['sex', 'pop2008_sum']
```



INTRODUCTION TO DATABASES IN PYTHON

Let's practice!



INTRODUCTION TO DATABASES IN PYTHON

SQLAlchemy and Pandas for Visualization

SQLAlchemy and Pandas

- DataFrame can take a SQLAlchemy ResultSet
- Make sure to set the DataFrame columns to the ResultSet keys



DataFrame Example

```
In [1]: import pandas as pd
```

```
In [2]: df = pd.DataFrame(results)
```

```
In [3]: df.columns = results[0].keys()
```

```
In [4]: print(df)
```

```
Out[4]:
```

	sex	pop2008_sum
0	F	2105442
1	F	2087705
2	F	2037280
3	F	2012742
4	F	2014825
5	F	1991082



Graphing

- We can graph just like we would normally



Graphing Example

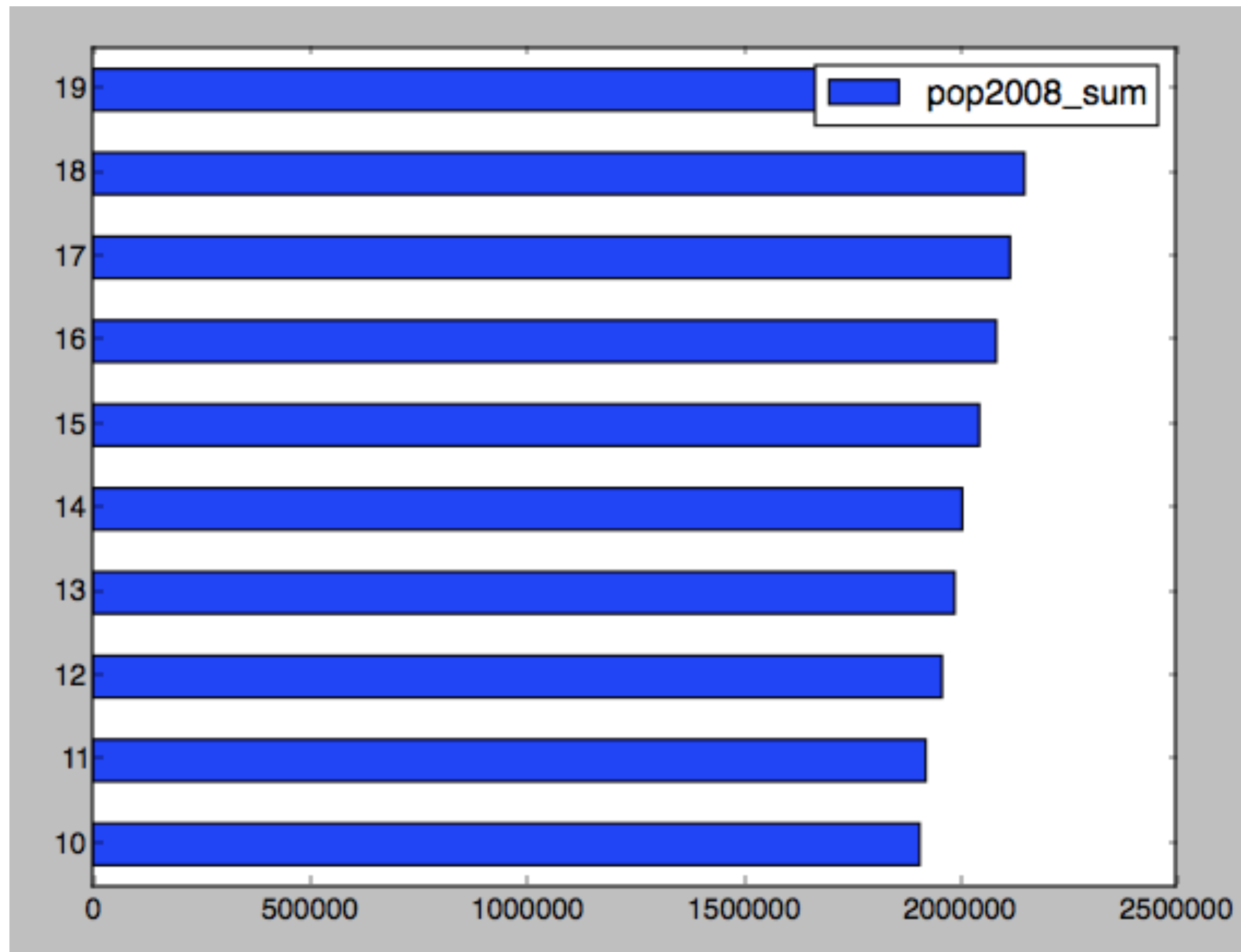
```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: df[10:20].plot.barh()
```

```
In [3]: plt.show()
```



Graphing Output





INTRODUCTION TO DATABASES IN PYTHON

Let's practice!