# CS6 Practical System Skills

**Fall 2019 edition**

Leonhard Spiegelberg
lspiegel@cs.brown.edu

# 21.98 Logistics

⇒ Last week of lectures

⇒ Final projects due **15th Dec**

⇒ Presentations for final projects: **15th Dec, 3pm-5pm**.

⇒ Details on Piazza

⇒ TA hours till **Sun, 8th Dec midnight**. No TA hours next week.

⇒ please check-in with your project TA

⇒ Last Lab today on how to use a SSL certificate with your app.

# 21.99 Review - Practical Flask

Last lecture:

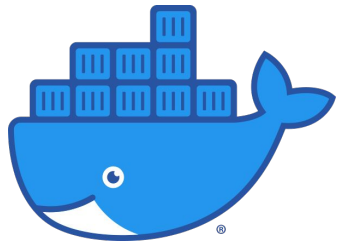
- **flask_login**
  → add quickly (safe) user-login functionality
  → protect routes via login_required decorator
- **Docker**
  → deploy database and flask application in separate containers
  → Lab: Use docker-composer to start/watch containers
  → Code: github.com/browncs6/FlaskExamples

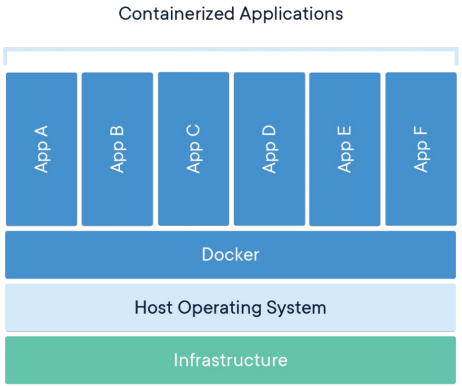
# 21.99 Deployment via docker-composer

docker-compose.yml

```yaml
version: '3'
services:
  login:
    image: "login:latest"
    ports:
      - "80:5000"
    env_file: .env
    links:
      - postgres:dbserver
    restart: always
  postgres:
    image: "postgres:12.1"
    env_file: .env-postgres
    restart: always
    volumes:
      - ${DATA_DIR}:/var/lib/postgresql/data
```

```
export DATA_DIR=/db-data
&& docker-compose up -d
```





# 22 Dataframes & Viz

## CS6 Practical System Skills

Fall 2019

Leonhard Spiegelberg *lspiegel@cs.brown.edu*

# 22.01 What are DataFrames?

⇒ For manipulating data we've learned the following 3 tools

1. (Relational) Databases
2. pure Python & hand coded data pipelines
3. Shell tools like awk/sed/uniq/sort …

⇒ DataFrames are table-like data structures, popular amongst data scientists to manipulate tabular data quickly.

→ typically use one DataFrame library to manipulate small to medium sized datasets (there exist frameworks for large datasets too!).

→ use when tasks are too complicated for shell tools, writing a pipeline by hand is a waste of time and a database is overkill (you don't need ACID for analytics!)

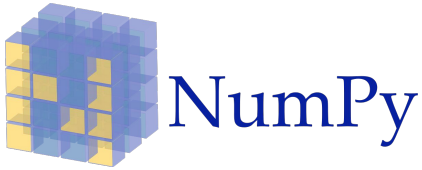
# 22.01 When to use DataFrames?

⇒ Typical tasks include

- **loading**/combining/saving data from/to CSV/JSON/Excel files or from results of a SQL query/to a database
- **filtering** a DataFrame down to rows and columns of interest
- **cleaning** values with arithmetic and string operations
- **summarizing** groups of rows, aggregates
- **computing** new columns based on existing ones
- **joining** data frames with others
- **plotting**/visualizing data

# 22.02 DataFrames in Python

⇒ **The** DataFrame library for Python is Pandas which is based on Numpy, a powerful library for multi-dimensional data
→ install via `pip3 install pandas`




⇒ Numpy adds support for addition, subtraction, multiplication & more to lists of numbers

⇒ If you take a Data Science or ML class, these will become your best friends

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([5, 4, 3, 2])

a + b
```

# 22.03 Pandas - Foundations

⇒ A Dataframe is made up of rows and columns.

→ in Pandas, each column is represented by a *Series* object, which itself holds the values and an index (default: `0, …, #numelements - 1`).

→ The Dataframe is thus a collection of named Series objects (i.e. the columns)

⇒ Note: Most data scientists prefer working with Dataframes in a notebook

```
import pandas as pd

colA = pd.Series([10, 20, 30], index=[1, 3, 4])

colB = pd.Series([9, -3, -2.41], index=[0, 1, 2])

df = pd.DataFrame({'columnA' : colA, 'columnB' : colB})
```



| | columnA | columnB |
|---|---|---|
| **0** | NaN | 9.00 |
| **1** | 10.0 | -3.00 |
| **2** | NaN | -2.41 |
| **3** | 20.0 | NaN |
| **4** | 30.0 | NaN |

# Pandas I/O

# 22.04 Pandas - Loading data I/III

⇒ There are multiple ways to load data into a dataframe

⇒ (1) Load data from main-memory (i.e. python objects)

**List of dicts**

```
records = [{'A' : 20, 'B' : 'Tux', 'C' :
3.141},
           {'A' : None, 'B' : 'Sealion'},
           {'A' : 10, 'B': 'Crabby', 'C' : 6.0}]

pd.DataFrame(records)
```



| | A | B | C |
|---|---|---|---|
| 0 | 20.0 | Tux | 3.141 |
| 1 | NaN | Sealion | NaN |
| 2 | 10.0 | Crabby | 6.000 |

**List of tuples**

```
records = [(20, 'Tux', 3.141),
           (None, 'Sealion', np.NaN),
           (10, 'Crabby', 6.0)]

pd.DataFrame(records,
             columns=['A', 'B', 'C'])
```



| | A | B | C |
|---|---|---|---|
| 0 | 20.0 | Tux | 3.141 |
| 1 | NaN | Sealion | NaN |
| 2 | 10.0 | Crabby | 6.000 |

# 22.04 Pandas - Loading data II/III

⇒ (2) Load data directly from CSV/JSON/Excel files

**CSV**

```
a,b,c
1,2,3
4,5,6
7,8,9
```

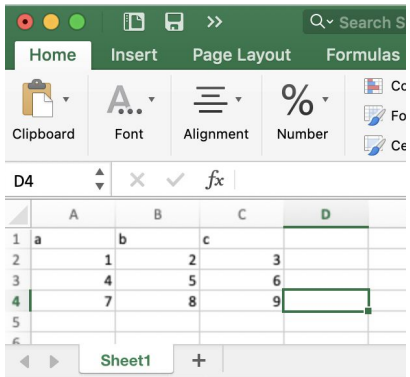
```
pd.read_csv('sample.csv')
```

**JSON**

```
{"a":1,"b":2,"c":3}
{"a":4,"b":5,"c":6}
{"a":7,"b":8,"c":9}
```

```
pd.read_json('sample.json',
             orient='records',
             lines=True)
```

**Excel**




```
pd.read_excel('sample.xlsx',
              'Sheet1')
```

# 22.04 Pandas - Loading data III/III

⇒ (3) From a database

```
import sqlalchemy

dburi = 'postgresql://postgres:docker@localhost/postgres'

db = sqlalchemy.create_engine(dburi)

pd.read_sql('SELECT * FROM sample', db)
```

Start postgresql database via docker image! I.e. to run the above code use

```
docker run -p 5432:5432 -e POSTGRES_PASSWORD=docker -v \
    $PWD/data:/var/lib/postgresql/data --rm postgres
```

# 22.05 Pandas - Saving data

| target | commands |
| --- | --- |
| main-memory | `df.to_dict(orient='records')  # list of dicts`<br><br>`list(df.to_records(index=None))  # list of tuples` |
| CSV / JSON / Excel | `df.to_csv('sample.csv', index=None)`<br><br>`df.to_json('sample.json',`<br>`  orient='records',`<br>`  lines=True)`<br><br>`df.to_excel('sample.xlsx', index=None)` |
| Database<br>(via SQLAlchemy) | `df.to_sql('sample', db, index=None, if_exists='replace')` |

# Manipulating data

# 22.06 Manipulating data - columns I/II

⇒ columns can be added or replaced by Numpy operations

⇒ to get the series corresponding to column label, use
`df[label]`
→ to get the i-th column, use `df[df.columns[i]]`

⇒ to get a subset (i.e. another dataframe) of columns, use
`df[[label1, label2, …, labelN]]`

| | a | b | c |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 |
| **2** | 7 | 8 | 9 |

```
df[['c', 'a']]
```

| | c | a |
|---|---|---|
| **0** | 3 | 1 |
| **1** | 6 | 4 |
| **2** | 9 | 7 |

# 22.06 Manipulating data - columns II/II

⇒ instead of directly applying a operation on a column,
apply allows to use a user defined function across a column

```
# manipulating columnmns via element-wise Numpy operations
df['a^2 - b^2'] = df['a'] * df['a'] + df['b'] * df['b']

# apply over a single column
df['fmt'] = df['a'].apply(lambda x: '{:04d}'.format(x))

# apply using multiple columns
df['a+b'] = df[['a', 'b']].apply(lambda row: row['a'] + row['b'], axis=1)
```

| | a | b | c |
|---|---|---|---|
| **0** | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 |
| **2** | 7 | 8 | 9 |



| | a | b | c | a^2 - b^2 | fmt | a+b |
|---|---|---|---|---|---|---|
| **0** | 1 | 2 | 3 | 5 | 0001 | 3 |
| **1** | 4 | 5 | 6 | 41 | 0004 | 9 |
| **2** | 7 | 8 | 9 | 113 | 0007 | 15 |

# 22.07 Manipulating data - rows

⇒ to get a subset of rows, you can use `.head()` or `.tail()`

⇒ the other option is to use label-based indexing (`.loc`) or index-based indexing (`.iloc`)

→ you can also use `df[<idx>]` with a logical/boolean index, e.g. `df['a'] > 10`.

⇒ Indexing in Pandas DataFrames:

`df.iloc[<row sel>, <col sel>]`

selection have to be integer based, can be 1. single values, e.g. 0 2. slice of rows, e.g. [4:7] or 3. integer list of rows, e.g. [2, 5, 7]

`df.loc[<row sel>, <col sel>]`

selection have to be label/index based, can be 1. single label, e.g. 'column' 2. list of labels ['a', 'b'] or 3. for rows: boolean index, i.e. df['a'] > 10, for columns: slice, i.e. 'Boston':'Providence'

Note: To assign new values, use iloc or loc!

# 22.07 Indexing - Examples

| | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Jane** | NY | blue | Steak | 30 | 165 | 4.6 |
| **Niko** | TX | green | Lamb | 2 | 70 | 8.3 |
| **Aaron** | FL | red | Mango | 12 | 120 | 9.0 |
| **Penelope** | AL | white | Apple | 4 | 80 | 3.3 |
| **Dean** | AK | gray | Cheese | 32 | 180 | 1.8 |
| **Christina** | TX | black | Melon | 33 | 172 | 9.5 |
| **Cornelia** | TX | red | Beans | 69 | 150 | 2.2 |

```
df[df['age'] < 10]
```

| | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Niko** | TX | green | Lamb | 2 | 70 | 8.3 |
| **Penelope** | AL | white | Apple | 4 | 80 | 3.3 |

```
df.iloc[[0, 1], 3:]
```

| | age | height | score |
|---|---|---|---|
| **Jane** | 30 | 165 | 4.6 |
| **Niko** | 2 | 70 | 8.3 |

```
df['Aaron':, ['food', 'state']]
```

| | food | state |
|---|---|---|
| **Aaron** | Mango | FL |
| **Penelope** | Apple | AL |
| **Dean** | Cheese | AK |
| **Christina** | Melon | TX |
| **Cornelia** | Beans | TX |

# 22.07 Manipulating data - rows

⇒ Filtering can be done in Pandas using boolean indices.

→ simplest index is a (numpy) array of booleans, used to index a dataframe, e.g. `df[[True, False, …]]`

⇒ Can easily create an index using e.g. `df['column'] > 10`

⇒ To combine multiple boolean indices, use `&` (and) and `|` (or)

→ to use `&` and `I`, indices must be of type Pandas Index/numpy arrays.

| | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Jane** | NY | blue | Steak | 30 | 165 | 4.6 |
| **Niko** | TX | green | Lamb | 2 | 70 | 8.3 |
| **Aaron** | FL | red | Mango | 12 | 120 | 9.0 |
| **Penelope** | AL | white | Apple | 4 | 80 | 3.3 |
| **Dean** | AK | gray | Cheese | 32 | 180 | 1.8 |
| **Christina** | TX | black | Melon | 33 | 172 | 9.5 |
| **Cornelia** | TX | red | Beans | 69 | 150 | 2.2 |

```
df[(df['age'] < 10) |
(df['age'] > 65)]
```

| | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Niko** | TX | green | Lamb | 2 | 70 | 8.3 |
| **Penelope** | AL | white | Apple | 4 | 80 | 3.3 |
| **Cornelia** | TX | red | Beans | 69 | 150 | 2.2 |

```
df[(df['age'] < 20) &
(df['height'] > 100)]
```

| | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Aaron** | FL | red | Mango | 12 | 120 | 9.0 |

# Summarizing, Grouping, Aggregating

# 22.08 More on indices

⇒ there are 2 functions which can be used to either convert an index to a column or a column to an index

→ to (re)name an index, use `df.index.rename('...', inplace=True)` or overwrite `df.index`

(1)
```
df.index =
df.index.rename('name')
df.reset_index()
```

(2)
```
df.set_index('name')
```

| | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Jane** | NY | blue | Steak | 30 | 165 | 4.6 |
| **Niko** | TX | green | Lamb | 2 | 70 | 8.3 |
| **Aaron** | FL | red | Mango | 12 | 120 | 9.0 |
| **Penelope** | AL | white | Apple | 4 | 80 | 3.3 |
| **Dean** | AK | gray | Cheese | 32 | 180 | 1.8 |
| **Christina** | TX | black | Melon | 33 | 172 | 9.5 |
| **Cornelia** | TX | red | Beans | 69 | 150 | 2.2 |

(1)

| | name | state | color | food | age | height | score |
|---|---|---|---|---|---|---|---|
| **0** | Jane | NY | blue | Steak | 30 | 165 | 4.6 |
| **1** | Niko | TX | green | Lamb | 2 | 70 | 8.3 |
| **2** | Aaron | FL | red | Mango | 12 | 120 | 9.0 |
| **3** | Penelope | AL | white | Apple | 4 | 80 | 3.3 |
| **4** | Dean | AK | gray | Cheese | 32 | 180 | 1.8 |
| **5** | Christina | TX | black | Melon | 33 | 172 | 9.5 |
| **6** | Cornelia | TX | red | Beans | 69 | 150 | 2.2 |

(2)

| name | state | color | food | age | height | score |
|---|---|---|---|---|---|---|
| **Jane** | NY | blue | Steak | 30 | 165 | 4.6 |
| **Niko** | TX | green | Lamb | 2 | 70 | 8.3 |
| **Aaron** | FL | red | Mango | 12 | 120 | 9.0 |
| **Penelope** | AL | white | Apple | 4 | 80 | 3.3 |
| **Dean** | AK | gray | Cheese | 32 | 180 | 1.8 |
| **Christina** | TX | black | Melon | 33 | 172 | 9.5 |
| **Cornelia** | TX | red | Beans | 69 | 150 | 2.2 |

# 22.09 Grouping

⇒ use `.groupby` to group values based on one or more columns.

⇒ Either retrieve groups via the property .groups or perform an aggregate like `.count/.describe/.mean/.std/.agg(...)`

⇒ You can run these aggregates also over the whole DataFrame, e.g. `df.count()`

| | A | B | C |
|---|---|---|---|
| 0 | foo | one | 1 |
| 1 | bar | one | 4 |
| 2 | foo | two | 7 |
| 3 | bar | three | 2 |
| 4 | foo | two | 3 |
| 5 | bar | two | 1 |
| 6 | foo | one | 5 |
| 7 | foo | three | 5 |

```
df.groupby('A') \
  .count()
```

| A | B | C |
|---|---|---|
| bar | 3 | 3 |
| foo | 5 | 5 |

| | A | B | C |
|---|---|---|---|
| 0 | foo | one | 1 |
| 1 | bar | one | 4 |
| 2 | foo | two | 7 |
| 3 | bar | three | 2 |
| 4 | foo | two | 3 |
| 5 | bar | two | 1 |
| 6 | foo | one | 5 |
| 7 | foo | three | 5 |

```
df.groupby(['A', 'B']) \
  .count()
```

| | | C |
|---|---|---|
| A | B | |
| bar | one | 1 |
| | three | 1 |
| | two | 1 |
| foo | one | 2 |
| | three | 1 |
| | two | 2 |

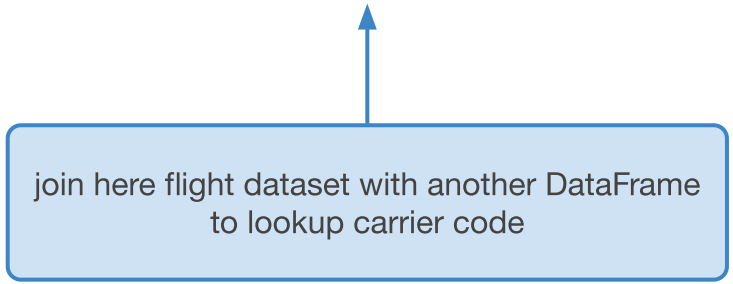
# Joining and Combining data

# 22.10 Joins

⇒ Similar to Joins in a database, we can also join two dataframes based on an Index or a column

```
df.join(df_carrier.set_index('Code'), on='OP_UNIQUE_CARRIER')
```

| | OP_UNIQUE_CARRIER | DEP_DELAY |
|---|---|---|
| 0 | UA | -13.0 |
| 1 | UA | -4.0 |
| 2 | UA | -2.0 |
| 3 | UA | -9.0 |
| 4 | UA | -14.0 |




| | Code | Description |
|---|---|---|
| 0 | 02Q | Titan Airways |
| 1 | 04Q | Tradewind Aviation |
| 2 | 05Q | Comlux Aviation, AG |
| 3 | 06Q | Master Top Linhas Aereas Ltd. |
| 4 | 07Q | Flair Airlines Ltd. |

Great, what now?

# 22.11 How you can use pandas... I/II

⇒ For reports/papers/...: Pandas has a function to produce a Latex table! No more manual typing of results into a Latex table...

→ `df_results.head().to_latex(index=False)`

| | CarrierName | DEP_DELAY |
|---|---|---|
| 9 | JetBlue Airways | 20.429078 |
| 7 | Frontier Airlines Inc. | 15.982878 |
| 13 | SkyWest Airlines Inc. | 15.123184 |
| 11 | PSA Airlines Inc. | 13.794702 |
| 6 | ExpressJet Airlines LLC | 13.642608 |

```
\begin{tabular}{lr}
\toprule
             CarrierName &  DEP\_DELAY \\
\midrule
         JetBlue Airways &  20.429078 \\
  Frontier Airlines Inc. &  15.982878 \\
   SkyWest Airlines Inc. &  15.123184 \\
       PSA Airlines Inc. &  13.794702 \\
 ExpressJet Airlines LLC &  13.642608 \\
\bottomrule
\end{tabular}
```

# 22.11 How can you use pandas... II/II

⇒ Pandas has also a function to produce a HTML table!

→ use this e.g. in your Flask app!

⇒ **Note:** Pandas is slow for large data, use a database instead or speed up your query using flask-cache (pythonhosted.org/Flask-Cache/).

```
@app.route('/')
def index():
    # load dataset, perform analytics
    df_results = ...

    return df_results.head().to_html(index=None)
```

| CarrierName | DEP_DELAY |
|---|---|
| JetBlue Airways | 20.429078 |
| Frontier Airlines Inc. | 15.982878 |
| SkyWest Airlines Inc. | 15.123184 |
| PSA Airlines Inc. | 13.794702 |
| ExpressJet Airlines LLC | 13.642608 |

# Visualizing data

# 22.12 Visualization resources

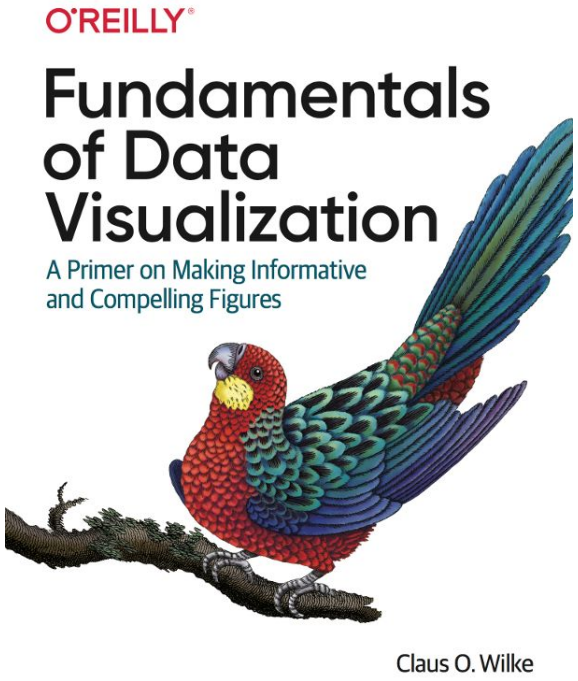
⇒ There are many popular libraries available to plot data (often in the form of DataFrames) like Matplotlib, Plot.ly, ggplot, Bokeh, pygal

A great resource for general visualization overview is

*Wilke, C. Fundamentals of data visualization : a primer on making informative and compelling figures. Sebastopol, CA: O'Reilly Media, Inc, 2019. Print.*

⇒ available freely online under serialmentor.com/dataviz/

⇒ Following slides have a sneak preview of matplotlib



# 22.13 Matplotlib

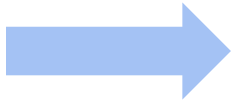
⇒ provides a state-based API to quickly create plots

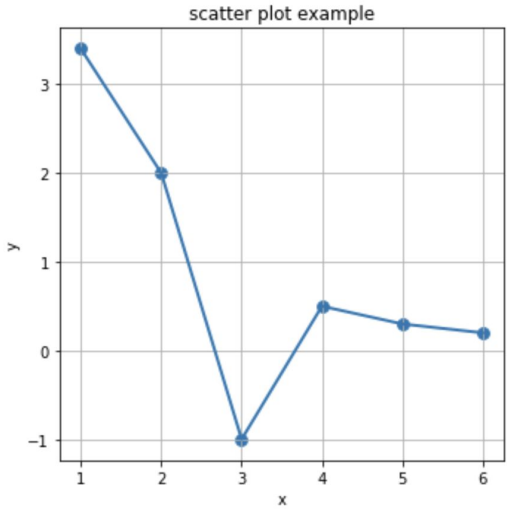
→ Tip: Use seaborn or pandas directly for quick plots!

→ Tip: Use seaborn to make plots more visually appealing

```
import matplotlib.pyplot as plt

plt.figure(figsize=(5, 5))
x = [1, 2, 3, 4, 5, 6]
y = [3.4, 2.0, -1, 0.5, .3, .2]
plt.grid()
plt.scatter(x, y, s=60)
plt.plot(x, y, lw=2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('scatter plot example')
plt.tight_layout()
plt.savefig('img.png', transparent=True)
```





# 22.14 Quick plots in pandas/seaborn

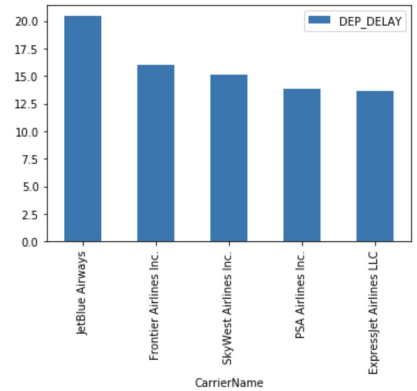
⇒ pandas has integrated functions to quickly plot out data via matplotlib

⇒ seaborn provides many convenience wrappers for high-level plots

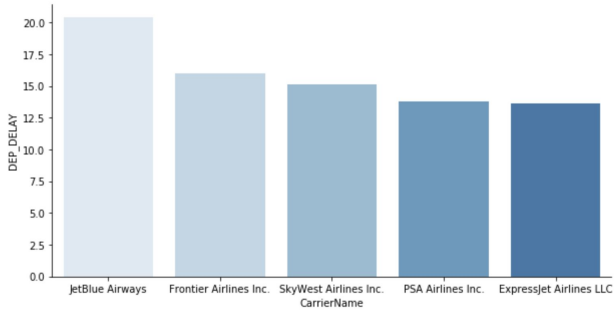
```
df.head()
```

| | CarrierName | DEP_DELAY |
|---|---|---|
| 0 | JetBlue Airways | 20.429078 |
| 1 | Frontier Airlines Inc. | 15.982878 |
| 2 | SkyWest Airlines Inc. | 15.123184 |
| 3 | PSA Airlines Inc. | 13.794702 |
| 4 | ExpressJet Airlines LLC | 13.642608 |

```
df.head().set_index('CarrierName') \
         .plot.bar()
```




```
import seaborn as sns
sns.barplot(x='CarrierName', y='DEP_DELAY',
            data=df.head(),
            palette=sns.color_palette('Blues'))
sns.despine()
```



# 22.15 How can we use this with Flask?

⇒ we can use matplotlib to render images on the server-side
→ i.e. return as SVG or PNG image

⇒ Can be slow, use `flask-cache` to cache requests!

⇒ For interactive graphics, better render data via a Javascript library like d3, chart.js, … or use Bokeh (http://biobits.org/bokeh-flask.html)

# End of lecture.

## Last class: Thu, 4pm @ CIT 477