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/brownecs6/FlaskExamples
21.99 Deployment via docker-composer

docker-compose.yml

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

```python
import numpy as np

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

a + b
```

\[ y_{it} = \beta'x_{it} + \mu_i + \epsilon_{it} \]
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

```python
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})
```
Pandas I/O
There are multiple ways to load data into a dataframe

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

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

pd.DataFrame(records)
```

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

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

```python
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

<table>
<thead>
<tr>
<th>target</th>
<th>commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>main-memory</td>
<td>df.to_dict(orient='records')</td>
</tr>
<tr>
<td></td>
<td># list of dicts</td>
</tr>
<tr>
<td></td>
<td>list(df.to_records(index=None))</td>
</tr>
<tr>
<td></td>
<td># list of tuples</td>
</tr>
<tr>
<td>CSV / JSON / Excel</td>
<td>df.to_csv('sample.csv', index=None)</td>
</tr>
<tr>
<td></td>
<td>df.to_json('sample.json',</td>
</tr>
<tr>
<td></td>
<td>orient='records',</td>
</tr>
<tr>
<td></td>
<td>lines=True)</td>
</tr>
<tr>
<td></td>
<td>df.to_excel('sample.xlsx', index=None)</td>
</tr>
<tr>
<td>Database (via SQLAlchemy)</td>
<td>df.to_sql('sample', db, index=None, if_exists='replace')</td>
</tr>
</tbody>
</table>
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
  \[ \text{df[label]} \]
  → to get the i-th column, use \[ \text{df[df.columns[i]]} \]

⇒ to get a subset (i.e. another dataframe) of columns, use
  \[ \text{df[[label1, label2, ..., labelN]]} \]
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 columns 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)
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>]`
- `df.loc[<row sel>, <col sel>]`

Note: To assign new values, use iloc or loc!
22.07 Indexing - Examples

<table>
<thead>
<tr>
<th>state</th>
<th>color</th>
<th>food</th>
<th>age</th>
<th>height</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>NY</td>
<td>Steak</td>
<td>30</td>
<td>165</td>
<td>4.6</td>
</tr>
<tr>
<td>Niko</td>
<td>TX</td>
<td>Lamb</td>
<td>2</td>
<td>70</td>
<td>8.3</td>
</tr>
<tr>
<td>Aaron</td>
<td>FL</td>
<td>Mango</td>
<td>12</td>
<td>120</td>
<td>9.0</td>
</tr>
<tr>
<td>Penelope</td>
<td>AL</td>
<td>Apple</td>
<td>4</td>
<td>80</td>
<td>3.3</td>
</tr>
<tr>
<td>Dean</td>
<td>AK</td>
<td>Cheese</td>
<td>32</td>
<td>180</td>
<td>1.8</td>
</tr>
<tr>
<td>Christina</td>
<td>TX</td>
<td>Melon</td>
<td>33</td>
<td>172</td>
<td>9.5</td>
</tr>
<tr>
<td>Cornelia</td>
<td>TX</td>
<td>Beans</td>
<td>69</td>
<td>150</td>
<td>2.2</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>state</th>
<th>color</th>
<th>food</th>
<th>age</th>
<th>height</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Niko</td>
<td>TX</td>
<td>green</td>
<td>Lamb</td>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>Penelope</td>
<td>AL</td>
<td>white</td>
<td>Apple</td>
<td>4</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>age</th>
<th>height</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>30</td>
<td>165</td>
</tr>
<tr>
<td>Niko</td>
<td>2</td>
<td>70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>food</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mango</td>
<td>FL</td>
</tr>
<tr>
<td>Apple</td>
<td>AL</td>
</tr>
<tr>
<td>Cheese</td>
<td>AK</td>
</tr>
<tr>
<td>Melon</td>
<td>TX</td>
</tr>
<tr>
<td>Beans</td>
<td>TX</td>
</tr>
</tbody>
</table>
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 `|`, indices must be of type Pandas Index/numpy arrays.
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')`
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()`

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>bar</td>
<td>one</td>
<td>4</td>
</tr>
<tr>
<td>foo</td>
<td>two</td>
<td>7</td>
</tr>
<tr>
<td>bar</td>
<td>three</td>
<td>2</td>
</tr>
<tr>
<td>foo</td>
<td>two</td>
<td>3</td>
</tr>
<tr>
<td>bar</td>
<td>two</td>
<td>1</td>
</tr>
<tr>
<td>foo</td>
<td>one</td>
<td>5</td>
</tr>
<tr>
<td>foo</td>
<td>three</td>
<td>5</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>bar</td>
<td>one</td>
<td>4</td>
</tr>
<tr>
<td>foo</td>
<td>two</td>
<td>7</td>
</tr>
<tr>
<td>bar</td>
<td>three</td>
<td>2</td>
</tr>
<tr>
<td>foo</td>
<td>two</td>
<td>3</td>
</tr>
<tr>
<td>bar</td>
<td>two</td>
<td>1</td>
</tr>
<tr>
<td>foo</td>
<td>one</td>
<td>5</td>
</tr>
<tr>
<td>foo</td>
<td>three</td>
<td>5</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>CarrierName</th>
<th>DEP_DELAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>9  JetBlue Airways</td>
<td>20.429078</td>
</tr>
<tr>
<td>7  Frontier Airlines Inc.</td>
<td>15.982878</td>
</tr>
<tr>
<td>13  SkyWest Airlines Inc.</td>
<td>15.123184</td>
</tr>
<tr>
<td>11  PSA Airlines Inc.</td>
<td>13.794702</td>
</tr>
<tr>
<td>6   ExpressJet Airlines LLC</td>
<td>13.642608</td>
</tr>
</tbody>
</table>

\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}
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/).

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

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


⇒ 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

```python
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

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

```
<table>
<thead>
<tr>
<th>CarrierName</th>
<th>DEP_DELAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>JetBlue Airways</td>
<td>20.429078</td>
</tr>
<tr>
<td>Frontier Airlines Inc.</td>
<td>15.982878</td>
</tr>
<tr>
<td>SkyWest Airlines Inc.</td>
<td>15.123184</td>
</tr>
<tr>
<td>PSA Airlines Inc.</td>
<td>13.794702</td>
</tr>
<tr>
<td>ExpressJet Airlines LLC</td>
<td>13.642608</td>
</tr>
</tbody>
</table>
```

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