Learning goals:

- Understand how state works within a notebook.
- Understand slicing DataFrames.
- Get hints for a bunch of questions on A2.

# More A2

---


**COGS 108 Fall 2019**

**Sam Lau**

**Discussion 5**

**bit.ly/sam-108-fa19**

**lau@ucsd.edu**

**OH: Wed 10-11a in SSRB 100**

**Why does my code sometimes break?**

**Keeping track of notebook state is very, very subtle!**

# What is df_income?

| | first_name | id | income | last_name |
|---|---|---|---|---|
| **0** | Lauren | 1592 | 23951.49 | Murphy |
| **1** | Rebecca | 27495 | 31019.37 | Walls |
| **2** | Alejandra | 19776 | 19058.09 | Garcia |
| **...** | ... | ... | ... | ... |
| **12662** | Mark | 58060 | 50696.11 | Torres |
| **12663** | Peter | 13881 | 0.00 | Gibson |
| **12664** | Michele | 35147 | 19864.48 | Robinson |

```
df_income.drop(['first_name', 'last_name'], axis=1)
```

| | id | income |
|---|---|---|
| **0** | 1592 | 23951.49 |
| **1** | 27495 | 31019.37 |
| **2** | 19776 | 19058.09 |
| **...** | ... | ... |
| **12662** | 58060 | 50696.11 |
| **12663** | 13881 | 0.00 |
| **12664** | 35147 | 19864.48 |

12665 rows × 2 columns

```
In [ ]: df_income
```

# What is df_income?

| | first_name | id | income | last_name |
|---|---|---|---|---|
| **0** | Lauren | 1592 | 23951.49 | Murphy |
| **1** | Rebecca | 27495 | 31019.37 | Walls |
| **2** | Alejandra | 19776 | 19058.09 | Garcia |
| **...** | ... | ... | ... | ... |
| **12662** | Mark | 58060 | 50696.11 | Torres |
| **12663** | Peter | 13881 | 0.00 | Gibson |
| **12664** | Michele | 35147 | 19864.48 | Robinson |

```
df_income = df_income.drop(['first_name', 'last_name'], axis=1)
```

```
In [ ]: df_income
```

**What happens if you run the first cell one time? Two times?**

# What is df_income?

| | first_name | id | income | last_name |
|---|---|---|---|---|
| **0** | Lauren | 1592 | 23951.49 | Murphy |
| **1** | Rebecca | 27495 | 31019.37 | Walls |
| **2** | Alejandra | 19776 | 19058.09 | Garcia |
| **...** | ... | ... | ... | ... |
| **12662** | Mark | 58060 | 50696.11 | Torres |
| **12663** | Peter | 13881 | 0.00 | Gibson |
| **12664** | Michele | 35147 | 19864.48 | Robinson |

```
df_income.drop(['first_name', 'last_name'], axis=1)
```

| | id | income |
|---|---|---|
| **0** | 1592 | 23951.49 |
| **1** | 27495 | 31019.37 |
| **2** | 19776 | 19058.09 |
| **...** | ... | ... |
| **12662** | 58060 | 50696.11 |
| **12663** | 13881 | 0.00 |
| **12664** | 35147 | 19864.48 |

12665 rows × 2 columns

```
In [ ]: df_income
```

**What happens if you run the first cell one time? Two times?**

# What is df_income?

| | first_name | id | income | last_name |
|---|---|---|---|---|
| **0** | Lauren | 1592 | 23951.49 | Murphy |
| **1** | Rebecca | 27495 | 31019.37 | Walls |
| **2** | Alejandra | 19776 | 19058.09 | Garcia |
| **...** | ... | ... | ... | ... |
| **12662** | Mark | 58060 | 50696.11 | Torres |
| **12663** | Peter | 13881 | 0.00 | Gibson |
| **12664** | Michele | 35147 | 19864.48 | Robinson |

```
df_income = df_income.drop(['first_name'], axis=1)
```

```
df_income = df_income.drop(['last_name'], axis=1)
```

In [ ]:
```
df_income
```

# What is df_income?

| | first_name | id | income | last_name |
|---|---|---|---|---|
| **0** | Lauren | 1592 | 23951.49 | Murphy |
| **1** | Rebecca | 27495 | 31019.37 | Walls |
| **2** | Alejandra | 19776 | 19058.09 | Garcia |
| **...** | ... | ... | ... | ... |
| **12662** | Mark | 58060 | 50696.11 | Torres |
| **12663** | Peter | 13881 | 0.00 | Gibson |
| **12664** | Michele | 35147 | 19864.48 | Robinson |

```
df_income = df_income.drop(['first_name'], axis=1)
```

**Edited to ->**

```
df_income = df_income.drop(['last_name'], axis=1)
```

```
In [ ]: df_income
```

**You will pass the local tests but fail the autograder! Be very careful when editing cells that mutate variables.**

# Okay, so I how do not screw things up?

- Avoid mutation until absolutely necessary!
  - Sam uses temporary variables to work around this.
- If a cell has code that results in mutation, only run it once.
  - If you need to run it again (e.g. because of a bug), run all cells above it first.
- Restart kernel and run all cells often, and especially before you turn in your assignment.

# What's the deal with brackets?

- **Why do I need brackets? When do I use parentheses and when do I use brackets?**
- **Why do I sometimes put strings in brackets but other times an expression?**
- **Why do I sometimes need double brackets??**

For more on this: http://bit.ly/sam-pandas-01

# Use brackets when taking slices (subsets) of a DF

Key idea: Only **one** value goes into the brackets.

| | Candidate | Party | % | Year | Result |
|---|---|---|---|---|---|
| 0 | Obama | Democratic | 52.9 | 2008 | win |
| 1 | McCain | Republican | 45.7 | 2008 | loss |
| 2 | Obama | Democratic | 51.1 | 2012 | win |
| 3 | Romney | Republican | 47.2 | 2012 | loss |
| 4 | Clinton | Democratic | 48.2 | 2016 | loss |
| 5 | Trump | Republican | 46.1 | 2016 | win |

## How do I grab a single column?

```
elections["Candidate"].head(6)
```

```
0      Reagan
1      Carter
2    Anderson
3      Reagan
4     Mondale
5        Bush
Name: Candidate, dtype: object
```

This is a Series!

## How do I grab multiple columns?

```
elections[["Candidate", "Party"]].head(6)
```

| | Candidate | Party |
|---|---|---|
| 0 | Reagan | Republican |
| 1 | Carter | Democratic |
| 2 | Anderson | Independent |
| 3 | Reagan | Republican |
| 4 | Mondale | Democratic |
| 5 | Bush | Republican |

This is a DF!

# Use brackets when taking slices (subsets) of a DF

| | Candidate | Party | % | Year | Result |
|---|---|---|---|---|---|
| 0 | Obama | Democratic | 52.9 | 2008 | win |
| 1 | McCain | Republican | 45.7 | 2008 | loss |
| 2 | Obama | Democratic | 51.1 | 2012 | win |
| 3 | Romney | Republican | 47.2 | 2012 | loss |
| 4 | Clinton | Democratic | 48.2 | 2016 | loss |
| 5 | Trump | Republican | 46.1 | 2016 | win |

## How do I grab rows?

```
elections[0:3]
```

| | Candidate | Party | % | Year | Result |
|---|---|---|---|---|---|
| 0 | Reagan | Republican | 50.7 | 1980 | win |
| 1 | Carter | Democratic | 41.0 | 1980 | loss |
| 2 | Anderson | Independent | 6.6 | 1980 | loss |

This is a DF!

```
elections[elections['Party'] == 'Independent']
```

| | Candidate | Party | % | Year | Result |
|---|---|---|---|---|---|
| 2 | Anderson | Independent | 6.6 | 1980 | loss |
| 9 | Perot | Independent | 18.9 | 1992 | loss |
| 12 | Perot | Independent | 8.4 | 1996 | loss |

Whoa, what's going on here?

# Demo with Elections Data

**bit.ly/108-sam05**

**Full video walkthrough available on my discussion GitHub page (see Sam's pandas lecture video).**

# Bracket Takeaways?

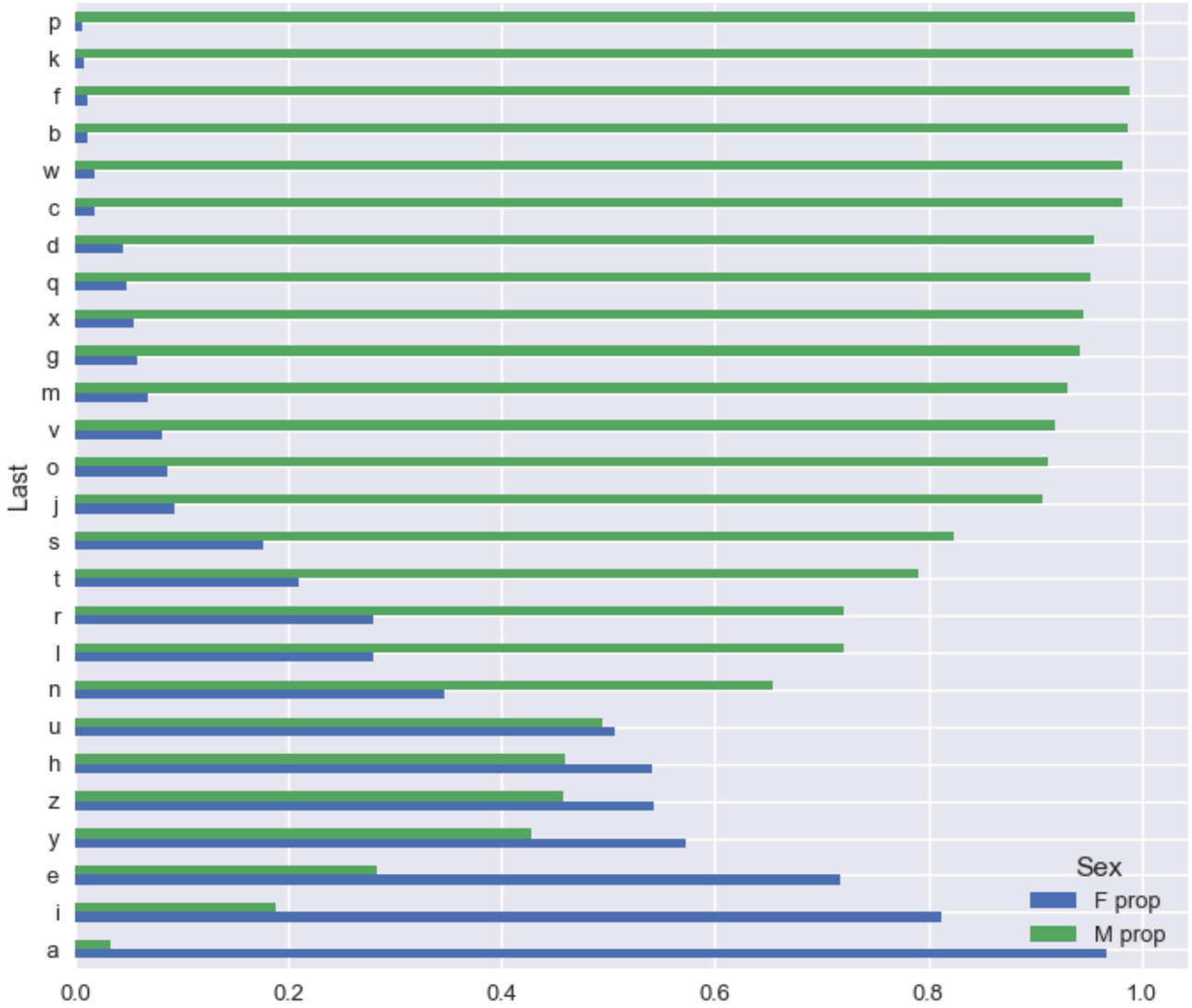
- **Brackets = slicing a DF.**
  **Parentheses = calculating something about a DF.**
- **Strings in brackets = grabbing column (Series)**
  **List of strings in brackets = grabbing columns (DF)**
- **Slice in brackets = grabbing rows (DF)**
  **Boolean expression in brackets = grabbing rows (DF)**
  **(You will need this last one for question 4b.)**

# Preview of next week

**Slicing: how do I filter my Data Frame?**

**String methods: how do I work with text?**

**Turns out that the last letter of a person's first name is a good predictor of sex!**



# A2 quick tips

- **Answer to 1b located in discussion notebook. Just copy it.**
- **You can leave 1e blank if your columns are already in the right order.**
- **Use Series.isnull() for 2a.**
  **(Series.isna() is only available in newer pandas versions.)**
- **Use plt.hist() for Part 3. Ignore warnings for 3d.**
- **Use boolean slicing for 4b, 4f, and 5e.**
- **Use np.log10() for 4d, not np.log()**
- **For question 6i, the better predictor is the one with the most non-zero correlation.**