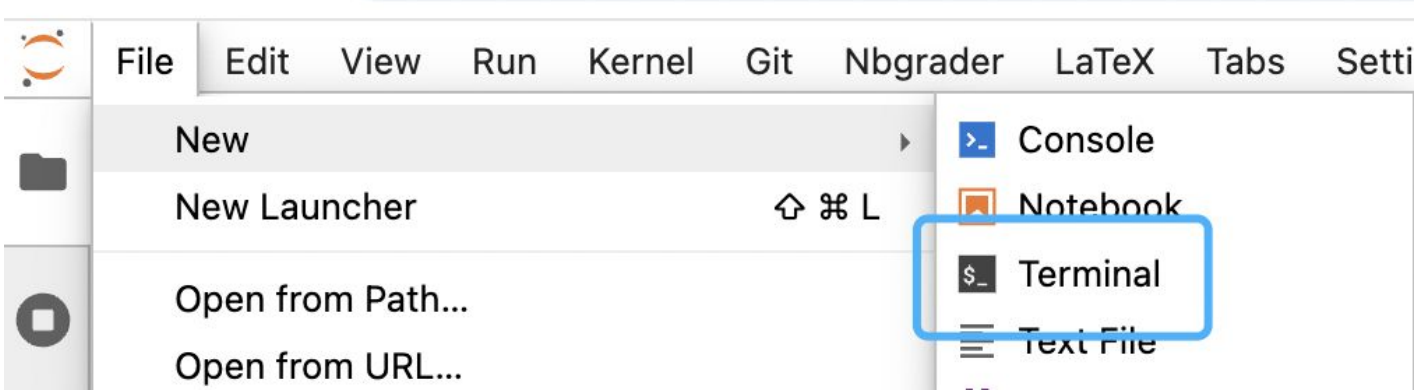
Week 3

COGS 108 Sp 2025

Due dates

- Q2: Monday (04/14)
- A1: Wednesday (04/16)
- Group Signup: Wednesday (04/16)
- D2: Friday (04/18)

Using Git in Datahub terminal



Using Git in Datahub terminal

```
z5ying@dsmlp-jupyter-z5ying:~$ git clone https://github.com/COGS108/Lectures-Sp25.g
it
Cloning into 'Lectures-Sp25'...
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 46 (delta 1), reused 13 (delta 1), pack-reused 32 (from 1)
Receiving objects: 100% (46/46), 69.81 MiB | 13.05 MiB/s, done.
Resolving deltas: 100% (12/12), done.
```

```
z5ying@dsmlp-jupyter-z5ying:~/Lectures-Sp25/Lectures-Sp25$ git commit -m "added sec
tion slides"
[main 91def82] added section slides
 11 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 section-slides/D1 SP25.pdf
 create mode 100644 section-slides/D2 SP25.pdf
 create mode 100644 section-slides/D3 SP25.pdf
```

```
z5ying@dsmlp-jupyter-z5ying:~/Lectures-Sp25/Lectures-Sp25$ git push
Username for 'https://github.com': z5ying
Password for 'https://z5ying@github.com':
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 256 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 6.66 MiB | 8.46 MiB/s, done.
Total 15 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/z5ying/Lectures-Sp25.git
   e7010d4..91def82  main -> main
```

Common errors

```
z5ying@dsmlp-jupyter-z5ying:~/Lectures-Sp25$ git commit -m "added section slides"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'z5ying@dsmlp-jupyter-z5ying.(none)
')
z5ying@dsmlp-jupyter-z5ying:~/Lectures-Sp25$ git config --global user.email "z5ying
@ucsd.edu"
```

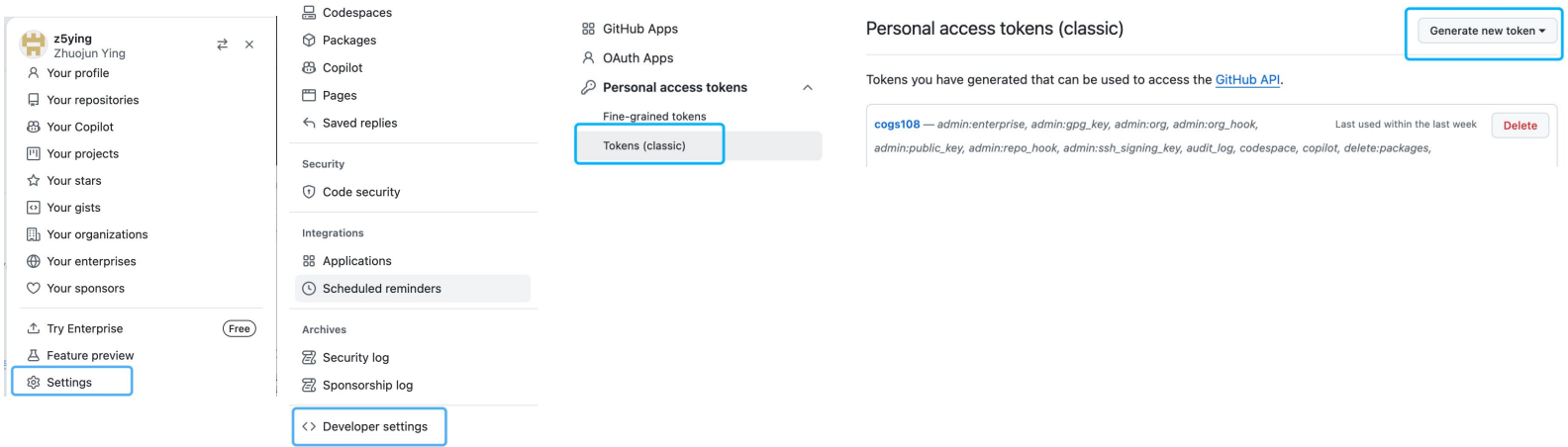
Solution

```
z5ying@dsmlp-jupyter-z5ying:~/Lectures-Sp25$ git config --global user.email "z5ying
@ucsd.edu"
```

Common errors

```
z5ying@dsmlp-jupyter-z5ying:~/Lectures-Sp25/Lectures-Sp25$ git push
Username for 'https://github.com': z5ying
Password for 'https://z5ying@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/abo
ut-remote-repositories#cloning-with-https-urls for information on currently recomme
nded modes of authentication.
fatal: Authentication failed for 'https://github.com/z5ying/Lectures-Sp25.git/'
```

Solution

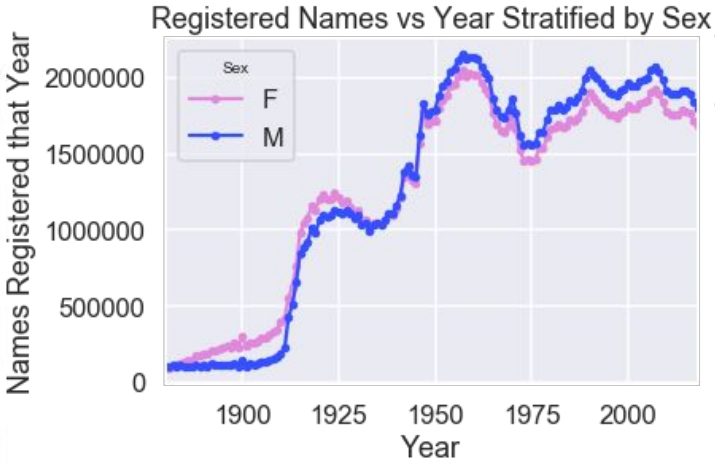


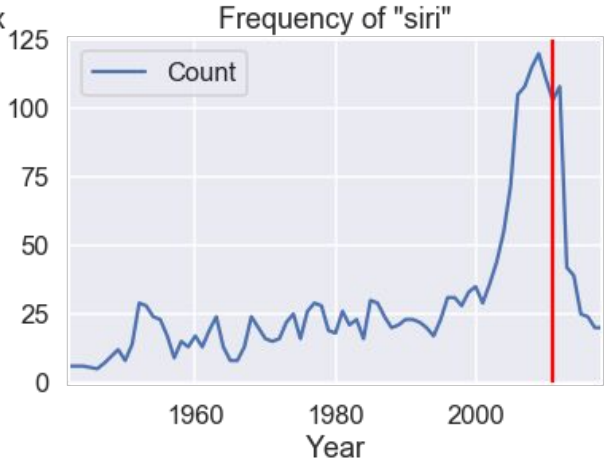
Welcome to the wonderful world of pandas!

Pandas is really useful!

	Name	Sex	Count	Year
0	Mary	F	7065	1880
1	Anna	F	2604	1880
2	Emma	F	2003	1880
...	...	...	...	...
1957043	Zyrie	M	5	2018
1957044	Zyron	M	5	2018
1957045	Zzyzx	M	5	2018

1957046 rows × 4 columns





It converts python into a usable (and good!) data analysis tool

Pandas has terrible error messages

	Timestamp	Name	Sex	Age
0	10/15/2019 21:49:38	samuel	M	24
1	10/16/2019 9:07:31	aditi	F	22
2	10/16/2019 9:07:34	hanyang	M	21
...	...	...	...	...
24	10/16/2019 16:08:45	amy	F	20
25	10/16/2019 16:08:46	sheila	F	21
26	10/16/2019 16:09:15	thomas	M	23

```
students['name']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2656             try:
-> 2657                 return self._engine.get_loc(key)
   2658             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'name'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-27-ae454297f350> in <module>()
----> 1 students['name']

~/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in __getitem__(self, key)
   2925             if self.columns.nlevels > 1:
   2926                 return self._getitem_multilevel(key)
-> 2927             indexer = self.columns.get_loc(key)
   2928             if is_integer(indexer):
   2929                 indexer = [indexer]

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2657                 return self._engine.get_loc(key)
   2658             except KeyError:
-> 2659                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2660         indexer = self.get_indexer([key], method=method, tolerance=tolerance)
   2661         if indexer.ndim > 1 or indexer.size > 1:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'name'
```

Pandas has unfriendly documentation

`DataFrame.rename`(*self*, *mapper=None*, *index=None*, *columns=None*, *axis=None*, *copy=True*, *inplace=False*, *level=None*, *errors='ignore'*) [source]

Alter axes labels.

Function / dict values must be unique (1-to-1). Labels not contained in a dict / Series will be left as-is. Extra labels listed don't throw an error.

See the user guide for more.

Parameters:

mapper : *dict-like or function*
Dict-like or functions transformations to apply to that axis' values. Use either `mapper` and `axis` to specify the axis to target with `mapper`, or `index` and `columns`.

index : *dict-like or function*
Alternative to specifying axis (`mapper, axis=0` is equivalent to `index=mapper`).

columns : *dict-like or function*
Alternative to specifying axis (`mapper, axis=1` is equivalent to `columns=mapper`).

axis : *int or str*
Axis to target with `mapper`. Can be either the axis name ('index', 'columns') or number (0, 1). The default is 'index'.

copy : *bool, default True*
Also copy underlying data.

inplace : *bool, default False*
Whether to return a new DataFrame. If True then value of copy is ignored.

level : *int or level name, default None*
In case of a MultiIndex, only rename labels in the specified level.

errors : *{'ignore', 'raise'}, default 'ignore'*
If 'raise', raise a *KeyError* when a dict-like *mapper*, *index*, or *columns* contains labels that are not present in the Index being transformed. If 'ignore', existing keys will be renamed and extra keys will be ignored.

Returns:

DataFrame
DataFrame with the renamed axis labels.

Raises:

KeyError
If any of the labels is not found in the selected axis and "errors='raise'".

Also, there are typically many ways to do the same thing in pandas.

3 skills that will save you 5+ hours on A2:

- Knowing the difference between a pandas Series and DataFrame.
- Knowing how to use Google effectively.
- Knowing how to read the pandas documentation.

What's a DataFrame?

Data Frame: two-dimensional table of data.

All columns are the same type (but not rows).

Every row and every column has a label.

We call the set of row labels the Index of a DataFrame

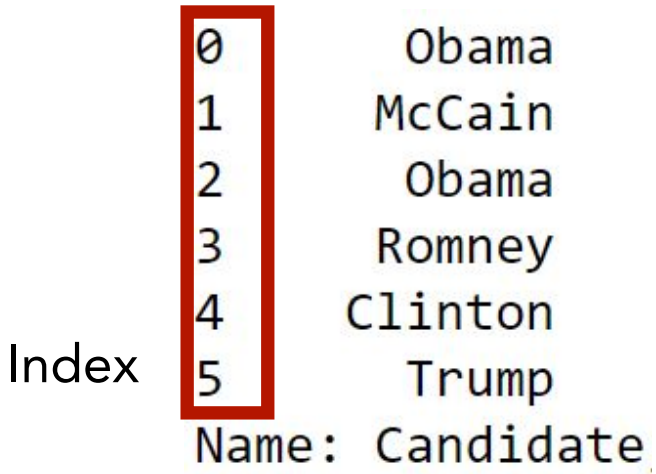
Year	Candidate	Party	%	Result
2008	Obama	Democratic	52.9	win
2008	McCain	Republican	45.7	loss
2012	Obama	Democratic	51.1	win
2012	Romney	Republican	47.2	loss
2016	Clinton	Democratic	48.2	loss
2016	Trump	Republican	46.1	win

Index

What's a Series?

Series: one-dimensional sequence of data.

Usually created by taking a single column from a Data Frame.



Why is this important?

Most pandas methods work differently between Data Frames and Series.

The documentation will tell you what type of object the method is for.

pandas.DataFrame.sort_values¶

DataFrame.**sort_values**(*self*, *by*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na_position='last'*) [source]

Sort by the values along either axis.

by : *str or list of str*

Name or list of names to sort by.

- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels

Changed in version 0.23.0: Allow specifying index or column level names.

pandas.Series.sort_values¶

Series.**sort_values**(*self*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na_position='last'*) [source]

Sort by the values.

Sort a Series in ascending or descending order by some criterion.

Parameters:

axis : *{0 or 'index'}, default 0*
Axis to direct sorting. The value 'index' is accepted for compatibility with DataFrame.sort_values.

ascending : *bool, default True*
If True, sort values in ascending order, otherwise descending.

inplace : *bool, default False*
If True, perform operation in-place.

kind : *{'quicksort', 'mergesort' or 'heapsort'}, default 'quicksort'*
Choice of sorting algorithm. See also `numpy.sort()` for more information. 'mergesort' is the only stable algorithm.

na_position : *{'first' or 'last'}, default 'last'*
Argument 'first' puts NaNs at the beginning, 'last' puts NaNs at the end.

Why is this important?

df.sort_values(…)

df[‘names’].sort_values(…)

pd.read_csv(…)

pandas.DataFrame.sort_values¶

DataFrame.**sort_values**(*self*, *by*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na_position='last'*) [source]

Sort by the values along either axis.

by : *str or list of str*

Name or list of names to sort by.

- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels

Changed in version 0.23.0: Allow specifying index or column level names.

pandas.Series.sort_values¶

Series.**sort_values**(*self*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na_position='last'*) [source]

Sort by the values.

Sort a Series in ascending or descending order by some criterion.

pandas.read_csv

pandas.**read_csv**(*filepath_or_buffer*, *sep=', '*, *delimiter=None*, *header='infer'*, *names=None*, *index_col=None*, *usecols=None*, *squeeze=False*, *prefix=None*, *mangle_dupe_cols=True*, *dtype=None*, *engine=None*, *converters=None*, *true_values=None*, *false_values=None*, *skipinitialspace=False*, *skiprows=None*, *nrows=None*, *na_values=None*, *keep_default_na=True*, *na_filter=True*, *verbose=False*, *skip_blank_lines=True*, *parse_dates=False*, *infer_datetime_format=False*, *keep_date_col=False*, *date_parser=None*, *dayfirst=False*, *iterator=False*, *chunksize=None*, *compression='infer'*, *thousands=None*, *decimal=b'.'*, *lineterminator=None*, *quotechar='"'*, *quoting=0*, *escapechar=None*, *comment=None*, *encoding=None*, *dialect=None*, *tupleize_cols=None*, *error_bad_lines=True*, *warn_bad_lines=True*, *skipfooter=0*, *doublequote=True*, *delim_whitespace=False*, *low_memory=True*, *memory_map=False*, *float_precision=None*) [source]

Read CSV (comma-separated) file into DataFrame

How to use Google properly

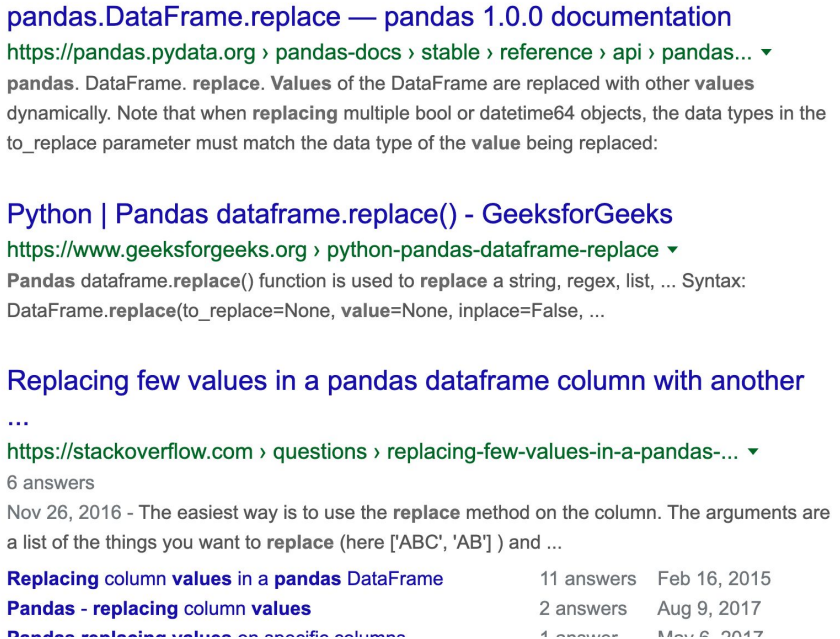
State your task:
“I need to replace 0 with False and 1 with True.”

Remove question-specific details:
“replace values”

Add the package name to the front:
“pandas replace values”

If you already know the right method, just google “pandas replace”

Cheat sheets can help you find the right method



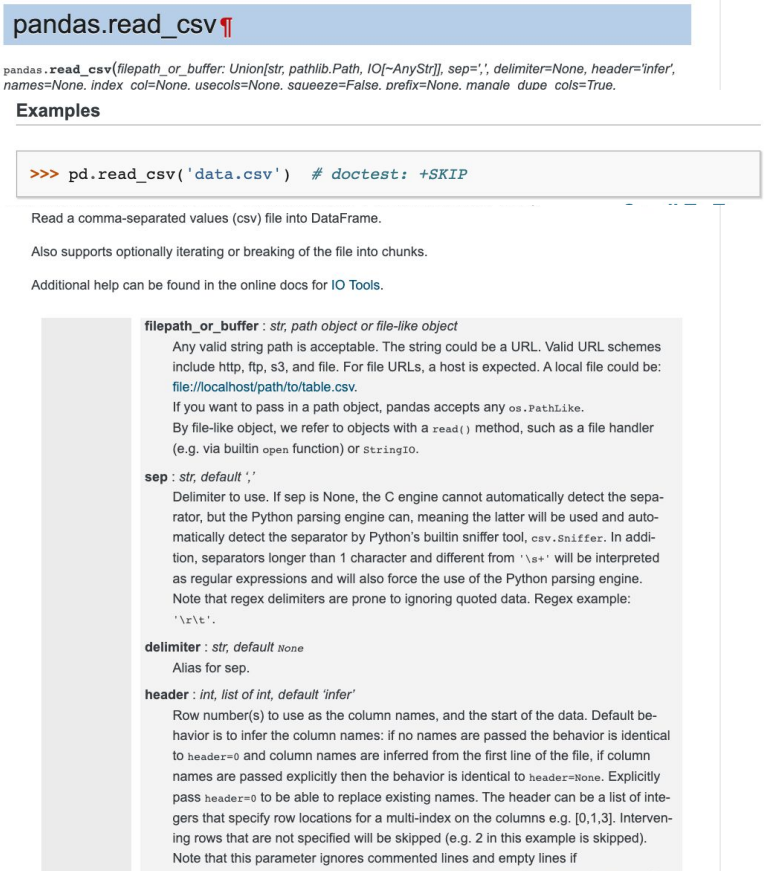
How to read pandas documentation

Skip the table of method parameters and look at the examples.

Copy example, then modify it to work for your notebook.

If needed, refer back to the method parameters for fine-tuning.

(The method in the picture on the right solves Q2.)



pandas.read_csv¶

pandas.**read_csv**(*filepath_or_buffer: Union[str, pathlib.Path, IO[~AnyStr]], sep=',', delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True,*

Examples

```
>>> pd.read_csv('data.csv')  # doctest: +SKIP
```

Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for IO Tools.

filepath_or_buffer : *str, path object or file-like object*
Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.csv.
If you want to pass in a path object, pandas accepts any `os.PathLike`.
By file-like object, we refer to objects with a `read()` method, such as a file handler (e.g. via builtin `open` function) or `StringIO`.

sep : *str, default ','*
Delimiter to use. If sep is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator by Python's builtin sniffer tool, `csv.Sniffer`. In addition, separators longer than 1 character and different from `'\s+'` will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: `'\r\t'`.

delimiter : *str, default None*
Alias for sep.

header : *int, list of int, default 'infer'*
Row number(s) to use as the column names, and the start of the data. Default behavior is to infer the column names: if no names are passed the behavior is identical to `header=0` and column names are inferred from the first line of the file, if column names are passed explicitly then the behavior is identical to `header=None`. Explicitly pass `header=0` to be able to replace existing names. The header can be a list of integers that specify row locations for a multi-index on the columns e.g. [0,1,3]. Intervening rows that are not specified will be skipped (e.g. 2 in this example is skipped). Note that this parameter ignores commented lines and empty lines if

Finally: don’t use loops

If you find yourself trying to write a for/while loop when working with pandas, you’re almost definitely doing it wrong.

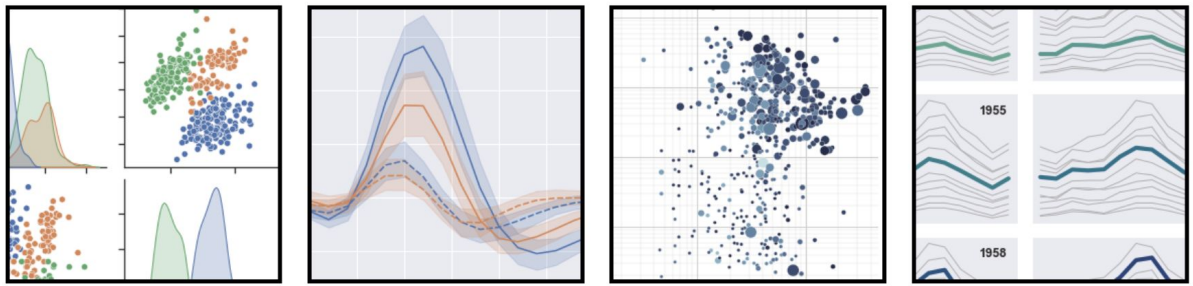
Look for the right pandas method. And ask your friend + staff for help.

Your time to …

- Talk to your classmates to find potential teammates!
- Work on D2 and A1

Pyplot and seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Alias : sns



Pandas Series and Dataframes

Python3

```python
#importing pandas library
import pandas as pd

#Creating a list
author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
#Creating a Series by passing list variable to Series() function
auth_series = pd.Series(author)
#Printing Series
print(auth_series)
```

Output:

```
0    Jitender
1     Purnima
2       Arpit
3       Jyoti
dtype: object
```

Pandas Series and Dataframes

We have created two lists ‘author’ and article’ which have been passed to Series() functions to create two Series.

After creating Series, we have created a dictionary and passed Series objects as values of the dictionary and keys of the dictionary will be served as Columns of the dataframe.

Python3

```
#Importing Pandas library
import pandas as pd

#Creating two lists
author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
article = [210, 211, 114, 178]
#Creating two Series by passing lists
auth_series = pd.Series(author)
article_series = pd.Series(article)
#Creating a dictionary by passing Series objects as values
frame = { 'Author': auth_series, 'Article': article_series }
#Creating DataFrame by passing Dictionary
result = pd.DataFrame(frame)
#Printing elements of Dataframe
print(result)
```

Output:

```
     Author  Article
0  Jitender      210
1   Purnima      211
2     Arpit      114
3     Jyoti      178
```

Part I : Cheating

```
feature_counts =
dataFrame['feature'].value_counts()
```

`df['your_column'].value_counts()` - this will return the count of unique occurences in the specified column.

It is important to note that `value_counts` only works on pandas series, not Pandas dataframes. As a result, we only include one bracket df['your_column'] and not two brackets df[['your_column']].

Parameters

- **normalize (bool, default False) -** If True then the object returned will contain the relative frequencies of the unique values.
- **sort (bool, default True) -** Sort by frequencies.
- **ascending (bool, default False) -** Sort in ascending order.
- **bins (int, optional) -** Rather than count values, group them into half-open bins, a convenience for `pd.cut`, only works with numeric data.
- **dropna (bool, default True) -**Don't include counts of NaN.

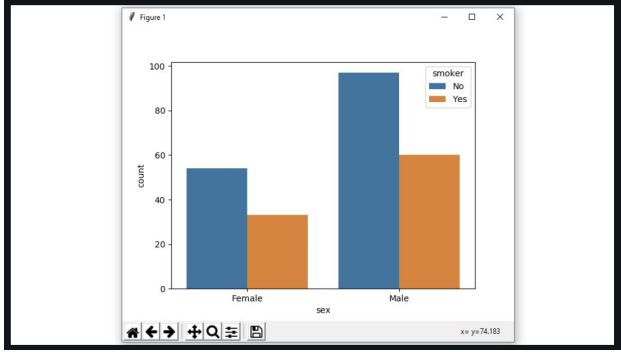
Part I : Cheating

sns.countplot(x, y, hue, data=df);

Python3

```
# importing the required library

import seaborn as sns
import matplotlib.pyplot as plt

# read a tips.csv file from seaborn library
df = sns.load_dataset('tips')

# count plot on two categorical variable
sns.countplot(x ='sex', hue = "smoker", data = df)

# Show the plot
plt.show()
```

Output :



Part I : Cheating

create a DataFrame prop_df with three columns, one for gender, one for cheated, and one including the proportion of respondents who cheated within each gender

```
prop_df = (survey['cheated']
        .groupby(...)
        .value_counts(normalize=True)
        .rename(...)
        .reset_index())
```

Part I : Cheating

Regenerate your barplot using the proportion data you just generated to determine which gender cheats more frequently.

Assign your seaborn plot to a variable named plot_proportion

```
plot_proportion = sns.barplot(x=' ',
        y=' ',
        hue=' ',
        data=dataFrame);
```

Y axis is cheated
Y axis is proportion
The hue is the gender

Swapping: include
hue_order=["Male","Female"],