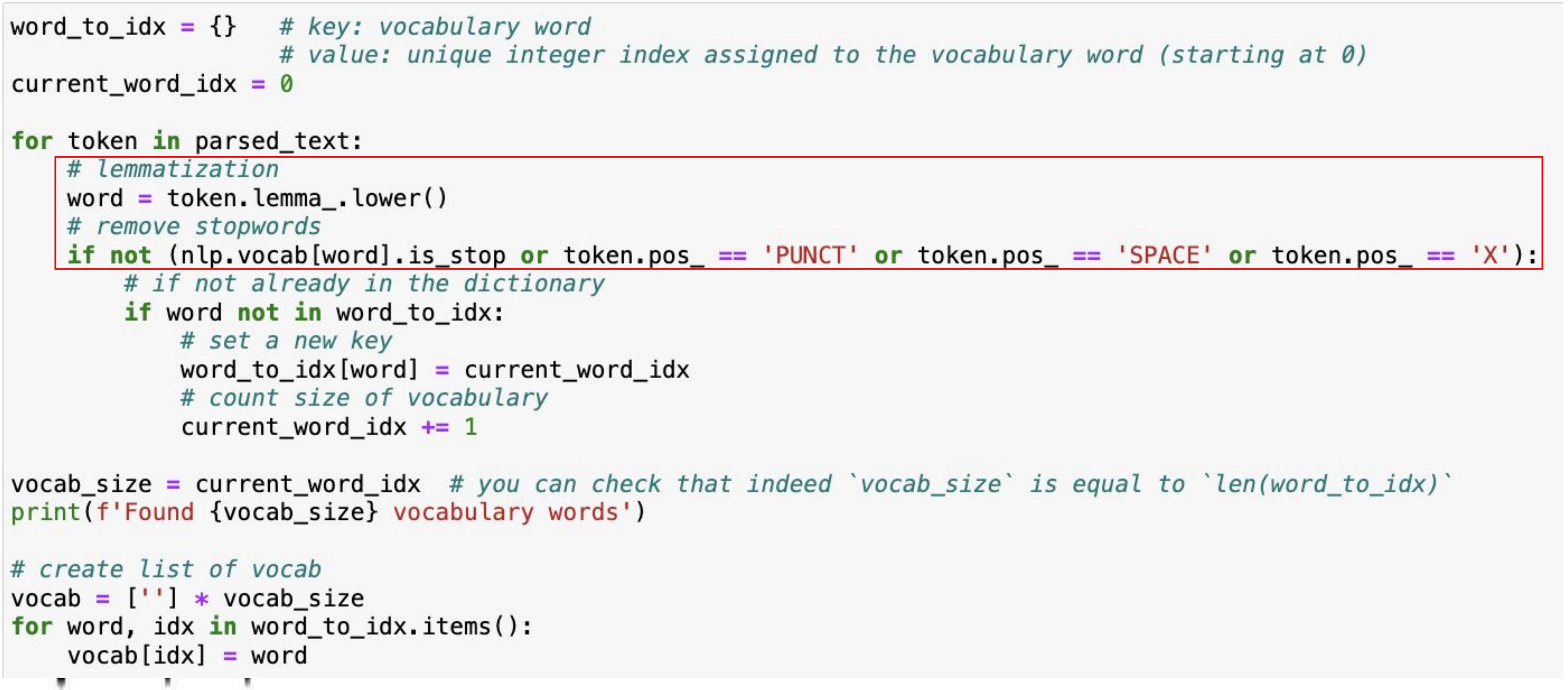
UDA Recitation

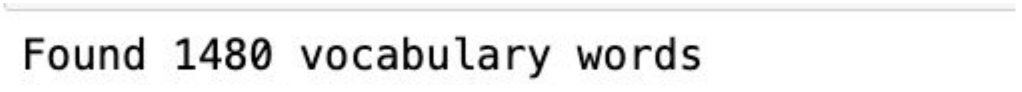
March 22 2024

1. Basic text analysis

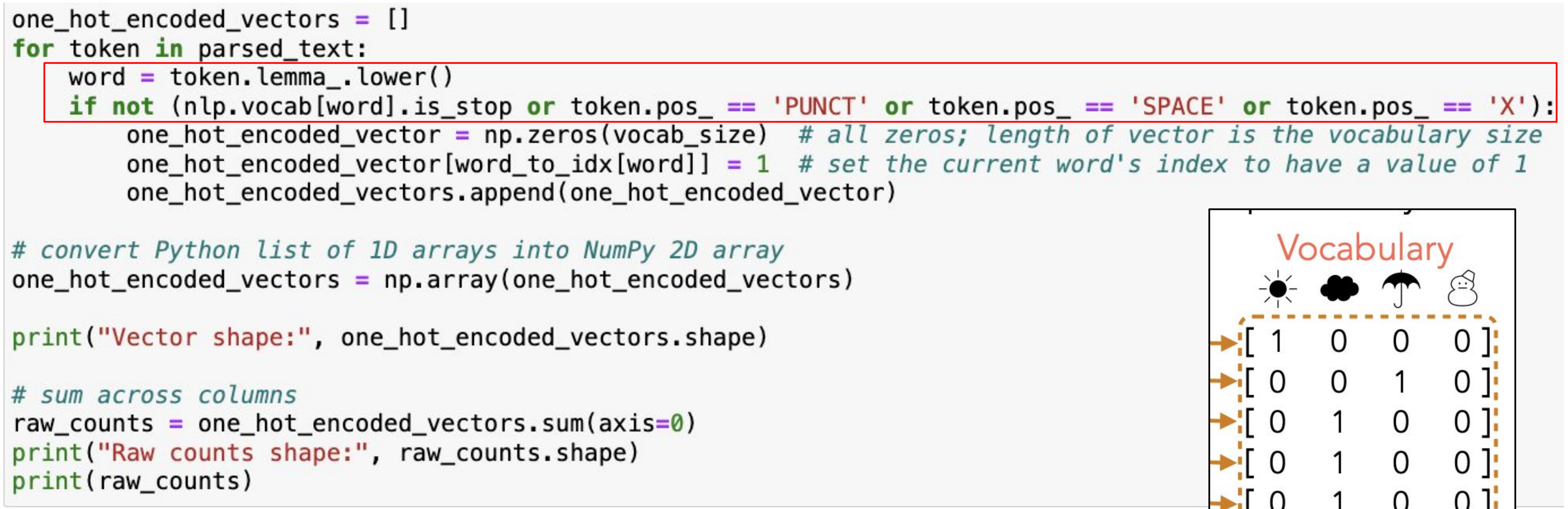
The code below re-creates the histogram plot from the previous demo (that uses lemmatization and stop words) using NumPy arrays



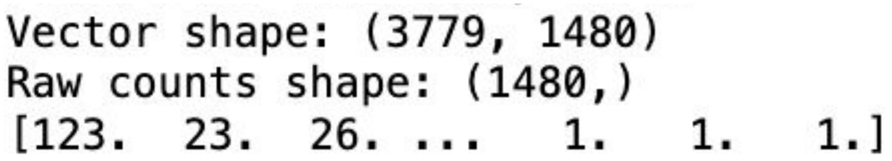
```python
word_to_idx = {}   # key: vocabulary word
                   # value: unique integer index assigned to the vocabulary word (starting at 0)
current_word_idx = 0

for token in parsed_text:
    # lemmatization
    word = token.lemma_.lower()
    # remove stopwords
    if not (nlp.vocab[word].is_stop or token.pos_ == 'PUNCT' or token.pos_ == 'SPACE' or token.pos_ == 'X'):
        # if not already in the dictionary
        if word not in word_to_idx:
            # set a new key
            word_to_idx[word] = current_word_idx
            # count size of vocabulary
            current_word_idx += 1

vocab_size = current_word_idx  # you can check that indeed `vocab_size` is equal to `len(word_to_idx)`
print(f'Found {vocab_size} vocabulary words')

# create list of vocab
vocab = [''] * vocab_size
for word, idx in word_to_idx.items():
    vocab[idx] = word
```



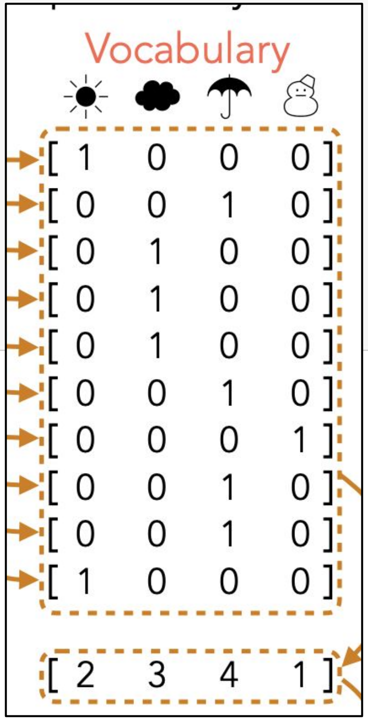
```
Found 1480 vocabulary words
```

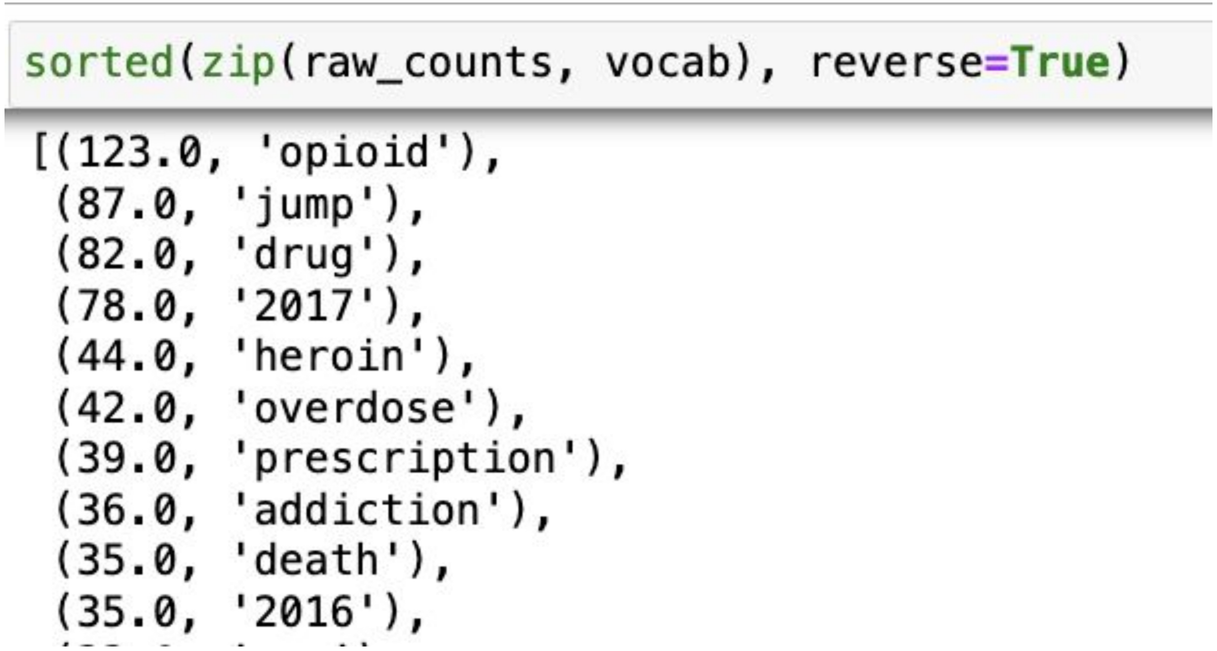


```
one_hot_encoded_vectors = []
for token in parsed_text:
    word = token.lemma_.lower()
    if not (nlp.vocab[word].is_stop or token.pos_ == 'PUNCT' or token.pos_ == 'SPACE' or token.pos_ == 'X'):
        one_hot_encoded_vector = np.zeros(vocab_size)  # all zeros; length of vector is the vocabulary size
        one_hot_encoded_vector[word_to_idx[word]] = 1  # set the current word's index to have a value of 1
        one_hot_encoded_vectors.append(one_hot_encoded_vector)

# convert Python list of 1D arrays into NumPy 2D array
one_hot_encoded_vectors = np.array(one_hot_encoded_vectors)

print("Vector shape:", one_hot_encoded_vectors.shape)

# sum across columns
raw_counts = one_hot_encoded_vectors.sum(axis=0)
print("Raw counts shape:", raw_counts.shape)
print(raw_counts)
```



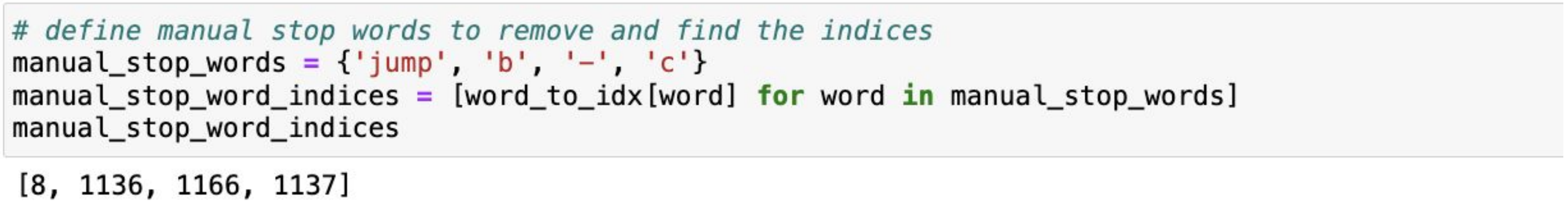
```
Vector shape: (3779, 1480)
Raw counts shape: (1480,)
[123.  23.  26. ...   1.   1.   1.]
```





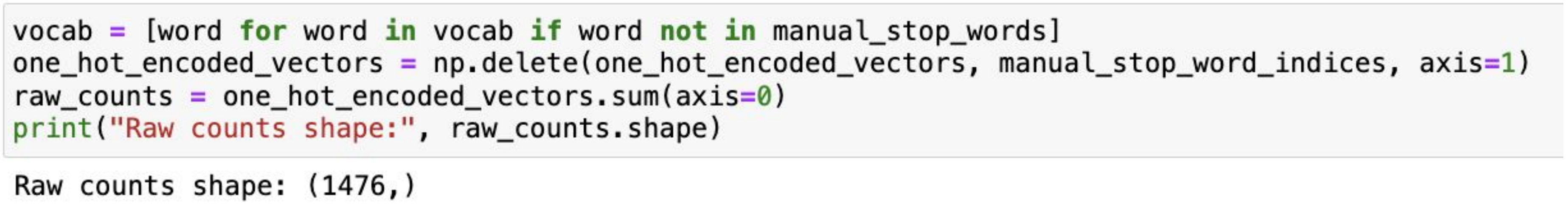
```
sorted(zip(raw_counts, vocab), reverse=True)
```

```
[(123.0, 'opioid'),
 (87.0, 'jump'),
 (82.0, 'drug'),
 (78.0, '2017'),
 (44.0, 'heroin'),
 (42.0, 'overdose'),
 (39.0, 'prescription'),
 (36.0, 'addiction'),
 (35.0, 'death'),
 (35.0, '2016'),
```



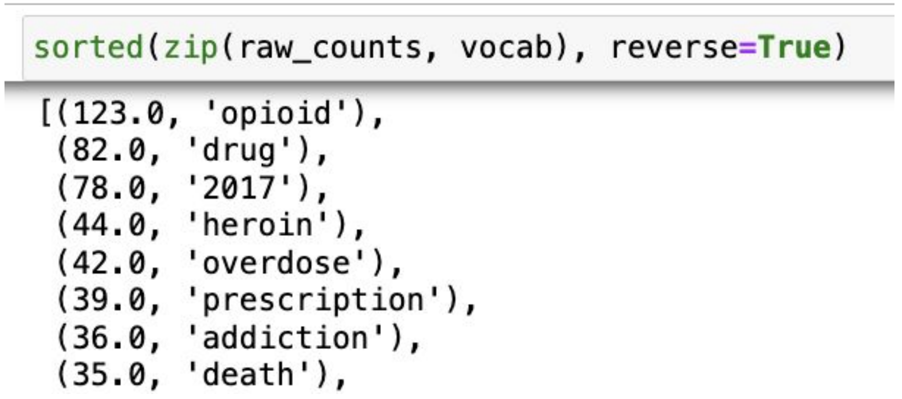
```python
# define manual stop words to remove and find the indices
manual_stop_words = {'jump', 'b', '-', 'c'}
manual_stop_word_indices = [word_to_idx[word] for word in manual_stop_words]
manual_stop_word_indices
```

```
[8, 1136, 1166, 1137]
```



```python
vocab = [word for word in vocab if word not in manual_stop_words]
one_hot_encoded_vectors = np.delete(one_hot_encoded_vectors, manual_stop_word_indices, axis=1)
raw_counts = one_hot_encoded_vectors.sum(axis=0)
print("Raw counts shape:", raw_counts.shape)
```

```
Raw counts shape: (1476,)
```

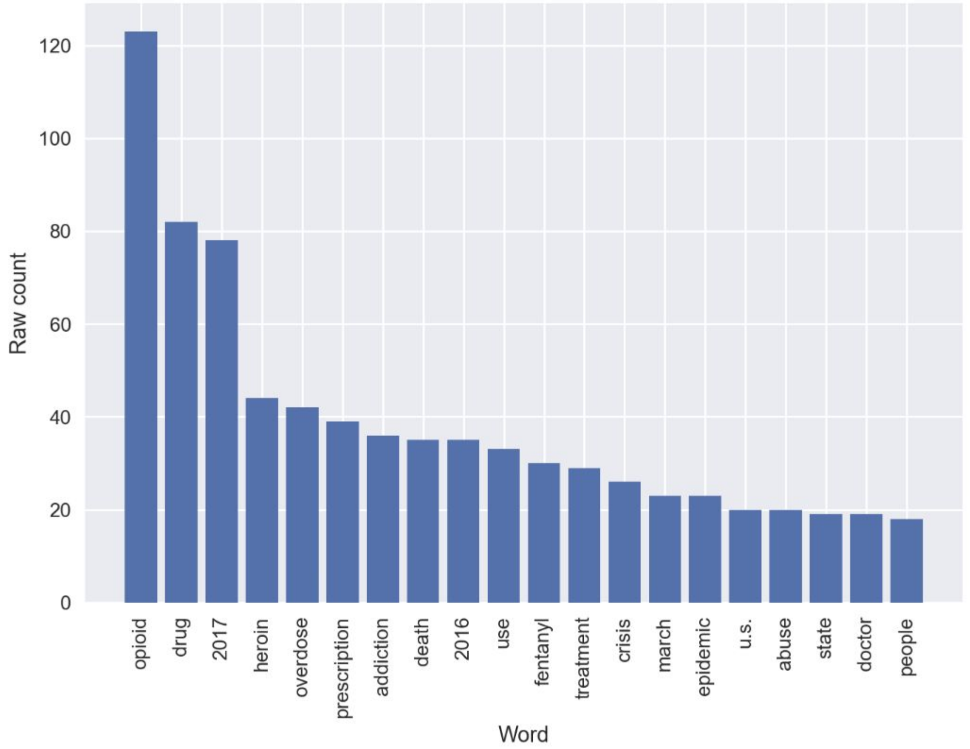


```python
sorted(zip(raw_counts, vocab), reverse=True)
```

```
[(123.0, 'opioid'),
 (82.0, 'drug'),
 (78.0, '2017'),
 (44.0, 'heroin'),
 (42.0, 'overdose'),
 (39.0, 'prescription'),
 (36.0, 'addiction'),
 (35.0, 'death'),
top_counts, top_words = zip(*sorted(zip(raw_counts, vocab), reverse=True))
```

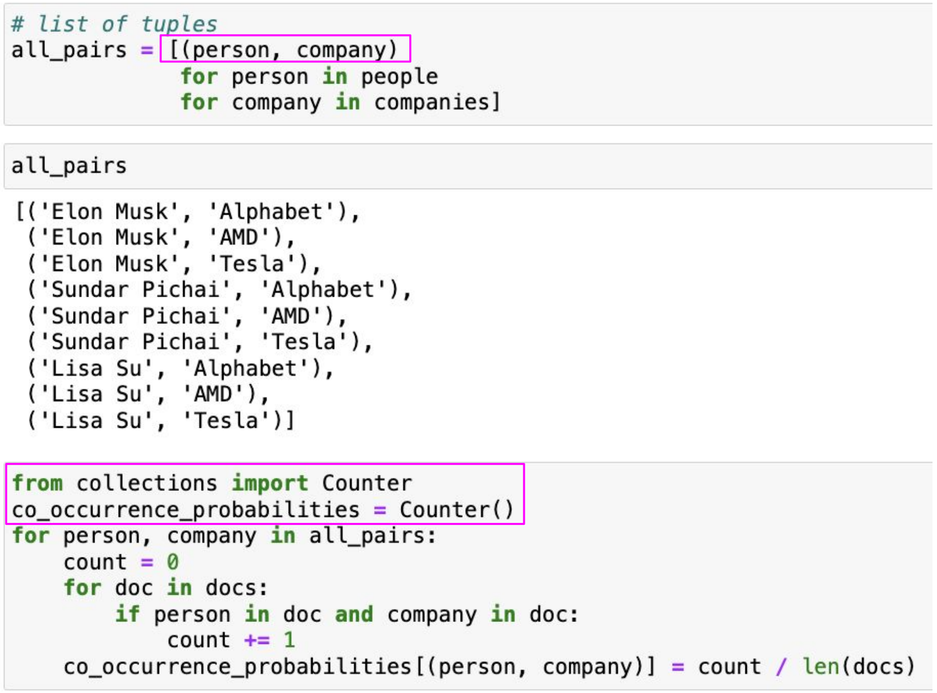
```
num_top_words_to_plot = 20
plt.bar(range(num_top_words_to_plot), top_counts[:num_top_words_to_plot])
plt.xticks(range(num_top_words_to_plot), top_words[:num_top_words_to_plot], rotation=90)
plt.xlabel('Word')
plt.ylabel('Raw count')
```

```
Text(0, 0.5, 'Raw count')
```



2. Co-occurrence analysis (using np arrays)

Using Counter



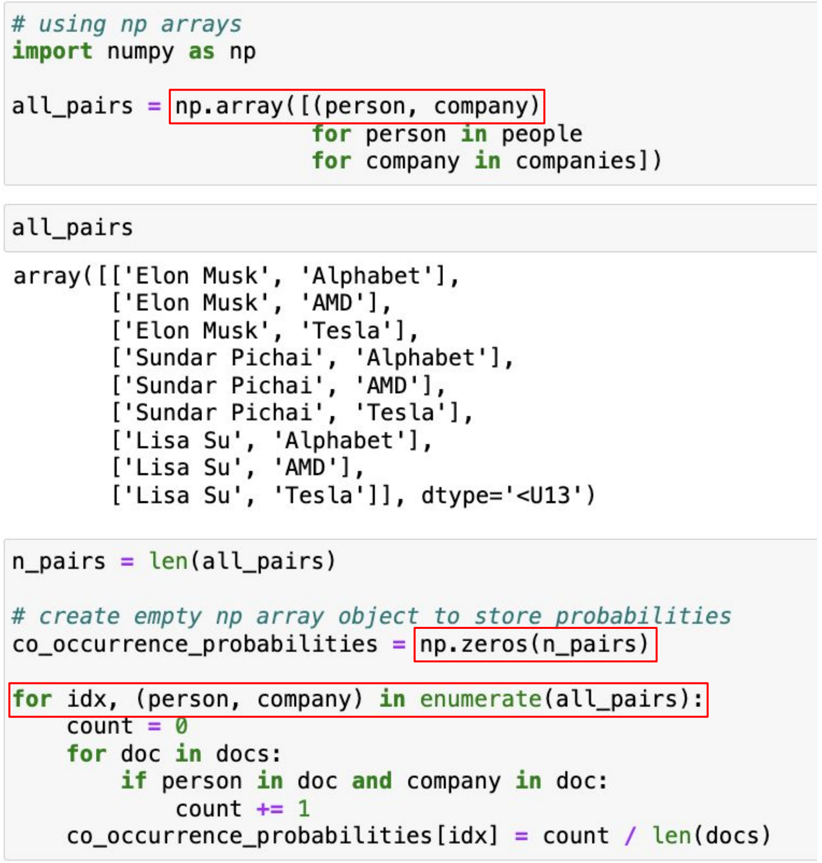
```
# list of tuples
all_pairs = [(person, company)
             for person in people
             for company in companies]
```

```
all_pairs
```

```
[('Elon Musk', 'Alphabet'),
 ('Elon Musk', 'AMD'),
 ('Elon Musk', 'Tesla'),
 ('Sundar Pichai', 'Alphabet'),
 ('Sundar Pichai', 'AMD'),
 ('Sundar Pichai', 'Tesla'),
 ('Lisa Su', 'Alphabet'),
 ('Lisa Su', 'AMD'),
 ('Lisa Su', 'Tesla')]
```

```
from collections import Counter
co_occurrence_probabilities = Counter()
for person, company in all_pairs:
    count = 0
    for doc in docs:
        if person in doc and company in doc:
            count += 1
    co_occurrence_probabilities[(person, company)] = count / len(docs)
```

Using np arrays

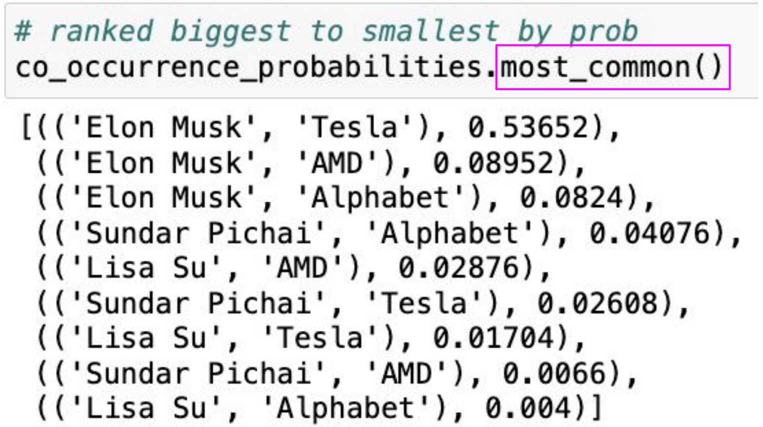


```
# using np arrays
import numpy as np

all_pairs = np.array([(person, company)
                      for person in people
                      for company in companies])
```

```
all_pairs
```

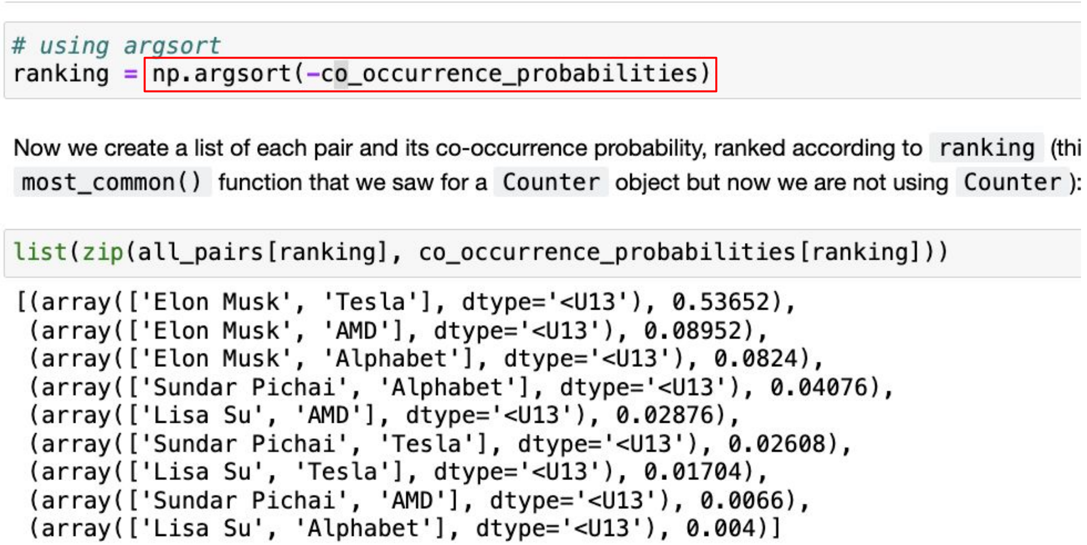
```
array([['Elon Musk', 'Alphabet'],
       ['Elon Musk', 'AMD'],
       ['Elon Musk', 'Tesla'],
       ['Sundar Pichai', 'Alphabet'],
       ['Sundar Pichai', 'AMD'],
       ['Sundar Pichai', 'Tesla'],
       ['Lisa Su', 'Alphabet'],
       ['Lisa Su', 'AMD'],
       ['Lisa Su', 'Tesla']], dtype='<U13')
```

```
n_pairs = len(all_pairs)

# create empty np array object to store probabilities
co_occurrence_probabilities = np.zeros(n_pairs)

for idx, (person, company) in enumerate(all_pairs):
    count = 0
    for doc in docs:
        if person in doc and company in doc:
            count += 1
    co_occurrence_probabilities[idx] = count / len(docs)
```



```python
# ranked biggest to smallest by prob
co_occurrence_probabilities.most_common()
```

```
[(('Elon Musk', 'Tesla'), 0.53652),
 (('Elon Musk', 'AMD'), 0.08952),
 (('Elon Musk', 'Alphabet'), 0.0824),
 (('Sundar Pichai', 'Alphabet'), 0.04076),
 (('Lisa Su', 'AMD'), 0.02876),
 (('Sundar Pichai', 'Tesla'), 0.02608),
 (('Lisa Su', 'Tesla'), 0.01704),
 (('Sundar Pichai', 'AMD'), 0.0066),
 (('Lisa Su', 'Alphabet'), 0.004)]
```

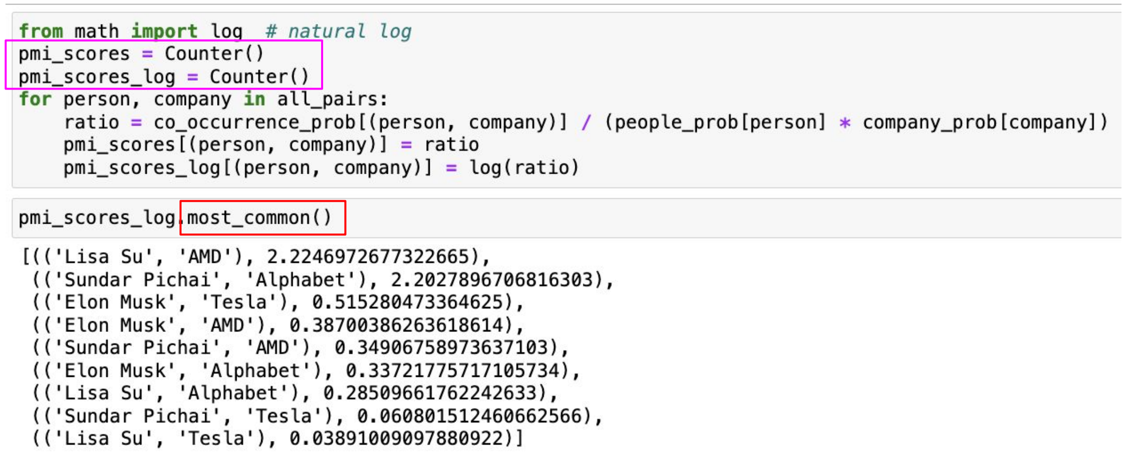


```python
# using argsort
ranking = np.argsort(-co_occurrence_probabilities)
```

Now we create a list of each pair and its co-occurrence probability, ranked according to `ranking` (thi
`most_common()` function that we saw for a `Counter` object but now we are not using `Counter`):

```python
list(zip(all_pairs[ranking], co_occurrence_probabilities[ranking]))
```

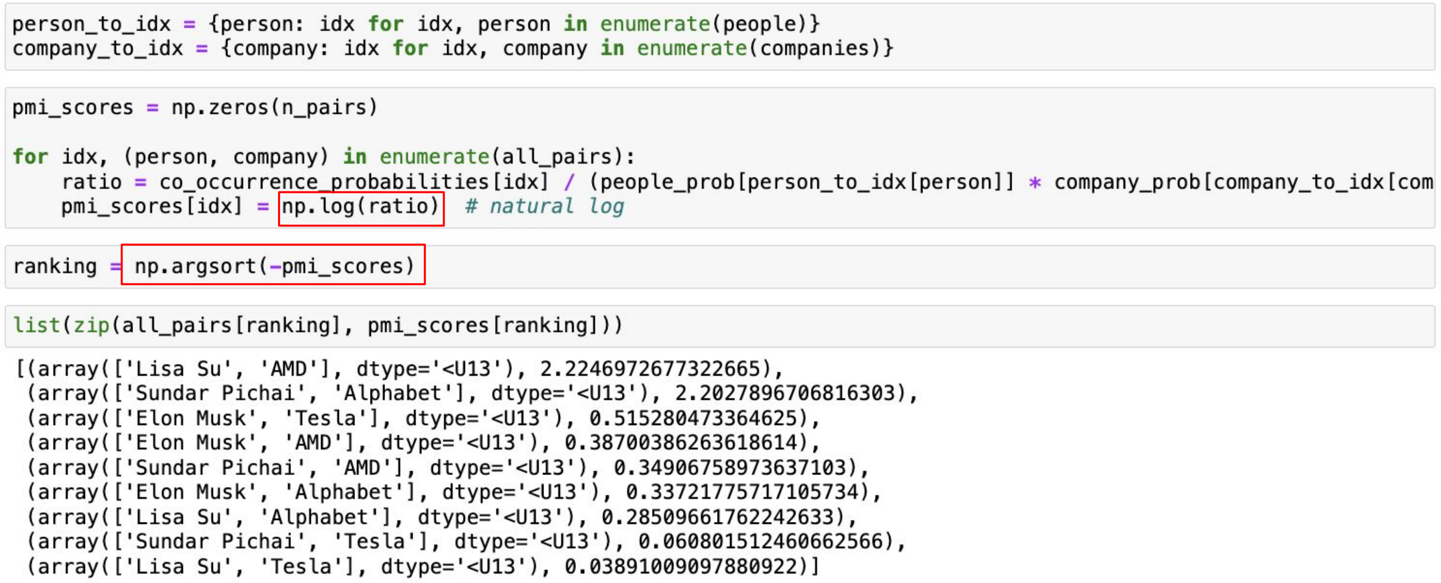
```
[(array(['Elon Musk', 'Tesla'], dtype='<U13'), 0.53652),
 (array(['Elon Musk', 'AMD'], dtype='<U13'), 0.08952),
 (array(['Elon Musk', 'Alphabet'], dtype='<U13'), 0.0824),
 (array(['Sundar Pichai', 'Alphabet'], dtype='<U13'), 0.04076),
 (array(['Lisa Su', 'AMD'], dtype='<U13'), 0.02876),
 (array(['Sundar Pichai', 'Tesla'], dtype='<U13'), 0.02608),
 (array(['Lisa Su', 'Tesla'], dtype='<U13'), 0.01704),
 (array(['Sundar Pichai', 'AMD'], dtype='<U13'), 0.0066),
 (array(['Lisa Su', 'Alphabet'], dtype='<U13'), 0.004)]
```



```python
from math import log  # natural log
pmi_scores = Counter()
pmi_scores_log = Counter()
for person, company in all_pairs:
    ratio = co_occurrence_prob[(person, company)] / (people_prob[person] * company_prob[company])
    pmi_scores[(person, company)] = ratio
    pmi_scores_log[(person, company)] = log(ratio)
```

```python
pmi_scores_log.most_common()
```

```
[(('Lisa Su', 'AMD'), 2.2246972677322665),
 (('Sundar Pichai', 'Alphabet'), 2.2027896706816303),
 (('Elon Musk', 'Tesla'), 0.515280473364625),
 (('Elon Musk', 'AMD'), 0.38700386263618614),
 (('Sundar Pichai', 'AMD'), 0.34906758973637103),
 (('Elon Musk', 'Alphabet'), 0.33721775717105734),
 (('Lisa Su', 'Alphabet'), 0.28509661762242633),
 (('Sundar Pichai', 'Tesla'), 0.060801512460662566),
 (('Lisa Su', 'Tesla'), 0.03891090978809922)]
```



```python
person_to_idx = {person: idx for idx, person in enumerate(people)}
company_to_idx = {company: idx for idx, company in enumerate(companies)}
```

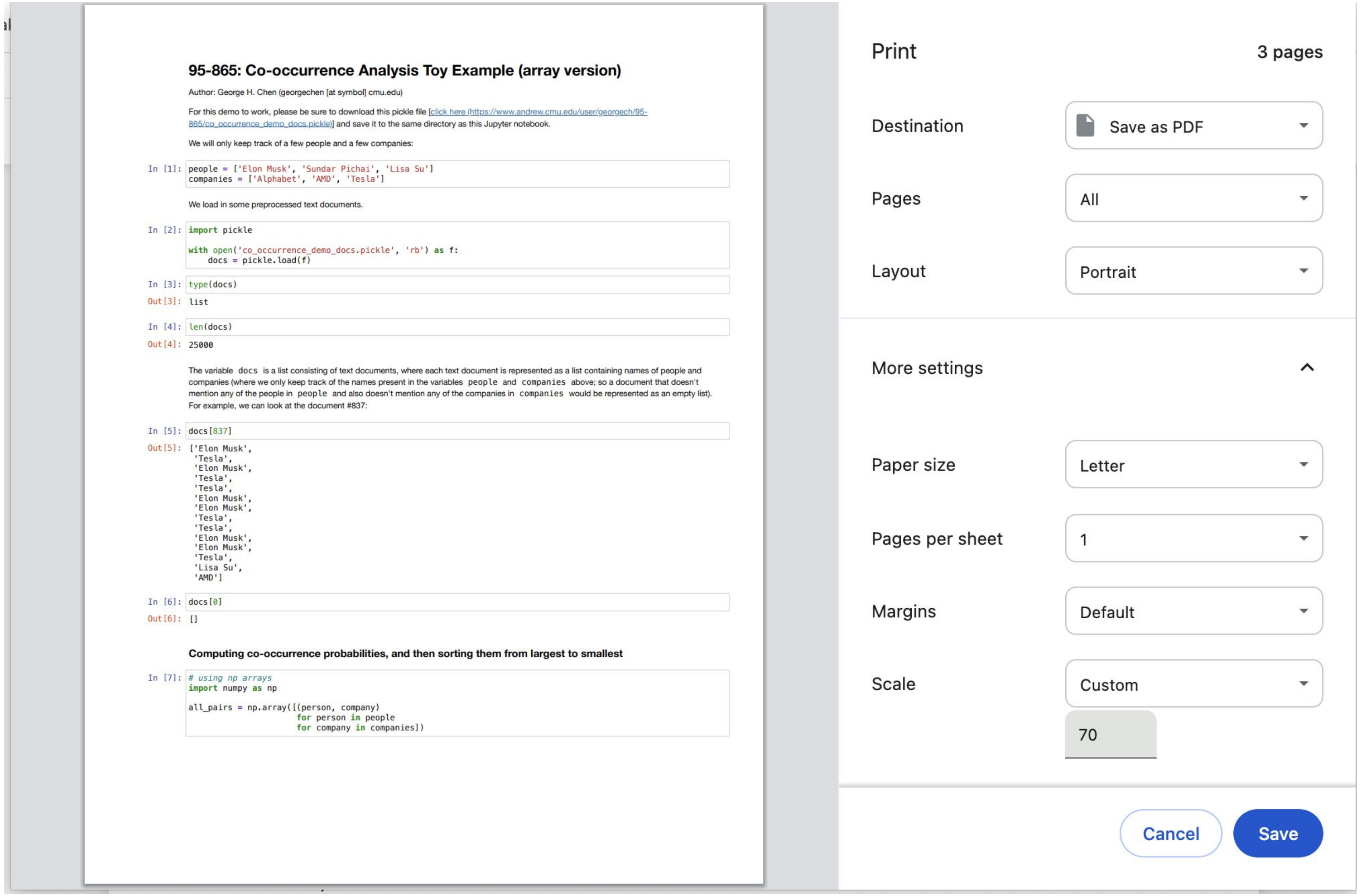
```python
pmi_scores = np.zeros(n_pairs)

for idx, (person, company) in enumerate(all_pairs):
    ratio = co_occurrence_probabilities[idx] / (people_prob[person_to_idx[person]] * company_prob[company_to_idx[com
    pmi_scores[idx] = np.log(ratio)  # natural log
```

```python
ranking = np.argsort(-pmi_scores)
```

```python
list(zip(all_pairs[ranking], pmi_scores[ranking]))
```

```
[(array(['Lisa Su', 'AMD'], dtype='<U13'), 2.2246972677322665),
 (array(['Sundar Pichai', 'Alphabet'], dtype='<U13'), 2.2027896706816303),
 (array(['Elon Musk', 'Tesla'], dtype='<U13'), 0.515280473364625),
 (array(['Elon Musk', 'AMD'], dtype='<U13'), 0.38700386263618614),
 (array(['Sundar Pichai', 'AMD'], dtype='<U13'), 0.34906758973637103),
 (array(['Elon Musk', 'Alphabet'], dtype='<U13'), 0.33721775717105734),
 (array(['Lisa Su', 'Alphabet'], dtype='<U13'), 0.28509661762242633),
 (array(['Sundar Pichai', 'Tesla'], dtype='<U13'), 0.060801512460662566),
 (array(['Lisa Su', 'Tesla'], dtype='<U13'), 0.03891090978809922)]
```

3. Saving jupyter notebook as pdf



95-865: Co-occurrence Analysis Toy Example (array version)

Author: George H. Chen (georgechen [at symbol] cmu.edu)

For this demo to work, please be sure to download this pickle file [click here (https://www.andrew.cmu.edu/user/georgech/95-865/co_occurrence_demo_docs.pickle)] and save it to the same directory as this Jupyter notebook.

We will only keep track of a few people and a few companies:

```
In [1]: people = ['Elon Musk', 'Sundar Pichai', 'Lisa Su']
        companies = ['Alphabet', 'AMD', 'Tesla']
```

We load in some preprocessed text documents.

```
In [2]: import pickle

        with open('co_occurrence_demo_docs.pickle', 'rb') as f:
            docs = pickle.load(f)
```

```
In [3]: type(docs)

Out[3]: list
```

```
In [4]: len(docs)

Out[4]: 25000
```

The variable `docs` is a list consisting of text documents, where each text document is represented as a list containing names of people and companies (where we only keep track of the names present in the variables `people` and `companies` above; so a document that doesn't mention any of the people in `people` and also doesn't mention any of the companies in `companies` would be represented as an empty list). For example, we can look at the document #837:

```
In [5]: docs[837]

Out[5]: ['Elon Musk',
         'Tesla',
         'Elon Musk',
         'Tesla',
         'Tesla',
         'Elon Musk',
         'Elon Musk',
         'Tesla',
         'Tesla',
         'Elon Musk',
         'Elon Musk',
         'Tesla',
         'Lisa Su',
         'AMD']
```

```
In [6]: docs[0]

Out[6]: []
```

Computing co-occurrence probabilities, and then sorting them from largest to smallest

```
In [7]: # using np arrays
        import numpy as np

        all_pairs = np.array([(person, company)
                              for person in people
                              for company in companies])
```

Print 3 pages

Destination: Save as PDF

Pages: All

Layout: Portrait

More settings

Paper size: Letter

Pages per sheet: 1

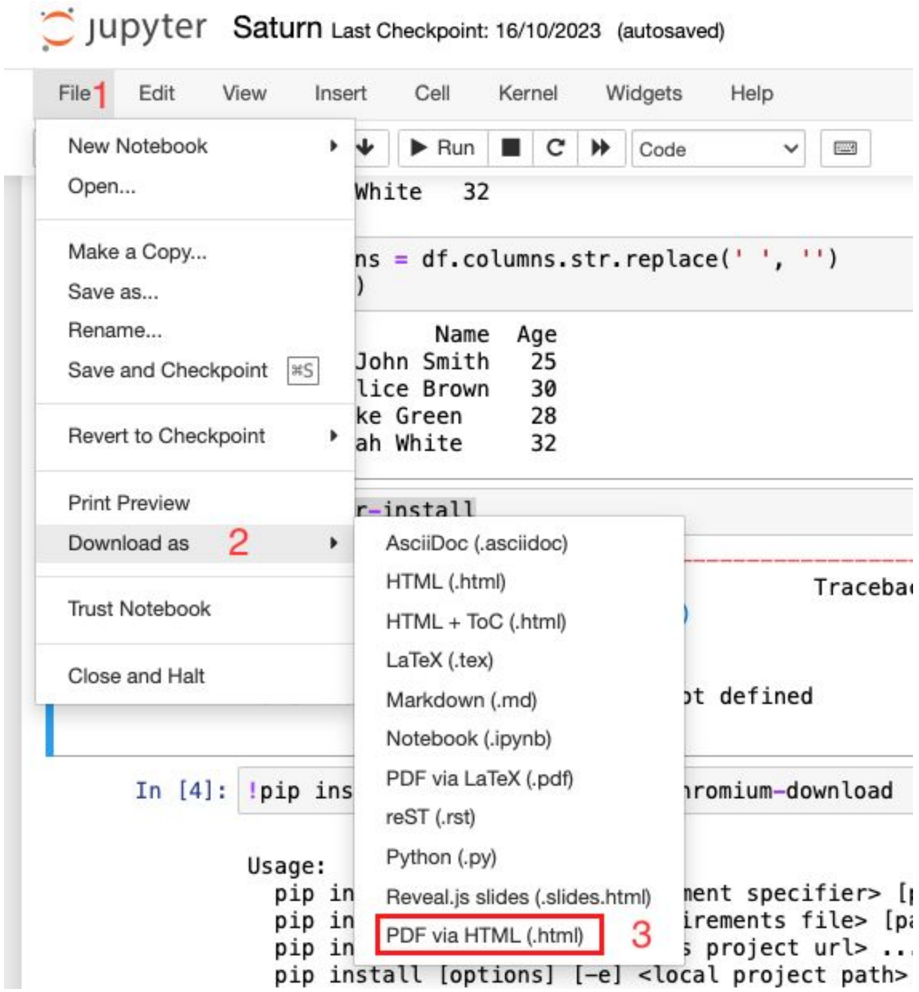
Margins: Default

Scale: Custom

70

Cancel Save

Using interface



```
White   32
ns = df.columns.str.replace(' ', '')
)
        Name  Age
John Smith   25
lice Brown   30
ke Green     28
ah White     32
r-install
                                   Traceba
ot defined
In [4]: !pip ins                    hromium-download
Usage:
  pip in                    ment specifier> [
  pip in                    irements file> [p
  pip in                    s project url> ..
  pip install [options] [-e] <local project path>
```

Using command line prompt

```
jupyter-nbconvert --to pdfviahtml example.ipynb
```