In [69]:

```
%%html
<style>
table {display: block;}
td {
  font-size: 18px
}
.rendered_html { font-size: 28px; }
*{ line-height: 200%; }
</style>
```

Natural Language Processing and the Web WS 2022/23 - Practice Class - Tutorial 4

In this practice class, we will discuss different ways of analyzing and visualizing text corpora with different libraries. Before building models for different NLP applications, it is very important to understanding how the corpora looks like.

Content

- Text concordance and frequency of n-grams
- Introduction to Numpy and Pandas
- Plotting different figures with matplotlib and Seaborn
- Text to vector conversion

Text Concordance in NLTK

Concordance finds a word in a text and displays the context in which this word is used. The Concordance function in NLTK takes a word of interest as an input and returns a list of every instance where that word is mentioned in the underlying text. The output includes some snippet words surrounding the target word.

In [2]:

```
import nltk
contents = open("data/book-excerpts.tab.txt").read()
book_tokens = nltk.word_tokenize(contents)
#nlt.Text performs a variety of analyses on the text's contexts (e.g., counting,
# concordancing, collocation discovery), and display the results.
book_text = nltk.Text(book_tokens)
# show all the concordances for the word 'doctor'
book_text.concordance('doctor',lines=-1)
```

```
Displaying -1 of 37 matches:
ief for us when the door opened and Doctor Livesey came in on his vis
it to my
ame in on his visit to my father Oh doctor we cried what shall we do
? Where i
 ? A fiddle-stick 's end ! said the doctor No more wounded than you o
r I The m
 When I got back with the basin the doctor had already ripped up the
captain '
ith great spirit Prophetic said the doctor touching this picture with
his fing
y about him First he recognized the doctor with an unmistakable frown
; then h
There is no Black Dog here said the doctor except what you have on yo
ur own ba
nterrupted Much I care returned the doctor It 's the name of a buccan
eer of my
most fainting Now mind you said the doctor I clear my conscience -- t
he name o
```

In [3]:

```
# using the Moby Dick text from the NLTK books
from nltk.book import *
text1.concordance('frightened')
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
Displaying 6 of 6 matches:
t , and was fast asleep . But the frightened master comes to him , and
shrieks
nd never came to good . He got so frightened about his plaguy soul , t
hat he s
 says he . Slid ! man , but I was frightened . Such a phiz ! But , som
ehow , n
aking of that buffalo robe to the frightened colt ! Though neither kno
ws where
ndostan coast with a deck load of frightened horses , careens , buries
, rolls
st - sou - east , sir ," said the frightened steersman . " Thou liest
!" smiti
```

In [4]:

```
# Get the individual lines of the concordance for further processing
def get_concordance_text(word, text, left_margin = 5, right_margin = 5):
    tokens = nltk.word_tokenize(text)
    text = nltk.Text(tokens)
    # indexes of the tokens
    indices = nltk.ConcordanceIndex(text.tokens, key = lambda s: s.lower())
    concordance_txt = ([text.tokens[offset-left_margin:offset+right_margin] # sc
                        for offset in indices.offsets(word)])
    return [''.join([snippet+' ' for snippet in concordances]) for concordances

results = get_concordance_text('doctor', contents)
print(len(results))
for result in results:
    print (result)
```

```
37
when the door opened and Doctor Livesey came in on
visit to my father Oh doctor we cried what shall
's end ! said the doctor No more wounded than
back with the basin the doctor had already ripped up
great spirit Prophetic said the doctor touching this picture with
him First he recognized the doctor with an unmistakable frown
Black Dog here said the doctor except what you have
Much I care returned the doctor It 's the name
Now mind you said the doctor I clear my conscience
I 'll raise Cain Your doctor hisself said one glass
I was reassured by the doctor 's words now quoted
And now matey did that doctor say how long I
position on the edge That doctor 's done me he
! -- to that eternal doctor swab and tell him
the whole story to the doctor for I was in
death for him and the doctor was suddenly taken up
at once and ride for Doctor Livesey would have left
to ride forward to the doctor 's in search of
```

Frequencies distribution

Show the number of times each unique word is used in a text. This helps us to understand the topic in the text and how they are discussed.

In [5]:

```
fdist = FreqDist(book_tokens)
# count unique words
count = len(fdist)
count
```

Out[5]:

```
12061
```

In [6]:

```
#print most common 100 words
fdist.most_common (100)
```

Out[6]:

```
[('the', 6711),
 ('and', 4461),
 ('of', 3293),
 ('to', 3179),
 ('I', 3102),
 ('a', 2754),
 ('in', 1901),
 ('was', 1592),
 ('that', 1460),
 ('it', 1280),
 ('her', 1099),
 ('with', 1091),
 ('you', 1023),
 ('as', 1018),
 ('he', 1006),
 ('had', 921),
 ('said', 845),
```

In [7]:

```
#proportions of the 50 most common words in the book_sample corpus
print (sum(common[1] for common in fdist.most_common (50))/len(book_tokens)*100)
```

```
42.5002490593221
```

In [8]:

```
# show words that occur only once (hapaxes)
fdist.hapaxes()
```

Out[8]:

```
['Category',
 'Text',
 'include=True',
 'unsteadied',
 'fouled',
 'fighting',
 'deary',
 'death-hurt',
 'scuffle',
 'Wounded',
 'fiddle-stick',
 'trebly',
 'worthless',
 'tattooed',
 'executed',
 'forearm',
 'Prophetic',
```

In [9]:

```
# print the percentage of hapaxes
print(len(fdist.hapaxes())/len(set(book_tokens))*100)
```

```
50.186551695547635
```

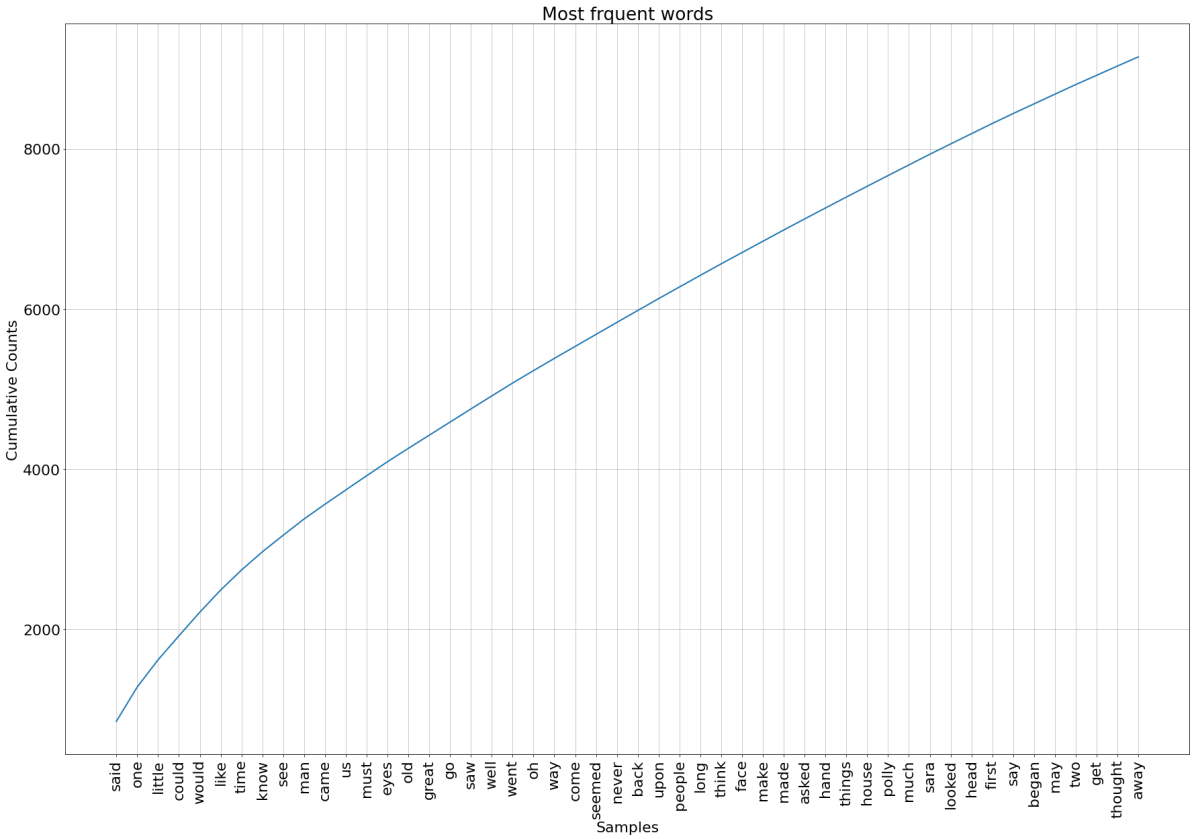
In [10]:

```
nltk.download('stopwords')
#remove stopwords
from nltk.corpus import stopwords
en_stopws = stopwords.words('english')
# remove stop words and punctuation marks
book_tokens_filtered = [t.lower() for t in book_tokens if t.lower() not in en_st
fdist = FreqDist(book_tokens_filtered)
#print most common 100 words
fdist.most_common (100)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/seidmuhieyimam/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [11]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(30, 20))
plt.rcParams.update({'font.size': 22})
# show the cumulative frequencies of the top 50 common words
fdist.plot( 50,title="Most frquent words", cumulative=True)
```

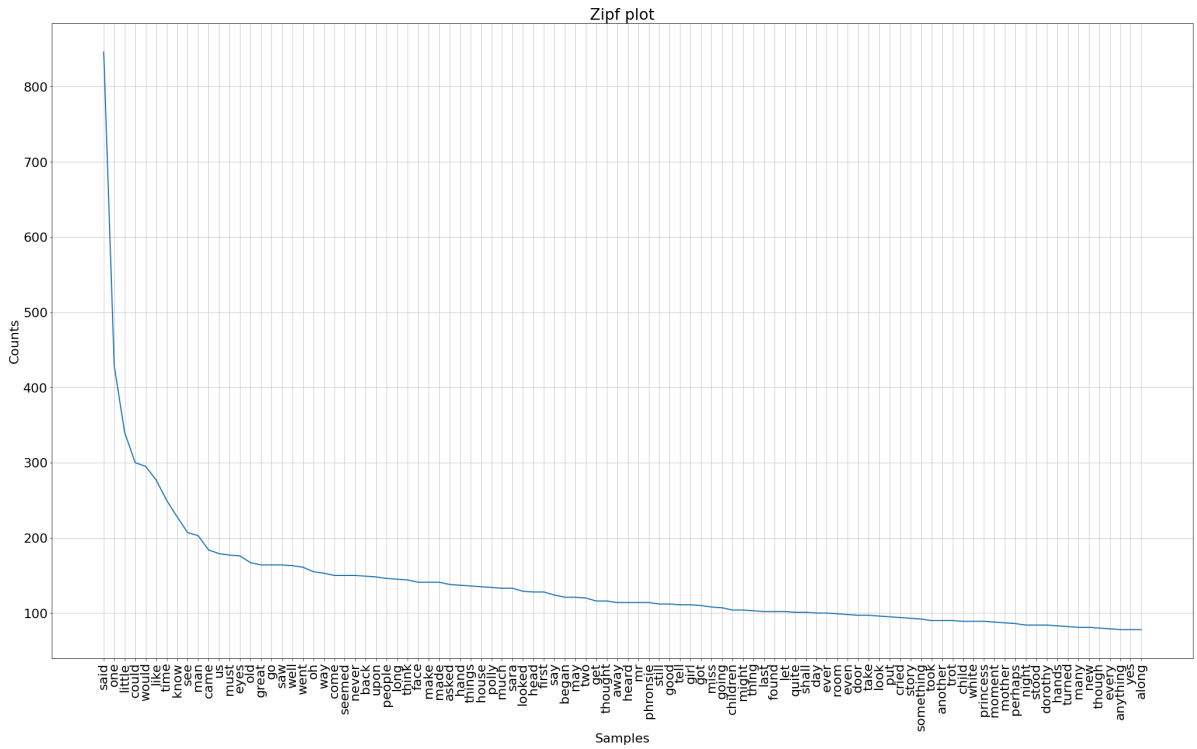


Out[11]:

```
<AxesSubplot:title={'center':'Most frquent words'}, xlabel='Samples',
ylabel='Cumulative Counts'>
```

In [12]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(35, 20))
plt.rcParams.update({'font.size': 22})
# plot the zipf plot for the most 100 words
fdist.plot(100,title="Zipf plot",)
```



Out[12]:

```
<AxesSubplot:title={'center':'Zipf plot'}, xlabel='Samples', ylabel='C
ounts'>
```

Conditional Freq Dist

A collection of frequency distributions for a single experiment run under different conditions. Conditional frequency distributions are used to record the number

of times each sample occurred, given the condition under which the experiment was run.

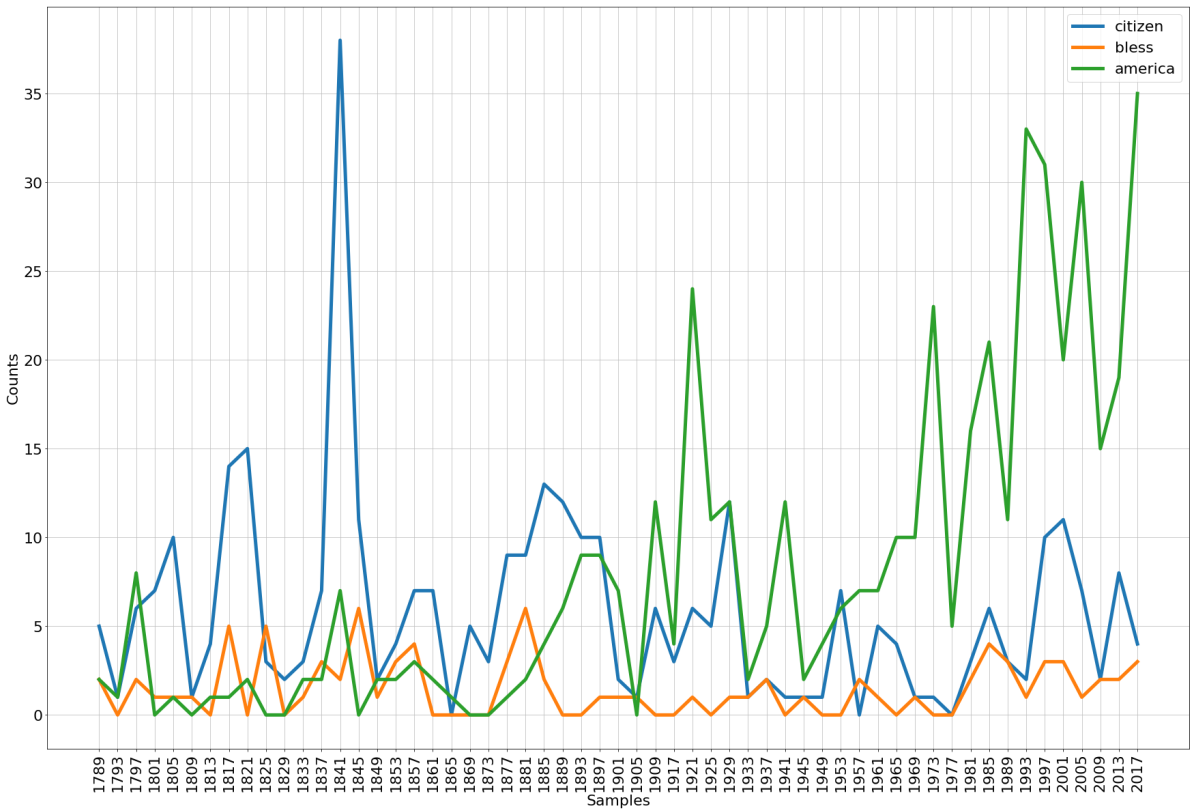
In [13]:

```
# collection of inagural speech, YEAR-PRESIDENT_NAME format
from nltk.corpus import inaugural
for fileid in inaugural.fileids():
    print ("Speach name:->", fileid)
    print ("The year part:->", fileid[:4])
    break # just show the first entry

```

```
Speach name:-> 1789-Washington.txt
The year part:-> 1789
```

In [14]:

```
#ploting conditional frequencies
from nltk.corpus import inaugural
import matplotlib.pyplot as plt
plt.figure(figsize=(30, 20))
cfd = nltk.ConditionalFreqDist(
          (target, fileid[:4])
          for fileid in inaugural.fileids()
            for w in inaugural.words(fileid)
          for target in ['america', 'citizen','bless']
          if w.lower().startswith(target))
cfd.plot(linewidth=5)
```



Out[14]:

```
<AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

N-grams

Word frequencies give a hint about the corpus you are dealing with. Moreover, you can analyze the n-grams, the sequence of N words, that will give hints on which word often occur together. This concepts are basics for a number of tasks such as computing text similarity, identifying chunks or noun phrases, disambiguating named entities and so on. N-grams are also very important build language models, which known as N-gram model.

In [15]:

```python
# read a text from a file
import os
content = open("data/ing.txt").read()
#Split words using whitespace
words = content.split()
# Count the frequencies of each word from the list
frequencies = [words.count(word) for word in words]
# show the frequencies of each word (set --> removes duplicates)
for key, val in set(zip(words,frequencies)):
    print (key, val)
```

```
wonderful 1
January 1
strong 1
before 1
borne 1
jobs 2
follow. 1
longer. 1
stealing 1
were 1
never, 1
love, 1
celebrated 1
first. 3
build 1
want 1
States 2
merely 1
safe 2
```

In [16]:

```
#let's create a method to build a dictionary of words to frequencies from the wo
def freqDict(words):
    frequencies = [words.count(word) for word in words]
    return dict( zip(words,frequencies))

# Lets sort the dictionary based on words' frequency, the most frequent at the t
def sortDict(freqdict):
    sorteddict = [(freqdict[key], key) for key in freqdict]
    sorteddict.sort()
    sorteddict.reverse()
    return sorteddict

freqdict = freqDict(words)
for word in sortDict(freqdict):
    print (word[1], word[0])
```

```
and 71
the 67
our 48
of 48
will 43
to 37
We 27
we 21
is 20
all 14
a 14
in 13
for 13
be 13
from 12
but 12
are 12
America 11
your 10
```

In [17]:

```
# remove stop words, lowercase the text
stopwords = ["the", "a", "no", "other", "we", "and", "of","will","is","in","our"
def removeStopWords(words, stopwords):
    return [word.lower() for word in words if word.lower() not in stopwords]
cleanwords = removeStopWords(words, stopwords)
freqdict = freqDict(cleanwords)
for word in sortDict(freqdict):
    print (word[1], word[0])
```

```
all 14
but 13
be 13
from 12
are 12
your 11
their 11
america 11
that 10
not 10
this 9
american 9
with 8
it 8
people 7
one 7
every 7
you. 6
while 6
```

In [18]:

```
# see the contents
content
```

Out[18]:

```
"Chief Justice Roberts, President Carter, President Clinton, Presiden
t Bush, President Obama, fellow Americans, and people of the world, t
hank you. We the citizens of America are now joined in a great nation
al effort to rebuild our country and restore its promise for all of o
ur people. Together we will determine the course of America, and the
world, for many, many years to come. We will face challenges. We will
confront hardships, but we will get the job done.\n\n\nEvery four yea
rs, we gather on these steps to carry out the orderly and peaceful tr
ansfer of power, and we are grateful to President Obama and First Lad
y Michelle Obama for their gracious aid throughout this transition. T
hey have been magnificent. Thank you.\n\nToday's ceremony, however, h
as very special meaning, because today we are not merely transferring
power from one administration to another, or from one party to anothe
r, but we are transferring power from Washington, D.C., and giving it
back to you, the people.\n\nFor too long, a small group in our natio
n's capital has reaped the rewards of government, while the people ha
ve borne the cost. Washington flourished, but the people did not shar
```

In [19]:

```
# and list of words
words = [word for word in content.lower().split(" ") if word != ""]
words
```

Out[19]:

```
['chief',
 'justice',
 'roberts,',
 'president',
 'carter,',
 'president',
 'clinton,',
 'president',
 'bush,',
 'president',
 'obama,',
 'fellow',
 'americans,',
 'and',
 'people',
 'of',
 'the',
```

In [20]:

```
# and n grams of size 2
ngrams = zip(*[words[i:] for i in range(2)])
[" ".join(ngram) for ngram in ngrams]
```

Out[20]:

```
['chief justice',
 'justice roberts,',
 'roberts, president',
 'president carter,',
 'carter, president',
 'president clinton,',
 'clinton, president',
 'president bush,',
 'bush, president',
 'president obama,',
 'obama, fellow',
 'fellow americans,',
 'americans, and',
 'and people',
 'people of',
 'of the',
 'the world,',
```

In [21]:

```
# put all together - generate n-grams from texts, here n = 4
def getNgrams(text, n):
    # lowercase the text
    text = text.lower()
    # create words from the text
    words = [word for word in text.split(" ") if word != ""]
    ngrams = zip(*[words[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]
getNgrams(content, 4)
```

Out[21]:

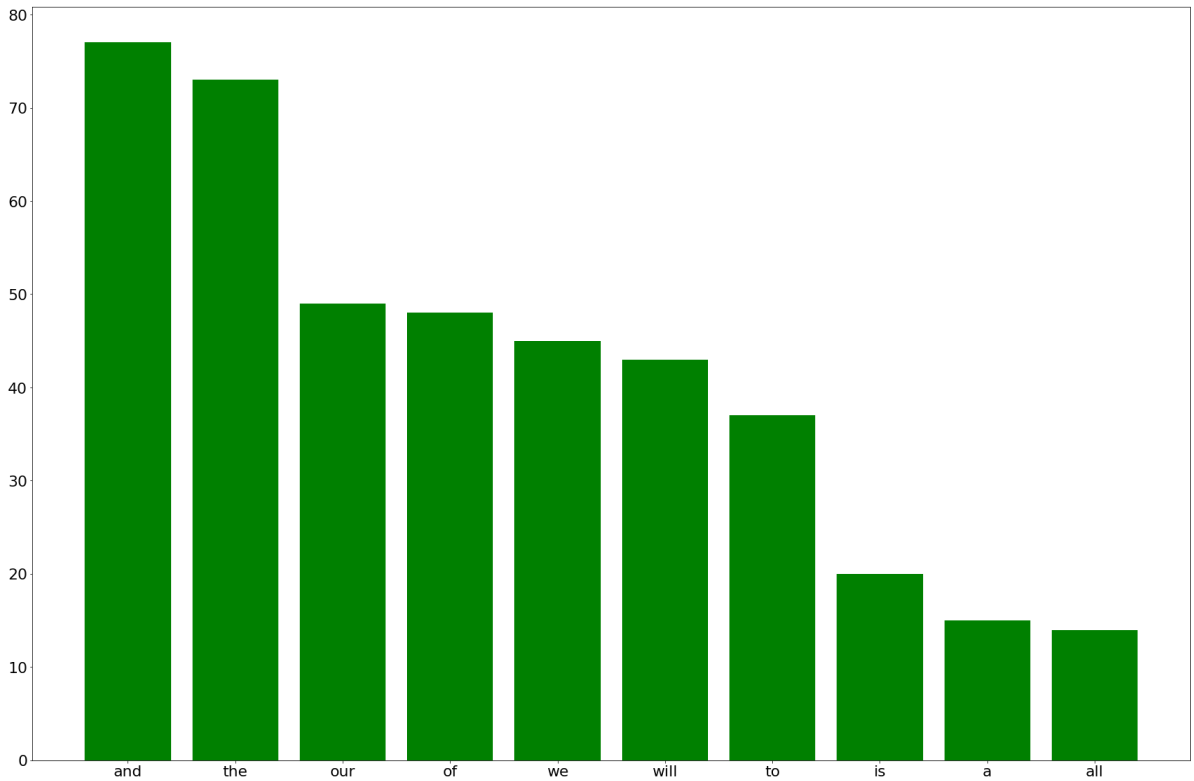
```
['chief justice roberts, president',
 'justice roberts, president carter,',
 'roberts, president carter, president',
 'president carter, president clinton,',
 'carter, president clinton, president',
 'president clinton, president bush,',
 'clinton, president bush, president',
 'president bush, president obama,',
 'bush, president obama, fellow',
 'president obama, fellow americans,',
 'obama, fellow americans, and',
 'fellow americans, and people',
 'americans, and people of',
 'and people of the',
 'people of the world,',
 'of the world, thank',
 'the world, thank you.',
```

In [22]:

```
# Draw bar chart for the top n words, with stopwords
import pandas as pd
import matplotlib.pyplot as plt
plt.figure(figsize=(30, 20))
plt.rcParams.update({'font.size': 22})
df = pd.DataFrame({'words': words})
df['frequency'] = 1
# group rows by words, count the frequencies in each group
# and sort by frequencies
df = df.groupby('words').sum().sort_values('frequency', ascending=False)
# draw the top n frequent words
ndf = df.head(10)
plt.bar(ndf.index, ndf.frequency, width=0.8, color='g')
```

Out[22]:

<BarContainer object of 10 artists>

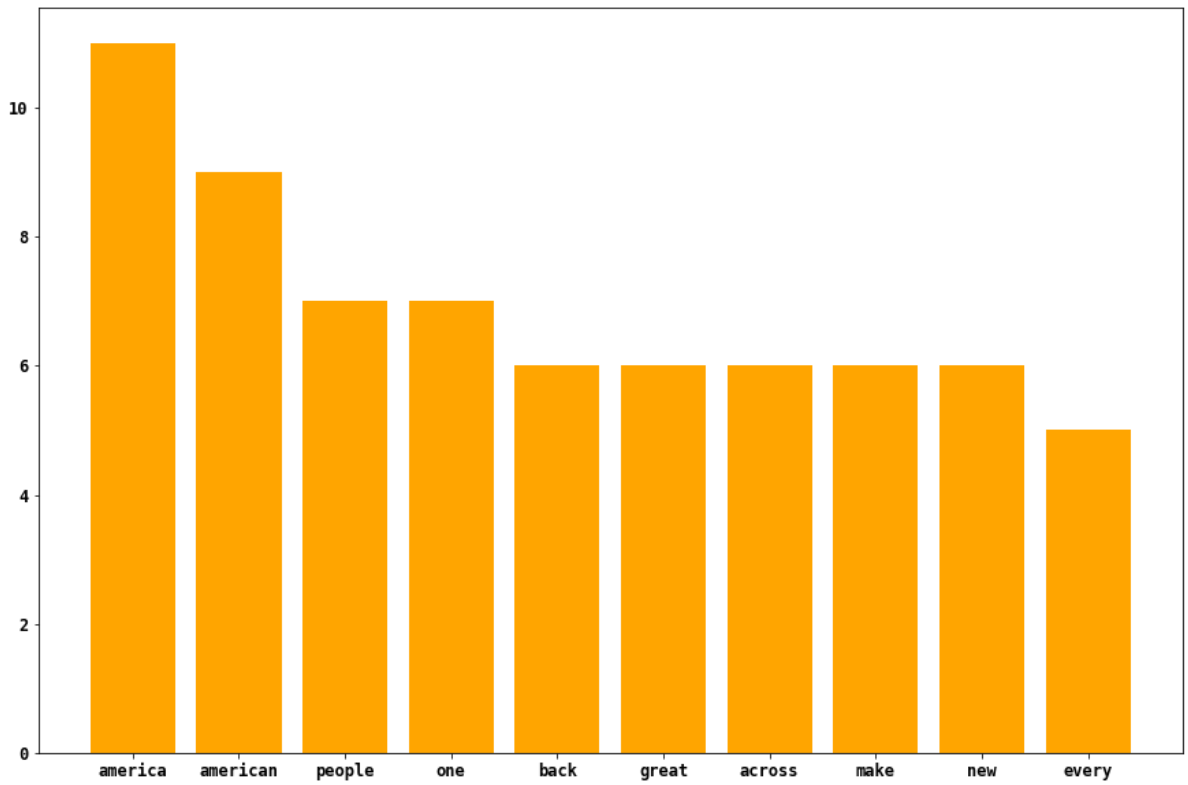


In [23]:

```
# Draw bar chart for the top n words, without stopwords
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))
df = pd.DataFrame({'words': removeStopWords(words, stopwords)})
df['frequency'] = 1
# group rows by words, count the frequencies in each group
# and sort by frequencies
df = df.groupby('words').sum().sort_values('frequency', ascending=False)
# draw the top n frequent words
ndf = df.head(10)
# increase width of figure (so we can see the words clearly) and change font siz
plt.figure(figsize=(15,10))
font = {'family' : 'monospace',
        'weight' : 'bold',
        'size'   : 12}
plt.rc('font', **font)

plt.bar(ndf.index, ndf.frequency, width=0.8, color='orange')
```

Out[23]:

<BarContainer object of 10 artists>



Excercise 1 (4 pts)

1. Use the NLTK book corpus (`from nltk.corpus import inaugural`) to analyze the `Inaugural Address Corpus`. Which words are frequent for all inaugural address speech over time (since 1789)? For example if we compare the two words `america` and `citizen` with the list `[4,1,1,4]` and `[3,2,2,3]` respectively, we can tell that `citizen` is more popular as it in average occurs more often each year. Draw a diagram which shows the frequency of the top 10 words over time. `inaugural.fileids()` will list all the inaugural texts. `inaugural.raw(fileids=inaugural.fileid[0])` will give you the raw text for the first speech. Do not use the NLTK built-in methods.

2. Improve the `getNgrams` function so that it will return also the frequencies of each n-grams in the range `1-->N` (sorted in descending order). Try to generate n-grams from 2-4 excluding unigrams (inclusive 2 and 4)?

Example

The lazy dog jumps 123

The lazy dog 320

The lazy 589

...

Word Similarity

NLTK provides a simple text similarity function that shows which words are used in a similar context

In [24]:

```
# Show text concordance for the word "frightened"
text1.concordance('frightened')
```

```
Displaying 6 of 6 matches:
t , and was fast asleep . But the frightened master comes to him , and
shrieks
nd never came to good . He got so frightened about his plaguy soul , t
hat he s
 says he . Slid ! man , but I was frightened . Such a phiz ! But , som
ehow , n
aking of that buffalo robe to the frightened colt ! Though neither kno
ws where
ndostan coast with a deck load of frightened horses , careens , buries
, rolls
st - sou - east , sir ," said the frightened steersman . " Thou liest
!" smiti
```

In [25]:

```
# show words that are used in a similar cotntext as "frightened"
text1.similar("frightened")
```

```
silent wild racing
```

In [26]:

```
text1.concordance('racing')
```

```
Displaying 1 of 1 matches:
nail - stubbs of the steel shoes of racing horses ." " Horse - shoe st
ubbs , s
```

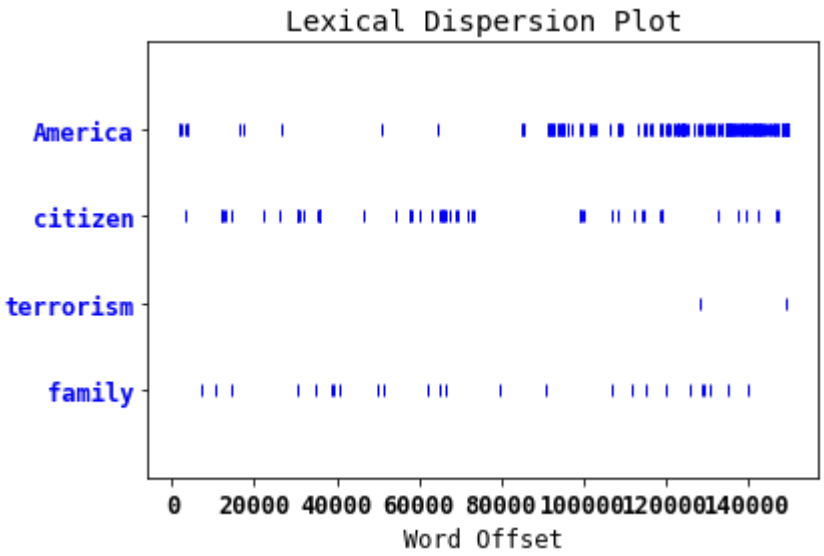
Dispersion plot

The dispersion plot is helpful to determine the location of a word in a sequence of text sentences. It shows the spread of any particular word across the whole

text. In the plot, the x axis represents the narrative time measured by the number of words in the text. Also, when the desired word appears in the text a black vertical line is plotted, otherwise it remains blank (white line).

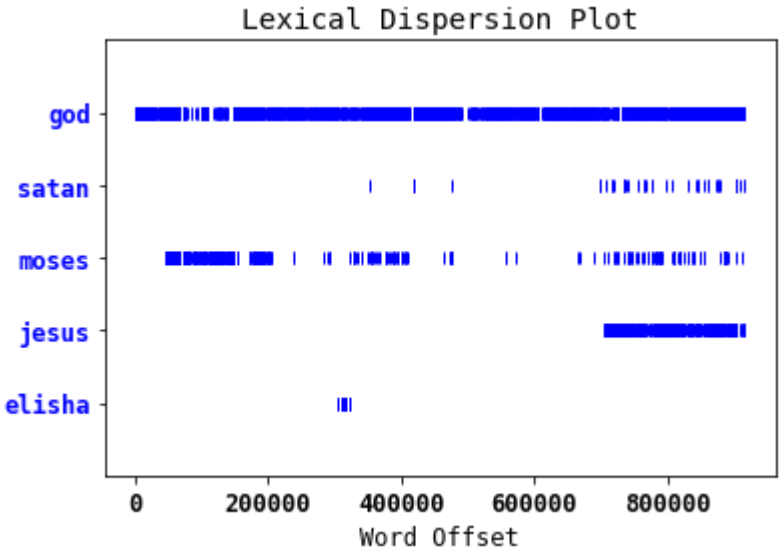
In [27]:

```
1 text4.dispersion_plot(['America', 'citizen', 'terrorism', 'family'])
```



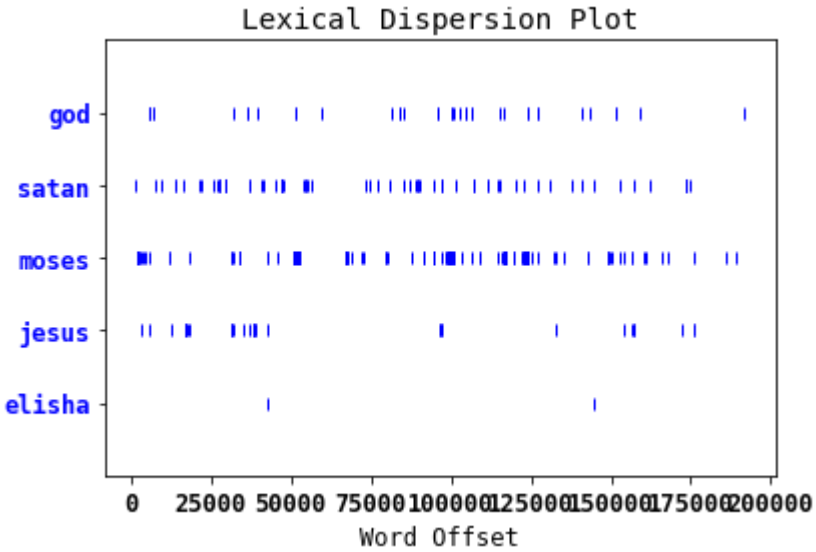
In [28]:

```
# Dispersion plot from bible text
lines = []
with open("data/kjvdat.txt",'r') as bib_file:
    for line in bib_file:
        lines.append(line.split('| ')[1][:-2])

bib_tokens = nltk.word_tokenize(' '.join(lines).lower())
bib_text = nltk.Text(bib_tokens)
bib_text.dispersion_plot(['god', 'satan', 'moses', 'jesus', 'elisha'])
```



In [29]:

```
# Dispersion plot from quran text
lines = []
with open("data/en.sahih.txt",'r') as qur_file:
    for line in qur_file:
        lines.append(line.split('|')[2])

qur_tokens = nltk.word_tokenize(' '.join(lines).lower())
qur_text = nltk.Text(qur_tokens)
qur_text.dispersion_plot(['god', 'satan', 'moses', 'jesus', 'elisha'])
```



Word clouds

Word cloud or tag cloud represents the frequency or importance of each word in a corpus.

In [30]:

```
!pip install wordcloud
```

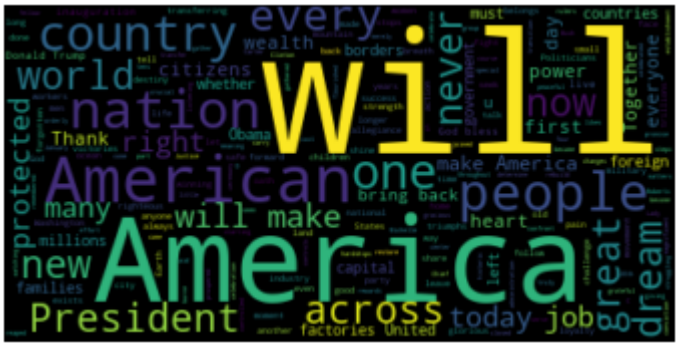
```
Requirement already satisfied: wordcloud in /Users/seidmuhieyimam/opt/
miniconda3/lib/python3.7/site-packages (1.8.1)
Requirement already satisfied: pillow in /Users/seidmuhieyimam/opt/min
iconda3/lib/python3.7/site-packages (from wordcloud) (8.4.0)
Requirement already satisfied: numpy>=1.6.1 in /Users/seidmuhieyimam/o
pt/miniconda3/lib/python3.7/site-packages (from wordcloud) (1.21.3)
Requirement already satisfied: matplotlib in /Users/seidmuhieyimam/op
t/miniconda3/lib/python3.7/site-packages (from wordcloud) (3.4.3)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/seidmuhieyim
am/opt/miniconda3/lib/python3.7/site-packages (from matplotlib->wordcl
oud) (3.0.7)
Requirement already satisfied: cycler>=0.10 in /Users/seidmuhieyimam/o
pt/miniconda3/lib/python3.7/site-packages (from matplotlib->wordcloud)
(0.11.0)
Requirement already satisfied: python-dateutil>=2.7 in /Users/seidmuhi
eyimam/opt/miniconda3/lib/python3.7/site-packages (from matplotlib->wo
rdcloud) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/seidmuhieyi
mam/opt/miniconda3/lib/python3.7/site-packages (from matplotlib->wordc
loud) (1.3.2)
Requirement already satisfied: six>=1.5 in /Users/seidmuhieyimam/opt/m
iniconda3/lib/python3.7/site-packages (from python-dateutil>=2.7->matp
lotlib->wordcloud) (1.16.0)
WARNING: You are using pip version 22.0.3; however, version 22.3 is av
ailable.
You should consider upgrading via the '/Users/seidmuhieyimam/opt/minic
onda3/bin/python -m pip install --upgrade pip' command.
```

In [ ]:

```

```

In [31]:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
%matplotlib inline
wordcloud = WordCloud().generate(content)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



In [32]:

```
# get the german stopwords from NLTK
from nltk.corpus import stopwords
stopwords = stopwords.words('german')
content = open("data/deu.txt").read()
words = content.split(" ")
cleanwords = removeStopWords(words, stopwords)

freqdict = freqDict(cleanwords)
# Build a dictionary of word:frequency pairs
d ={}
for word in sortDict(freqdict):
    d[word[1]] =word[0]
wordcloud = WordCloud()
# Create Word cloud from dictionaries based on frequencies
wordcloud.generate_from_frequencies(frequencies=d)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Introduction to Numpy and Pandas

Numpy

Numpy provides math and data manipulations using an ndarray (n dimensional array) objects.

In [33]:

```
# create numpy array
import numpy as np
list1 = [0,1,2,3,4]
listnp = np.array(list1)
print(listnp)
print("types of listnp:",type(listnp))
```

```
[0 1 2 3 4]
types of listnp: <class 'numpy.ndarray'>
```

In [34]:

```
# add 2 to each memebrs of the array
listnp +=  2
print(listnp)
```

```
[2 3 4 5 6]
```

In [35]:

```
# Create a 2d array from a list of lists
list2 = [[0,1,2], [3,4,5], [6,7,8]]
listnp2 = np.array(list2)
print(listnp2)
print("shapes:",listnp2.shape)
print("dimesnion:",listnp2.ndim)
print("total number of items:",listnp2.size)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
shapes: (3, 3)
dimesnion: 2
total number of items: 9
```

In [36]:

```
print("all:",listnp2)
# Extract the first 2 rows and columns
print("first 2:",listnp2[:2, :2])
```

```
all: [[0 1 2]
 [3 4 5]
 [6 7 8]]
first 2: [[0 1]
 [3 4]]
```

In [37]:

```
# Reverse only the row positions
listnp2[::-1, ]
```

Out[37]:

```
array([[6, 7, 8],
       [3, 4, 5],
       [0, 1, 2]])
```

In [38]:

```
# Reverse the row and column positions
listnp2[::-1, ::-1]
```

Out[38]:

```
array([[8, 7, 6],
       [5, 4, 3],
       [2, 1, 0]])
```

In [39]:

```
# mean, max and min
print("Mean value is: ", listnp2.mean())
print("Max value is: ", listnp2.max())
print("Min value is: ", listnp2.min())
# Row wise and column wise min
print("Column wise minimum: ", np.amin(listnp2, axis=0))
print("Row wise minimum: ", np.amin(listnp2, axis=1))
```

```
Mean value is:  4.0
Max value is:  8
Min value is:  0
Column wise minimum:  [0 1 2]
Row wise minimum:  [0 3 6]
```

In [40]:

```
# Add new row
newrow = [9,10,11]
listnp3 = np.vstack([listnp2, newrow])
print (listnp3)
print (listnp3.shape)
# Reshape a 4x4 array to 3x4 array
print(listnp3.reshape(3, 4))
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
(4, 3)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [41]:

```
# create 0 to 4 (size of 5)
print(np.arange(5))
# 0 to 9
print(np.arange(0, 10))
# 0 to 9 with step of 2
print(np.arange(0, 10, 2))
# 10 to 1, decreasing order
print(np.arange(10, 0, -1))
```

```
[0 1 2 3 4]
[0 1 2 3 4 5 6 7 8 9]
[0 2 4 6 8]
[10  9  8  7  6  5  4  3  2  1]
```

In [42]:

```
# arry of zeros
print(np.zeros([2,2]))

# array of one
print(np.ones([2,2,3]))
```

```
[[0. 0.]
 [0. 0.]]
[[[1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]]]
```

In [43]:

```
# Random numbers between [0,1) of shape 2,2
print("1.", np.random.rand(2,2))
# Normal distribution with mean=0 and variance=1 of shape 2,2
print("2.", np.random.randn(2,2))
# Random integers between [0, 10) of shape 2,2
print("3.", np.random.randint(0, 10, size=[2,2]))
# One random number between [0,1)
print("4.", np.random.random())
# Random numbers between [0,1) of shape 2,2
print("5.", np.random.random(size=[2,2])) # same as np.random.rand(2,2)
# Pick 10 items from a given list, with equal probability
print("6.", np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10))
# Pick 10 items from a given list with a predefined probability 'p'
print("7.", np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10, p=[0.3, .1, 0.1
```

```
1. [[0.19246673 0.50970366]
 [0.41393322 0.86434462]]
2. [[-0.19693399  0.5782392 ]
 [ 1.11055831  0.29546924]]
3. [[9 4]
 [2 5]]
4. 0.7132395901511481
5. [[0.99882933 0.97153534]
 [0.77046502 0.24215187]]
6. ['i' 'u' 'i' 'e' 'i' 'u' 'i' 'i' 'u' 'a']
7. ['i' 'a' 'o' 'o' 'a' 'a' 'o' 'e' 'a' 'a']
```

In [44]:

```

## Set the random seed == same random nubers will be generated if the process re
np.random.seed(100)
# Create random integers of size 10 between [0,10)
listnp4 = np.random.randint(0, 10, size=10)
print(listnp4)
```

```
[8 8 3 7 7 0 4 2 5 2]
```

In [45]:

```
# concatenating columnwise
a = np.zeros([4, 4])
b = np.ones([4, 4])
# Vertical Stack Equivalents (Row wise)
print("using concatenate", np.concatenate([a, b], axis=0))
print("using vstack",np.vstack([a,b]))
np.r_[a,b]  # concatenation along the first axis.
```

```
using concatenate [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
using vstack [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Out[45]:

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [46]:

```
# Coliumn wise stacking
print("using concatenate",np.concatenate([a, b], axis=1))
print("using vstack", np.hstack([a,b]))
np.c_[a,b]
```

```
using concatenate [[0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1.]]
using vstack [[0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1.]]
```

Out[46]:

```
array([[0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 1., 1., 1., 1.]])
```

In [47]:

```
# Import data from csv file (url)
path = 'https://raw.githubusercontent.com/selva86/datasets/master/Auto.csv'
data = np.genfromtxt(path, delimiter=',', dtype='f, d, f, f, f, f, d, d, U50', r
print("Headers:",data.dtype.names)
data[:3]  # see first 3 rows
```

```
Headers: ('mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
'acceleration', 'year', 'origin', 'name')
```

Out[47]:

```
array([(18., 8., 307., 130., 3504., 12. , 70., 1., '"chevrolet chevell
e malibu"'),
       (15., 8., 350., 165., 3693., 11.5, 70., 1., '"buick skylark 32
0"'),
       (18., 8., 318., 150., 3436., 11. , 70., 1., '"plymouth satellit
e"')],
      dtype=[('mpg', '<f4'), ('cylinders', '<f8'), ('displacement', '<
f4'), ('horsepower', '<f4'), ('weight', '<f4'), ('acceleration', '<f
4'), ('year', '<f8'), ('origin', '<f8'), ('name', '<U50')])
```

In [48]:

```
# Save the array as a csv file
with open('data/Auto_out.csv', 'wb') as f:
    np.savetxt(f, data, delimiter=",", header=','.join(data.dtype.names), fmt='%
```

Text to vector - one-hot encoding

One hot encoding convert categorical data to a numeric value. For example, if you have the category as "cat dog mouse" This can be represented as `[1 0 0] [0 1 0] [0 0 1]` where the first index is for cat and the last for mouse.

In [49]:

```
# change texts to a one hot encoding
sent = "Can I eat the Pizza? said the man".lower()
words = set(sent.split())
word_indexs = {}
for i, word in enumerate(words):
        word_indexs[word] = i
print(word_indexs)
word_vectors = [word_indexs[word] for word in sent.split()]
one_hot_encodings = np.eye(len(words))[word_vectors]
print(one_hot_encodings)
```

```
{'said': 0, 'pizza?': 1, 'i': 2, 'man': 3, 'eat': 4, 'the': 5, 'can':
6}
[[0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 0. 0.]]
```

In [50]:

```
# compute sentence similarity
sent1 = "I love Pizza" # [1 1 1 0] vocabs = I, love, pizza, hate , Here hate is
sent2 = "I hate Pizza" # [1 0 1 1 1]
vec1 = np.array([1, 1, 1, 0])
vec2 = np.array([1, 0, 1, 1])
print("similarity=", np.round(np.dot(vec1,vec2) / (np.sqrt(np.dot(vec1,vec1)) +
```

```
similarity= 0.58
```

Excercise 2 (6 pts)

1. Write a program that will produce one-hot encoding for sentences in a large corpus (use the NLTK books corpus). Before computing the one-hot encoding of sentences 1) remove stop words (you can use the builtin stopwords list in NLTK), and 2) compute the total number of unique words that is used to compute the size of the one-hot encoding vectors.
2. Using the one-hot encoding, write a program that is used to find the most similar sentence for a given input sentence. How do you handle out-of-vocabulary words in the input sentences, i.e if the input sentence contains words that do not occur while training the one-hot encoding?

Reading and plotting using pandas and Seaborn

In [51]:

```
# Read the auto_csv file we wrote to the file system earlier
import pandas as pd
df = pd.read_csv("data/Auto_out.csv")
df
```

Out[51]:

	# mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8.0	307.0	130.0	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
1	15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0	buick skylark 320
2	18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0	plymouth satellite

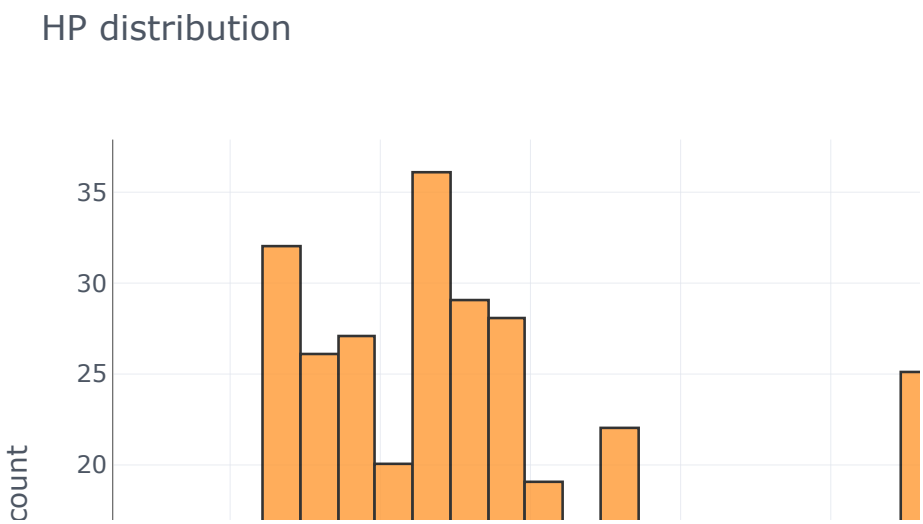
In [52]:

```
!pip install cufflinks --upgrade
```

```
Requirement already satisfied: cufflinks in /Users/seidmuhieyimam/op
t/miniconda3/lib/python3.7/site-packages (0.17.3)
Requirement already satisfied: colorlover>=0.2.1 in /Users/seidmuhiey
imam/opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (0.
3.0)
Requirement already satisfied: ipython>=5.3.0 in /Users/seidmuhieyima
m/opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (7.31.
1)
Requirement already satisfied: numpy>=1.9.2 in /Users/seidmuhieyimam/
opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (1.21.3)
Requirement already satisfied: ipywidgets>=7.0.0 in /Users/seidmuhiey
imam/opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (7.
6.5)
Requirement already satisfied: pandas>=0.19.2 in /Users/seidmuhieyima
m/opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (1.3.4)
Requirement already satisfied: plotly>=4.1.1 in /Users/seidmuhieyima
m/opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (5.3.1)
Requirement already satisfied: setuptools>=34.4.1 in /Users/seidmuhie
yimam/opt/miniconda3/lib/python3.7/site-packages (from cufflinks) (5
```

In [53]:

```
# A productivity tool that binds pandas and plotly.
import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
df['horsepower'].iplot(
    kind='hist',
    bins=50,
    xTitle='HP',
    linecolor='black',
    yTitle='count',
    title='HP distribution')
```



In [54]:

```
1 !pip install seaborn
```

```
Requirement already satisfied: seaborn in /Users/seidmuhieyimam/opt/mi
niconda3/lib/python3.7/site-packages (0.11.2)
Requirement already satisfied: pandas>=0.23 in /Users/seidmuhieyimam/o
pt/miniconda3/lib/python3.7/site-packages (from seaborn) (1.3.4)
Requirement already satisfied: numpy>=1.15 in /Users/seidmuhieyimam/op
t/miniconda3/lib/python3.7/site-packages (from seaborn) (1.21.3)
Requirement already satisfied: scipy>=1.0 in /Users/seidmuhieyimam/op
t/miniconda3/lib/python3.7/site-packages (from seaborn) (1.7.1)
Requirement already satisfied: matplotlib>=2.2 in /Users/seidmuhieyima
m/opt/miniconda3/lib/python3.7/site-packages (from seaborn) (3.4.3)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/seidmuhieyim
am/opt/miniconda3/lib/python3.7/site-packages (from matplotlib>=2.2->s
eaborn) (3.0.7)
Requirement already satisfied: cycler>=0.10 in /Users/seidmuhieyimam/o
pt/miniconda3/lib/python3.7/site-packages (from matplotlib>=2.2->seabo
rn) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/seidmuhieyi
mam/opt/miniconda3/lib/python3.7/site-packages (from matplotlib>=2.2->
seaborn) (1.3.2)
Requirement already satisfied: pillow>=6.2.0 in /Users/seidmuhieyimam/
opt/miniconda3/lib/python3.7/site-packages (from matplotlib>=2.2->seab
orn) (8.4.0)
Requirement already satisfied: python-dateutil>=2.7 in /Users/seidmuhi
eyimam/opt/miniconda3/lib/python3.7/site-packages (from matplotlib>=2.
2->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /Users/seidmuhieyimam/o
pt/miniconda3/lib/python3.7/site-packages (from pandas>=0.23->seaborn)
(2021.3)
Requirement already satisfied: six>=1.5 in /Users/seidmuhieyimam/opt/m
iniconda3/lib/python3.7/site-packages (from python-dateutil>=2.7->matp
lotlib>=2.2->seaborn) (1.16.0)
WARNING: You are using pip version 22.0.3; however, version 22.3 is av
ailable.
You should consider upgrading via the '/Users/seidmuhieyimam/opt/minic
onda3/bin/python -m pip install --upgrade pip' command.
```

Vizualization of the Predicting Upvotes

Dataset

An online question and answer platform has hired you as a data scientist to identify the best question authors on the platform. This identification will bring more insight into increasing the user engagement. Given the tag of the question, number of views received, number of answers, username and reputation of the question author, the problem requires you to predict the upvote count that the question will receive.

See details here (https://github.com/lawrence2269/Upvotes)

Variable	Definition
ID	Question ID
Tag	Anonymised tags representing question category
Reputation	Reputation score of question author
Answers	Number of times question has been answered
Username	Anonymised user id of question author

Variable	Definition
Views	Number of times question has been viewed
Upvotes	(Target) Number of upvotes for the question

In [55]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from scipy import stats
```

In [56]:

```
# Use the 'Predict Number of Upvotes' dataset:
# https://datahack.analyticsvidhya.com/contest/enigma-codefest-machine-learning-
#It deals with problem of predicting the upvote count for a queries posted and i
# the parameters that affect it the most.
df = pd.read_csv("data/train_upvotes.csv")
df.head()
```

Out[56]:

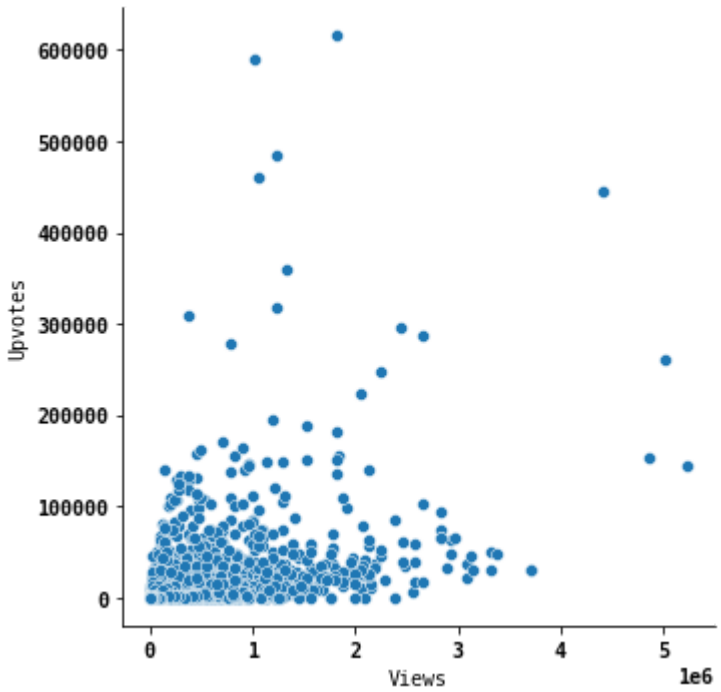
	ID	Tag	Reputation	Answers	Username	Views	Upvot
0	52664	a	3942.0	2.0	155623	7855.0	42.
1	327662	a	26046.0	12.0	21781	55801.0	1175.
2	468453	c	1358.0	4.0	56177	8067.0	60.
3	96996	a	264.0	3.0	168793	27064.0	9.
4	131465	c	4271.0	4.0	112223	13986.0	83.

In [57]:

```
#scatter plot to show relationship between views and upvotes
sns.relplot(x="Views", y="Upvotes", data = df)
```

Out[57]:

<seaborn.axisgrid.FacetGrid at 0x7fb03dda17f0>

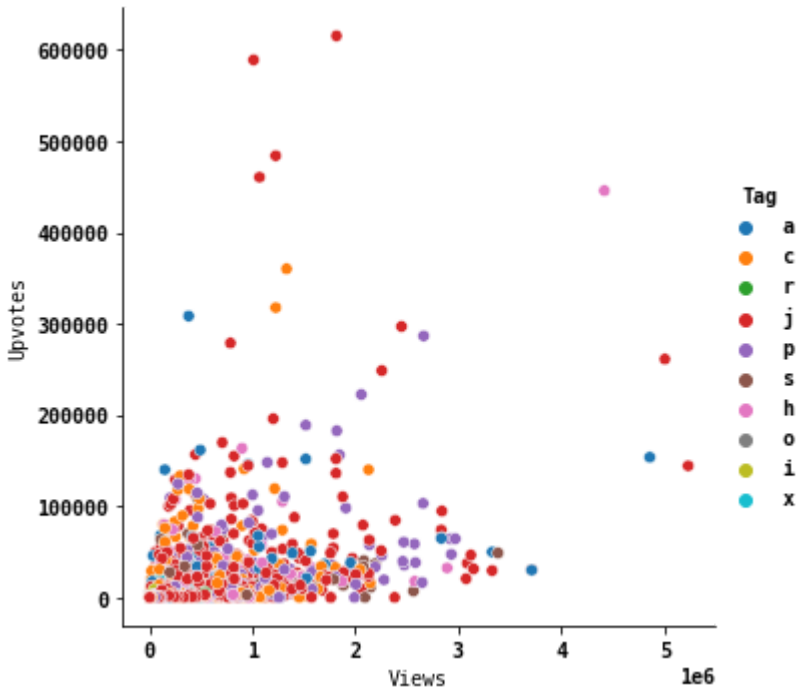


In [58]:

```
# See the tag associated with the data with a color attached to the points
sns.relplot(x="Views", y="Upvotes", hue = "Tag", data = df)

```

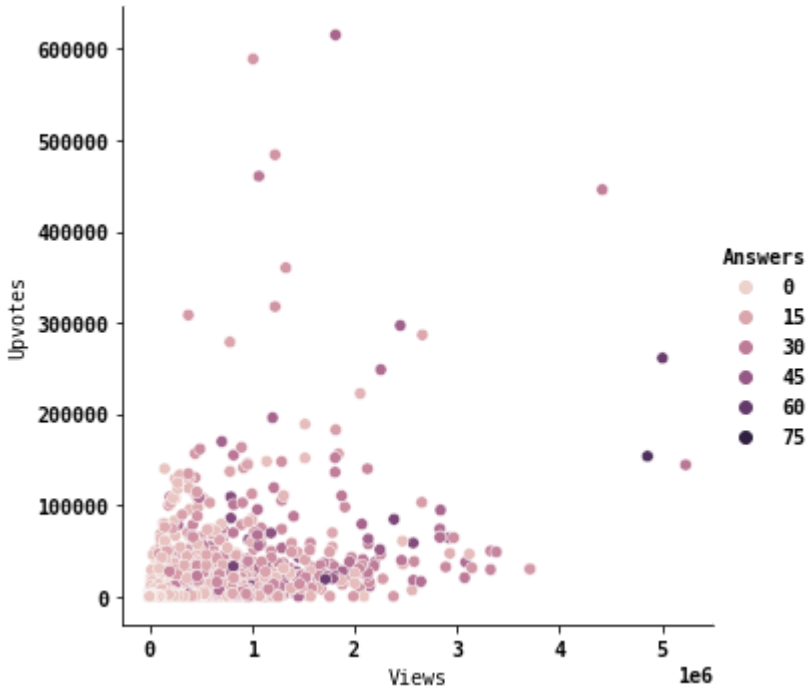
Out[58]:

<seaborn.axisgrid.FacetGrid at 0x7fb06f596df0>



In [59]:

```
1 sns.relplot(x="Views", y="Upvotes", hue = "Answers", data = df);
```



In [60]:

```
# Food servers's tips in a restuarant, which factor is important?..
tips = sns.load_dataset("tips")
tips
```

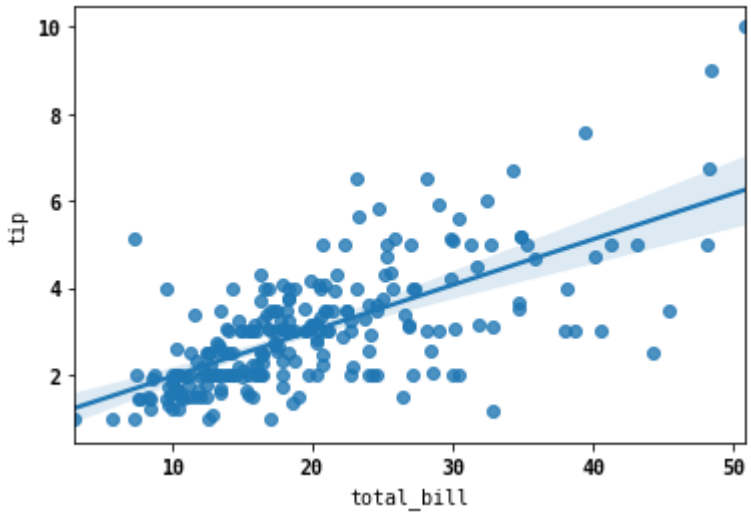
Out[60]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

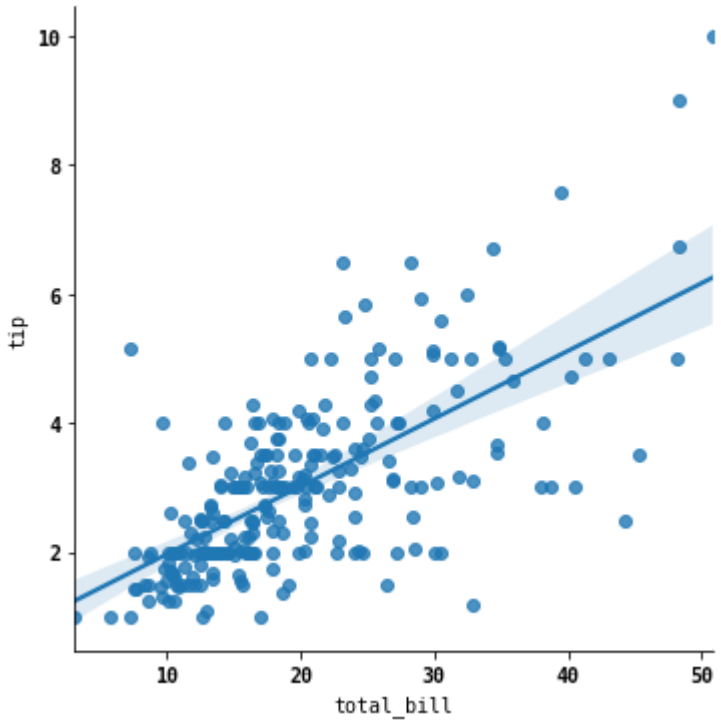
In [61]:

```
#visualize a linear relationship as determined through regression using the regp
sns.regplot(x="total_bill", y="tip", data=tips);
```



In [62]:

```
1 sns.lmplot(x="total_bill", y="tip", data=tips);
```

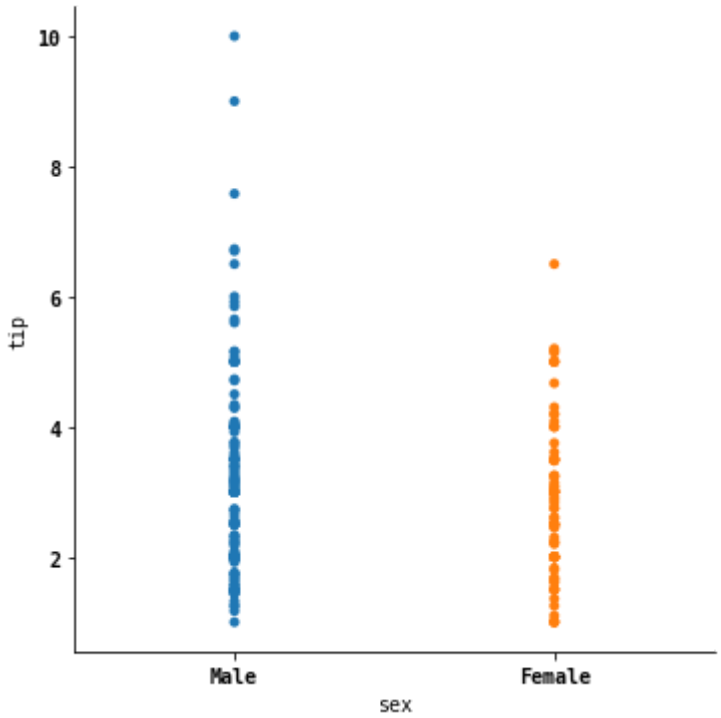


In [63]:

```
1 # Categorical onto a FaceGrid
2 sns.catplot(x="sex", y="tip", jitter = False, data=tips)
```

Out[63]:

```
<seaborn.axisgrid.FacetGrid at 0x7fb03ffe4d90>
```

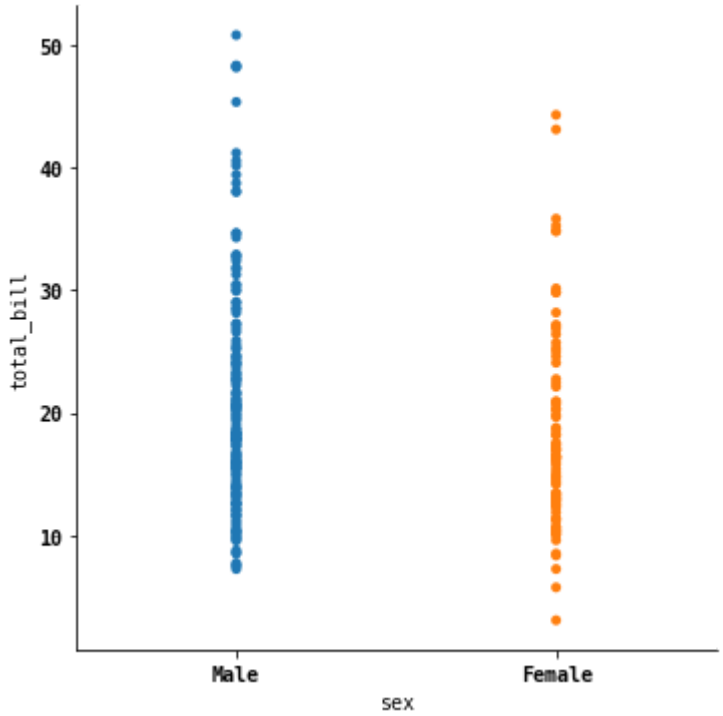


In [64]:

```
1 sns.catplot(x="sex", y="total_bill", jitter = False, data=tips)
```

Out[64]:

<seaborn.axisgrid.FacetGrid at 0x7fb03ffec040>

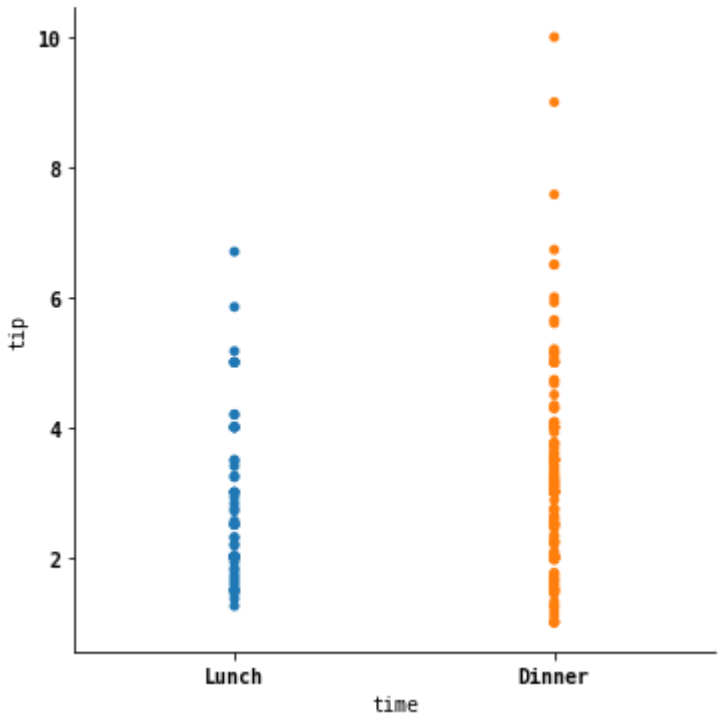


In [65]:

```
1 sns.catplot(x="time", y="tip", jitter = False, data=tips)
```

Out[65]:

<seaborn.axisgrid.FacetGrid at 0x7fb06d9bed90>

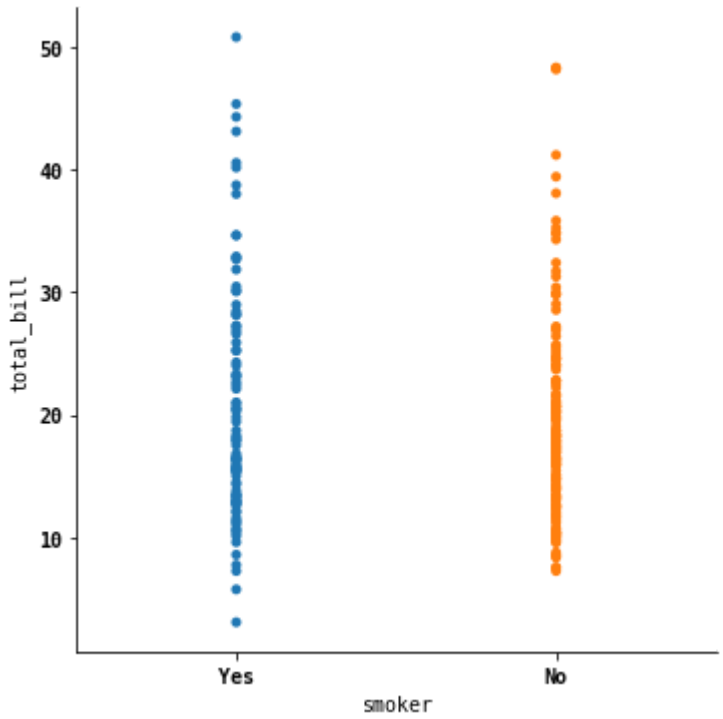


In [66]:

```
1 sns.catplot(x="smoker", y="total_bill", jitter = False, data=tips)
```

Out[66]:

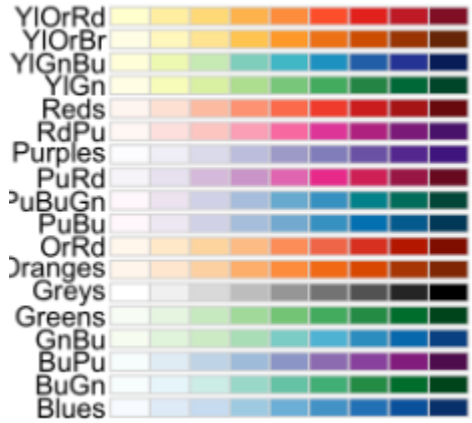
```
<seaborn.axisgrid.FacetGrid at 0x7fb08b92edc0>
```



Heat Map

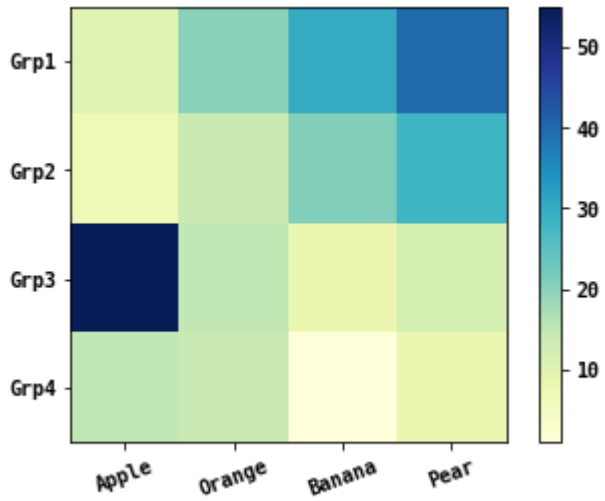
A heat map (or heatmap) is a graphical representation of data where the individual values contained in a matrix are represented as colors.

You can use sequential color palettes which are suited to ordered data that progress from low to high (gradient). The palettes names are : Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu YlOrBr, YlOrRd.



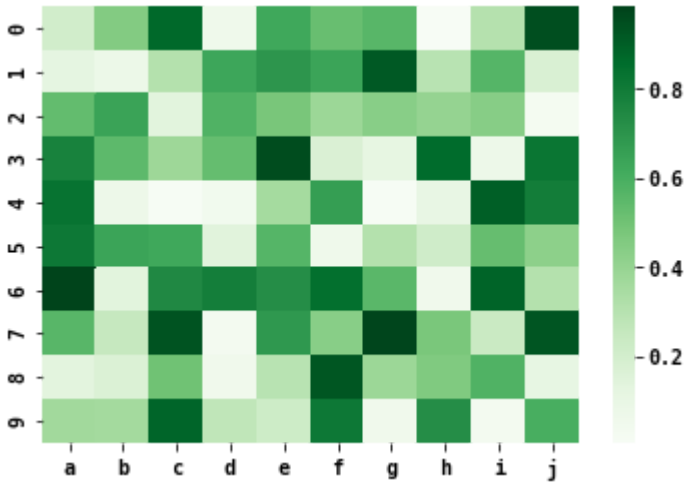
In [67]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
df = pd.DataFrame([[10, 20, 30, 40], [7, 14, 21, 28], [55, 15, 8, 12],
                   [15, 14, 1, 8]],
                  columns=['Apple', 'Orange', 'Banana', 'Pear'],
                  index=['Grp1', 'Grp2', 'Grp3', 'Grp4']
                  )
plt.imshow(df, cmap="YlGnBu")
plt.colorbar()
plt.xticks(range(len(df)),df.columns, rotation=20)
plt.yticks(range(len(df)),df.index)
plt.show()
```



In [68]:

```
import seaborn as sns
import pandas as pd
import numpy as np

# Create a sample dataset
df = pd.DataFrame(np.random.random((10,10)), columns=["a","b","c","d","e","f","g
# plot using a color palette
#chose one of them below
#sns.heatmap(df, cmap="YlGnBu")
#sns.heatmap(df, cmap="Blues")
#sns.heatmap(df, cmap="BuPu")
sns.heatmap(df, cmap="Greens")

#add this after your favorite color to show the plot
plt.show()
```



Resources

- Text concordance (https://orange3-text.readthedocs.io/en/latest/widgets/concordance

- Data Analysis and Visualization for Text Data (https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a)
- Heatmap Seaborn (https://likegeeks.com/seaborn-heatmap-tutorial/)
- Heatmap Seaborn 2 (https://stackabuse.com/seaborn-library-for-data-visualization-in-python-part-2/)
- Word cloud (https://www.datacamp.com/community/tutorials/w python)
- Word Cloud 2 (https://www.datacamp.com/community/tutorials/w python)
- Word Frequencies (https://programminghistorian.org/en/lessons/count frequencies)

- Seaborn (https://www.analyticsvidhya.com/blog/2019/09/co data-visualization-guide-seaborn-python/)
- seaborn tutorial (https://www.analyticsvidhya.com/blog/2019/09/co data-visualization-guide-seaborn-python/)

In [ ]:

```
1
```