Sprint 9. Anàlisi de sentiment i textos

Una gran part de les dades que podem trobar per analitzar utilitzant l'aprenentatge automàtic és text lliure. En aquest sprint veurem com analitzar-lo, netejar-lo i realitzar anàlisi de sentiments.

```
!pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from nltk) (1.1.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.7/dist-packages (from nltk) (2022.6.2)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from nltk) (4.64.1)
```

- Exercici 1

Agafa un text en anglès que vulguis, i calcula'n la freqüència de les paraules.

```
#Loading NLTK
import nltk
```

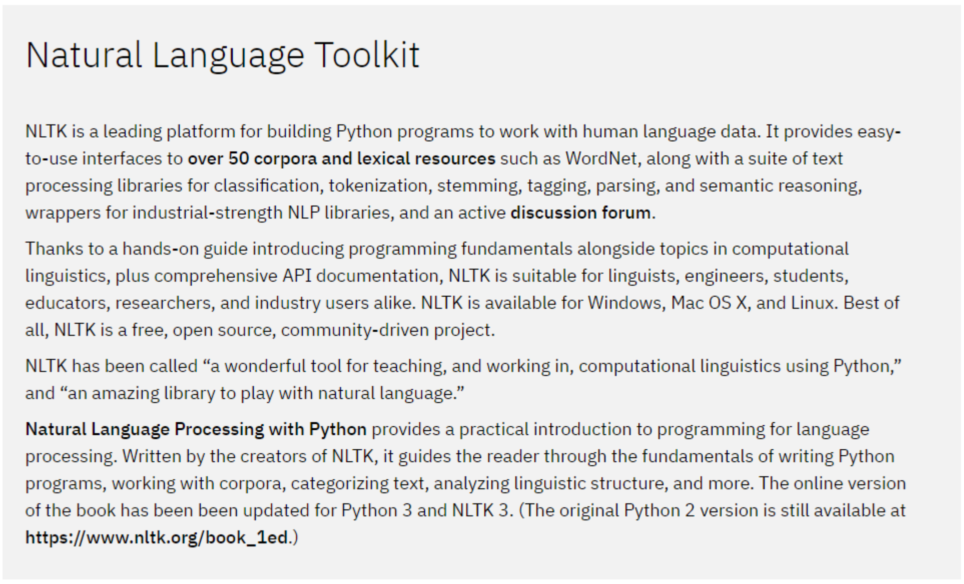
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

El texto sale de aquí:

https://www.nltk.org/

Imagen:



Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to **over 50 corpora and lexical resources** such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active **discussion forum**.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3. (The original Python 2 version is still available at **https://www.nltk.org/book_1ed**.)

```
from nltk.tokenize import sent_tokenize

text="""Natural Language Toolkit
NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation,

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with r

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, i

tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Natural Language Toolkit\nNLTK is a leading platform for building Python programs to work with human language data.', 'It provides easy-to-use interfaces to over 50 corpora anc
```

```
#Identifico los tokems:

from nltk.tokenize import word_tokenize

tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Natural', 'Language', 'Toolkit', 'NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python', 'programs', 'to', 'work', 'with', 'human', 'language', 'data', '.', 'It
```

```
# Analizo la frecuencia con la que salen los tokens
```

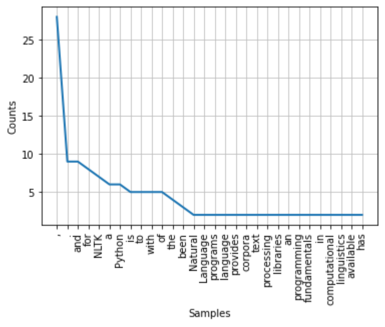
```

from nltk.probability import FreqDist
fdist = FreqDist(tokenized_word)
print(fdist)
```

```
<FreqDist with 137 samples and 247 outcomes>
```

```
fdist.most_common(5)
```

```
[(',', 28), ('.', 9), ('and', 9), ('for', 8), ('NLTK', 7)]
```

```
# Frequency Distribution Plot
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```



Vemos que las palabras que más salen no son las palabras más i mportantes porque la 'coma' o la conjunción 'and' no nos aportarán mucha información

- Exercici 2

Treu les stopwords i realitza stemming al teu conjunt de dades.

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
# buscamos las stopwords:

from nltk.corpus import stopwords
#nltk.download('stopwords')

stop_words=set(stopwords.words("english"))
print(stop_words)
print('Total cantidad de stopwords indentificadas: ', len(stop_words))
```

```
{'about', 'while', 'can', 'yourself', 'doing', 'who', 'are', "you'd", 'hadn', 'won', "shan't", "shouldn't", 'each', 'theirs', "wasn't", 'itself', "you've", 'there', 'me', 'any',
Total cantidad de stopwords indentificadas:  179
```

tokenized_text=sent_tokenize(text)

```
tokenized_sent = tokenized_word

filtered_sent=[]
for w in tokenized_sent:
    #print(w)
    if w not in stop_words:
        filtered_sent.append(w)
print("frases Tokenized:",tokenized_sent)
print("Tolkens filtrados: ",filtered_sent)
```

```
frases Tokenized: ['Natural', 'Language', 'Toolkit', 'NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python', 'programs', 'to', 'work', 'with', 'human', 'language'
Tolkens filtrados:  ['Natural', 'Language', 'Toolkit', 'NLTK', 'leading', 'platform', 'building', 'Python', 'programs', 'work', 'human', 'language', 'data', '.', 'It', 'provides
```

```
# Stemming

from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()

stemmed_words=[]
for w in filtered_sent:
    stemmed_words.append(ps.stem(w))

print("Filtered Sentence:",filtered_sent)
print("Stemmed Sentence:",stemmed_words)
```

```
Filtered Sentence: ['Natural', 'Language', 'Toolkit', 'NLTK', 'leading', 'platform', 'building', 'Python', 'programs', 'work', 'human', 'language', 'data', '.', 'It', 'provides'
Stemmed Sentence: ['natur', 'languag', 'toolkit', 'nltk', 'lead', 'platform', 'build', 'python', 'program', 'work', 'human', 'languag', 'data', '.', 'it', 'provid', 'easy-to-us'
```

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
True
```

```
#Lexicon Normalization

#Nos vamos a  aquedar con la parte de la palabra que realmente aporta información

from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()


print('\n Ejemplo de lematización:\n')
word = "fly"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:   ",stem.stem(word))
```

```
 Ejemplo de lematización:

Lemmatized Word: fly
Stemmed Word:    fli
```

- Exercici 2, continuación.

Treu les stopwords i realitza stemming al teu conjunt de dades.

Para entender el funcionameinto, tomaremos esta frase ***Albert Einstein was born in Ulm, Germany in 1879.*** e identificaremos los elementos importantes de la frase.

```
sent = "Albert Einstein was born in Ulm, Germany in 1879."

tokens=nltk.word_tokenize(sent)
print('\n Los tokens son: \n')
print(tokens)
```

```
 Los tokens son:

['Albert', 'Einstein', 'was', 'born', 'in', 'Ulm', ',', 'Germany', 'in', '1879', '.']
```

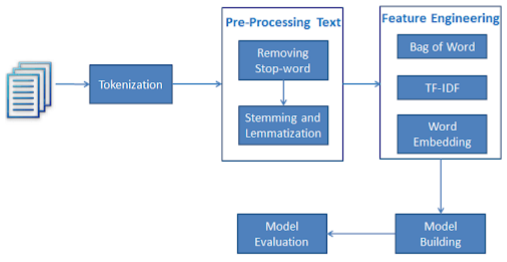
```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
# Cada elemento de la palabra identifico que función hace dentro de la oración:

nltk.pos_tag(tokens)
```

```
[('Albert', 'NNP'),
 ('Einstein', 'NNP'),
 ('was', 'VBD'),
 ('born', 'VBN'),
 ('in', 'IN'),
 ('Ulm', 'NNP'),
 (',', ','),
 ('Germany', 'NNP'),
 ('in', 'IN'),
 ('1879', 'CD'),
 ('.', '.')]
```

El proceso que seguiremos es el siguiente:



Ahora que con una frase vemos como funciona vamos a trabajar con toda la bases de datos:

```
# Activo Google Drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Import pandas
import pandas as pd


data=pd.read_csv('/content/drive/MyDrive/01_COLAB/train.tsv', sep='\t')
```

```
data.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

```
# Cojo una sola línea:

data.iloc[0][2]
```

```
'A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much
of a story . '
```

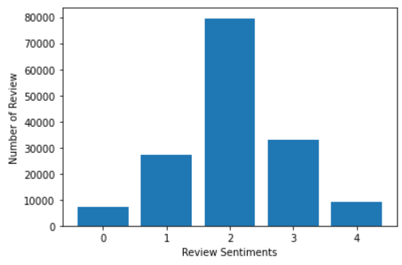
```
# Identifico que estructura tiene el archivo:

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156060 entries, 0 to 156059
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   PhraseId    156060 non-null  int64 
 1   SentenceId  156060 non-null  int64 
 2   Phrase      156060 non-null  object
 3   Sentiment   156060 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 4.8+ MB
```

```
# En este fichero la parte importante es la columna sentimiento.
# Voy a ver que cantidad de puntuacion hay para cada valor.

data.Sentiment.value_counts()
```

```
2    79582
3    32927
1    27273
4     9206
0     7072
Name: Sentiment, dtype: int64
```

```
# Mirar las cantidades aorta poco
# Mejor graficarlo

Sentiment_count=data.groupby('Sentiment').count()
plt.bar(Sentiment_count.index.values, Sentiment_count['Phrase'])
plt.xlabel('Review Sentiments')
plt.ylabel('Number of Review')
plt.show()
```



Vemos que la mayor valoración es 2 que es una puntuación media y los extremos la cantidad de veces que lo han valorado es poco.

Esto nos podría hacer obligar a forzar a trabajar con los datos de los extremos y una solución es eliminar las frases que su puntuación sea 2

```
# Ahora vamos a vectorizar las palabras

from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import RegexpTokenizer
#tokenizer to remove unwanted elements from out data like symbols and numbers
token = RegexpTokenizer(r'[a-zA-Z0-9]+')
cv = CountVectorizer(lowercase=True,stop_words='english',ngram_range = (1,1),tokenizer = token.tokenize)
text_counts= cv.fit_transform(data['Phrase'])
```

```
print(text_counts)
```

```
  (0, 11671)	1
  (0, 4517)	1
  (0, 3444)	1
  (0, 294)	1
  (0, 5735)	2
  (0, 5751)	1
  (0, 5512)	1
  (0, 9065)	1
  (0, 593)	1
  (0, 584)	1
  (0, 12673)	1
  (1, 11671)	1
  (1, 4517)	1
  (1, 3444)	1
  (1, 294)	1
  (1, 5735)	1
  (1, 5751)	1
  (2, 11671)	1
  (4, 11671)	1
  (5, 4517)	1
  (5, 3444)	1
  (5, 294)	1
  (5, 5735)	1
  (5, 5751)	1
  (7, 4517)	1
  :	:
  (156050, 11305)	1
  (156050, 9054)	1
  (156051, 11305)	1
  (156051, 9054)	1
  (156052, 11305)	1
  (156053, 11281)	1
  (156053, 1281)	1
  (156053, 5252)	1
  (156053, 6156)	1
  (156053, 1006)	1
  (156053, 2271)	1
  (156054, 11281)	1
  (156054, 5252)	1
  (156054, 6156)	1
  (156054, 1006)	1
  (156054, 2271)	1
  (156055, 11281)	1
  (156055, 6156)	1
  (156056, 5252)	1
  (156056, 1006)	1
  (156056, 2271)	1
  (156057, 1006)	1
  (156057, 2271)	1
  (156058, 1006)	1
  (156059, 2271)	1
```

```python
# Creamos un modelo y para ello vamos a separar las frases de test y de Train

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    text_counts, data['Sentiment'], test_size=0.3, random_state=1)
```

```python
from sklearn.naive_bayes import MultinomialNB
#Importamos el módulo scikit-learn para el cálculo de la precisión

from sklearn import metrics
# Generamos el modeloo utilizando el Naive Bayes multinomial

clf = MultinomialNB().fit(X_train, y_train)
predicted= clf.predict(X_test)


print("MultinomialNB Accuracy: ",round(metrics.accuracy_score(y_test, predicted),2))
```

```
MultinomialNB Accuracy:  0.6
```

Nos ha dado un valor de de Accuracy de 0.59 que yo lo consideraría un poco bajo

```python
# Aplicamos el modelo Naive Bayes multinomial

from sklearn.feature_extraction.text import TfidfVectorizer
tf=TfidfVectorizer()
text_tf= tf.fit_transform(data['Phrase'])
```

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    text_tf, data['Sentiment'], test_size=0.3, random_state=123)
```

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
# Generamos el modeloo utilizando el Naive Bayes multinomial

clf = MultinomialNB().fit(X_train, y_train)
predicted= clf.predict(X_test)
print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted))
```

```
MultinomialNB Accuracy: 0.5865265496176684
```

```python
import nltk
sentence = """At eight o'clock on Thursday morning Arthur didn't feel very good."""

tokens = nltk.word_tokenize(sentence)
tokens
tagged = nltk.pos_tag(tokens)
tagged[0:6]
```

```
[('At', 'IN'),
 ('eight', 'CD'),
 ("o'clock", 'NN'),
 ('on', 'IN'),
 ('Thursday', 'NNP'),
 ('morning', 'NN')]
```

```python
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
True
```

```
entities = nltk.chunk.ne_chunk(tagged)
#entities
```

```
nltk.download('treebank')
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
True
```

Tomo como documentación el siguiente video https://www.youtube.com/watch?v=DxnXVbHGeBg

- Exercici 3

Realitza sentiment analysis al teu conjunt de dades.

Utilizaré el modulo **SentimentIntensityAnalyzer** , que es un modelo ya entrenado, que me dará un score con el porcentaje de palabras positivas, negativas y neutras.

Lo aplicaré al texto que he excogido anteriormente en el ejercicio 1 y veré que resultado me da.

SentimentIntensityAnalyzer() de VADER, toma una cadena y devuelve un diccionario de puntajes en cada una de las cuatro categorías:

- negativo
- neutral
- positivo
- compuesto (calculado mediante la normalización de las puntuaciones anteriores)

```
import nltk
nltk.download('vader_lexicon')
nltk.download('punkt')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Haz doble clic (o pulsa Intro) para editar

```

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
sentences = tokenizer.tokenize(text)
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk import sentiment
from nltk import word_tokenize
```

```
analizador = SentimentIntensityAnalyzer()
```

```
# Listado de las frases
sentences
```

```
['Natural Language Toolkit\nNLTK is a leading platform for building Python programs to work with human language data.',
 'It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization,
stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.',
 'Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for
linguists, engineers, students, educators, researchers, and industry users alike.',
 'NLTK is available for Windows, Mac OS X, and Linux.',
 'Best of all, NLTK is a free, open source, community-driven project.',
 'NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural
language.”\n\nNatural Language Processing with Python provides a practical introduction to programming for language processing.',
 'Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic
structure, and more.',
 'The online version of the book has been been updated for Python 3 and NLTK 3.',
 '(The original Python 2 version is still available at https://www.nltk.org/book_1ed.)']
```

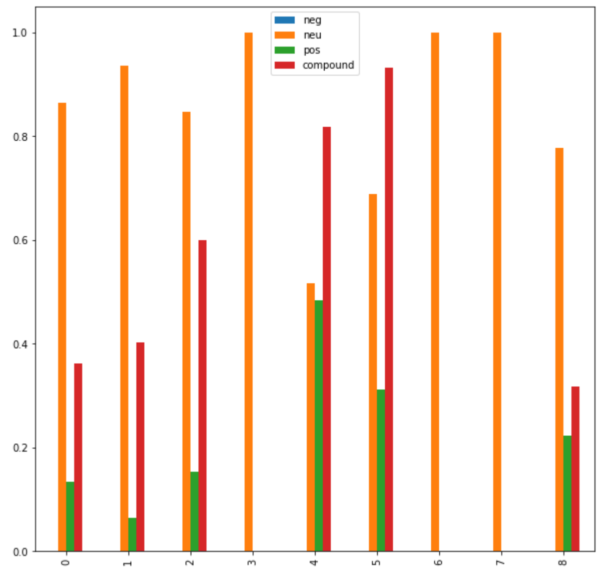
```
import statistics

valores=[]

neg=[]
neu=[]
pos=[]
compound=[]


for i, sentence in enumerate(sentences):
    #print(i, sentence)
    scores = analizador.polarity_scores(sentence)
    #print('i= ', i, scores, type(scores))
    neu.append(scores['neu'])
    neg.append(scores['neg'])
    compound.append(scores['compound'])
    pos.append(scores['pos'])
    valores.append(scores)

    #for key in scores:
        #print(key, ': ', scores[key])
        #print()

print('\n Valores: ')
valores[2]['neu']

print('neutros:', neu,'    -> ', round(statistics.mean(neu),2))
print('pos:    ', pos,'    -> ', round(statistics.mean(pos),2))
print('neg:    ', neg,'    -> ', round(statistics.mean(neg),2))
print('Compound:','    -> ', compound, round(statistics.mean(compound),2))
```

```
 Valores: 
neutros: [0.865, 0.935, 0.846, 1.0, 0.516, 0.689, 1.0, 1.0, 0.777]     ->  0.85
pos:     [0.135, 0.065, 0.154, 0.0, 0.484, 0.311, 0.0, 0.0, 0.223]     ->  0.15
neg:     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]     ->  0.0
Compound:     ->  [0.3612, 0.4019, 0.5994, 0.0, 0.8176, 0.9313, 0.0, 0.0, 0.3182] 0.38
```

```
valores
valores1=pd.DataFrame(valores)
valores2 = valores1.mean()
```

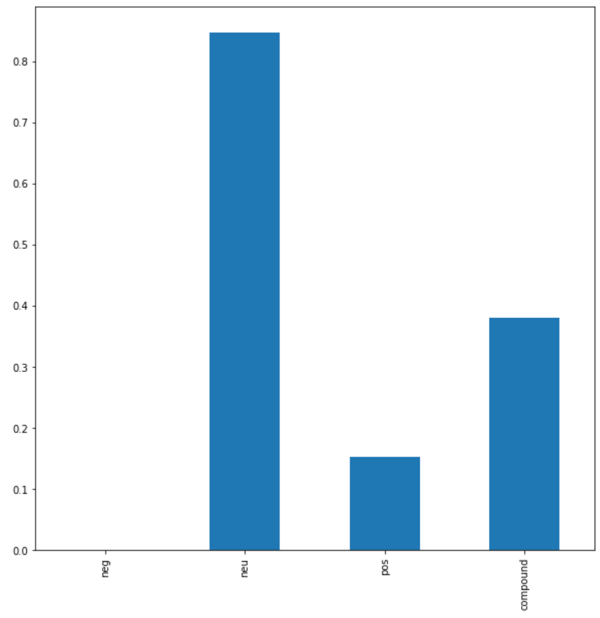
```
valores1.plot(kind = 'bar',figsize=(10,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f390803cb90>
```



```
valores2.plot(kind = 'bar',figsize=(10,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3907e5d690>
```



Conclusiones:

Vemos que utilizar NLTK no es complicado porque el kit de herramientas de lenguaje natural, es un conjunto de módulos de programas de código abierto que proporciona recursos de lingüística computacional para usar en la interpretción de textos.

NLTK cubre el procesamiento de lenguaje natural simbólico y estadístico. Realmente ha sido muy útil este modulo.

Productos de pago de Colab - Cancelar contratos

✓ 0 s completado a las 20:52