

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.

```
1 !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: click in /usr/local/lib/python3.7/dist-packages (from nltk) (7.1.2)
Requirement already satisfied: regex<=2021.8.3 in /usr/local/lib/python3.7/dist-packages (from nltk) (2022.6.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from nltk) (4.64.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from nltk) (1.1.0)
```

## ▶ - Exercici 1

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

```
1 #Loading NLTK
2 import nltk
```

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
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 updated for Python 3 and NLTK 3. (The original Python 2 version is still available at [https://www.nltk.org/book\\_1ed.](https://www.nltk.org/book_1ed.))

```
1 from nltk.tokenize import sent_tokenize
2
3 text="""Natural Language Toolkit
```

```

4 NLTK is a leading platform for building Python programs to work with human language data. It provides ea
5
6 Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational lingui
7
8 NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Pyt
9
10 Natural Language Processing with Python provides a practical introduction to programming for language pr
11
12 tokenized_text=sent_tokenize(text)
13 print(tokenized_text)

```

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

```

1 #Identifico los tokems:
2
3 from nltk.tokenize import word_tokenize
4
5 tokenized_word=word_tokenize(text)
6 print(tokenized_word)

```

```
['Natural', 'Language', 'Toolkit', 'NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python', 'programs', 'to',
```

```

1 # Analizo la frecuencia con la que salen los tokens
2
3 from nltk.probability import FreqDist
4 fdist = FreqDist(tokenized_word)
5 print(fdist)

```

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

```

1 fdist.most_common(5)

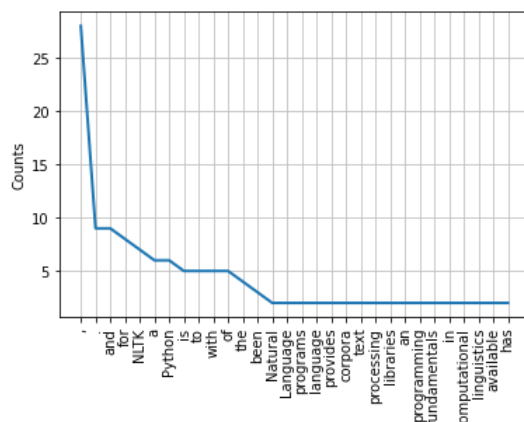
[(',', 28), ('.', 9), ('and', 9), ('for', 8), ('NLTK', 7)]

```

```

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

```



Vemos que las palabras que más salen no son las palabras más importantes 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.

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

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

```
1 # buscamos las stopwords:
2
3 from nltk.corpus import stopwords
4 #nltk.download('stopwords')
5
6 stop_words=set(stopwords.words("english"))
7 print(stop_words)
8 print('Total cantidad de stopwords indentificadas: ', len(stop_words))

{'them', "couldn't", 'itself', 'for', 'about', 'mightn', 'nor', "you've", 'are', 'd', 's', 'hasn', 'before', 'once', 'who',
Total cantidad de stopwords indentificadas: 179
```

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

```
1 tokenized_sent = tokenized_word
2
3 filtered_sent=[]
4 for w in tokenized_sent:
5     #print(w)
6     if w not in stop_words:
7         filtered_sent.append(w)
8 print("frases Tokenized:",tokenized_sent)
9 print("Tolkens filtrados: ",filtered_sent)

frases Tokenized: ['Natural', 'Language', 'Toolkit', 'NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python',
Tolkens filtrados: ['Natural', 'Language', 'Toolkit', 'NLTK', 'leading', 'platform', 'building', 'Python', 'programs', 'work
```

```
1 # Stemming
2
3 from nltk.stem import PorterStemmer
4 from nltk.tokenize import sent_tokenize, word_tokenize
5
6 ps = PorterStemmer()
7
8 stemmed_words=[]
9 for w in filtered_sent:
10     stemmed_words.append(ps.stem(w))
11
12 print("Filtered Sentence:",filtered_sent)
13 print("Stemmed Sentence:",stemmed_words)

Filtered Sentence: ['Natural', 'Language', 'Toolkit', 'NLTK', 'leading', 'platform', 'building', 'Python', 'programs', 'work
Stemmed Sentence: ['natur', 'languag', 'toolkit', 'nltk', 'lead', 'platform', 'build', 'python', 'program', 'work', 'human',
```

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

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

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

[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Package omw-1.4 is already up-to-date!
True
```

```
1 #Lexicon Normalization
2
3 #Nos vamos a quedar con la parte de la palabra que realmente aporta información
4
5 from nltk.stem.wordnet import WordNetLemmatizer
```

```

6 lem = WordNetLemmatizer()
7
8 from nltk.stem.porter import PorterStemmer
9 stem = PorterStemmer()
10
11
12 print('\n Ejemplo de lematización:\n')
13 word = "fly"
14 print("Lemmatized Word:",lem.lemmatize(word,"v"))
15 print("Stemmed Word:   ",stem.stem(word))

```

Ejemplo de lematización:

```

Lemmatized Word: fly
Stemmed Word:   fli

```

## ▼ - Exercici 2, continuació.

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

---

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

```

1 sent = "Albert Einstein was born in Ulm, Germany in 1879."
2
3 tokens=nltk.word_tokenize(sent)
4 print('\n Los tokens son: \n')
5 print(tokens)

```

Los tokens son:

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

```

1 nltk.download('averaged_perceptron_tagger')

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

```

```

1 # Cada elemento de la palabra identifico que función hace dentro de la oración:
2
3 nltk.pos_tag(tokens)

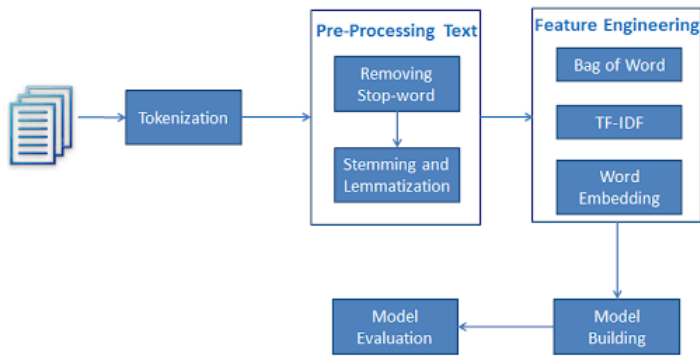
```

```

[('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:

```

1 # Activo Google Drive
2
3 from google.colab import drive
4 drive.mount('/content/drive')

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True)

```

1 # Import pandas
2 import pandas as pd
3
4
5 data=pd.read_csv('/content/drive/MyDrive/01_COLAB/train.tsv', sep='\t')

```

```
1 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

```

1 # Cojo una sola línea:
2
3 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

```

1 # Identifico que estructura tiene el archivo:
2
3 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

```

```

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

```

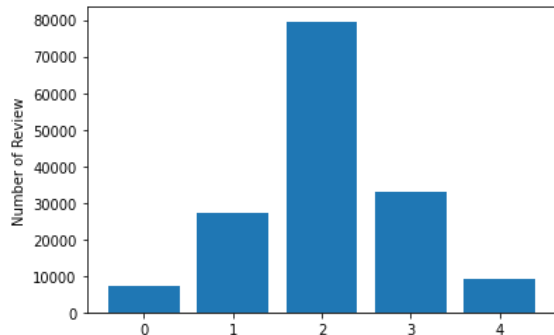
```

3
4 data.Sentiment.value_counts()

2    79582
3    32927
1    27273
4     9206
0     7072

1 # Mirar las cantidades aorta poco
2 # Mejor graficarlo
3
4 Sentiment_count=data.groupby('Sentiment').count()
5 plt.bar(Sentiment_count.index.values, Sentiment_count['Phrase'])
6 plt.xlabel('Review Sentiments')
7 plt.ylabel('Number of Review')
8 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

```

1 # Ahora vamos a vectorizar las palabras
2
3 from sklearn.feature_extraction.text import CountVectorizer
4 from nltk.tokenize import RegexpTokenizer
5 #tokenizer to remove unwanted elements from out data like symbols and numbers
6 token = RegexpTokenizer(r'[a-zA-Z0-9]+')
7 cv = CountVectorizer(lowercase=True, stop_words='english', ngram_range = (1,1), tokenizer = token.tokenize)
8 text_counts= cv.fit_transform(data['Phrase'])

```

```

1 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
```

```
1 # Creamos un modelo y para ello vamos a separar las frases de test y de Train
2
3 from sklearn.model_selection import train_test_split
4 X_train, X_test, y_train, y_test = train_test_split(
5     text_counts, data['Sentiment'], test_size=0.3, random_state=1)

1 from sklearn.naive_bayes import MultinomialNB
2 #Importamos el módulo scikit-learn para el cálculo de la precisión
3
4 from sklearn import metrics
5 # Generamos el modelo utilizando el Naive Bayes multinomial
6
7 clf = MultinomialNB().fit(X_train, y_train)
8 predicted= clf.predict(X_test)
9
10
11 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

```
1 # Aplicamos el modelo Naive Bayes multinomial
2
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 tf=TfidfVectorizer()
5 text_tf= tf.fit_transform(data['Phrase'])

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

1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn import metrics
3 # Generamos el modelo utilizando el Naive Bayes multinomial
4
5 clf = MultinomialNB().fit(X_train, y_train)
6 predicted= clf.predict(X_test)
7 print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted))

MultinomialNB Accuracy: 0.5865265496176684

1 import nltk
2 sentence = """"At eight o'clock on Thursday morning Arthur didn't feel very good.""
3
```

```

4 tokens = nltk.word_tokenize(sentence)
5 tokens
6 tagged = nltk.pos_tag(tokens)
7 tagged[0:6]

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

1 nltk.download('maxent_ne_chunker')
2 nltk.download('words')

[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] /root/nltk_data...
[nltk_data] Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data] Package words is already up-to-date!
True

1 entities = nltk.chunk.ne_chunk(tagged)
2 #entities

1 nltk.download('treebank')

[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data] Package treebank is already up-to-date!
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)

```

1 import nltk
2 nltk.download('vader_lexicon')
3 nltk.download('punkt')

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

```

1
2 tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
3 sentences = tokenizer.tokenize(text)

1 from nltk.sentiment.vader import SentimentIntensityAnalyzer

```



```

2 from nltk import sentiment
3 from nltk import word_tokenize

1 analizador = SentimentIntensityAnalyzer()

1 # Listado de las frases
2 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 updated for Python 3 and NLTK 3.',
'(The original Python 2 version is still available at https://www.nltk.org/book\_1ed.)']

1 import statistics
2
3 valores=[]
4
5 neg=[]
6 neu=[]
7 pos=[]
8 compound=[]
9
10
11 for i, sentence in enumerate(sentences):
12     #print(i, sentence)
13     scores = analizador.polarity_scores(sentence)
14     #print('i= ', i, scores, type(scores))
15     neu.append(scores['neu'])
16     neg.append(scores['neg'])
17     compound.append(scores['compound'])
18     pos.append(scores['pos'])
19     valores.append(scores)
20
21     #for key in scores:
22         #print(key, ': ', scores[key])
23         #print()
24
25 print('\n Valores: ')
26 valores[2]['neu']
27
28 print('neutros:', neu, '    -> ', round(statistics.mean(neu),2))
29 print('pos:    ', pos, '    -> ', round(statistics.mean(pos),2))
30 print('neg:    ', neg, '    -> ', round(statistics.mean(neg),2))
31 print('Compound:', '    -> ', compound, round(statistics.mean(compound),2))

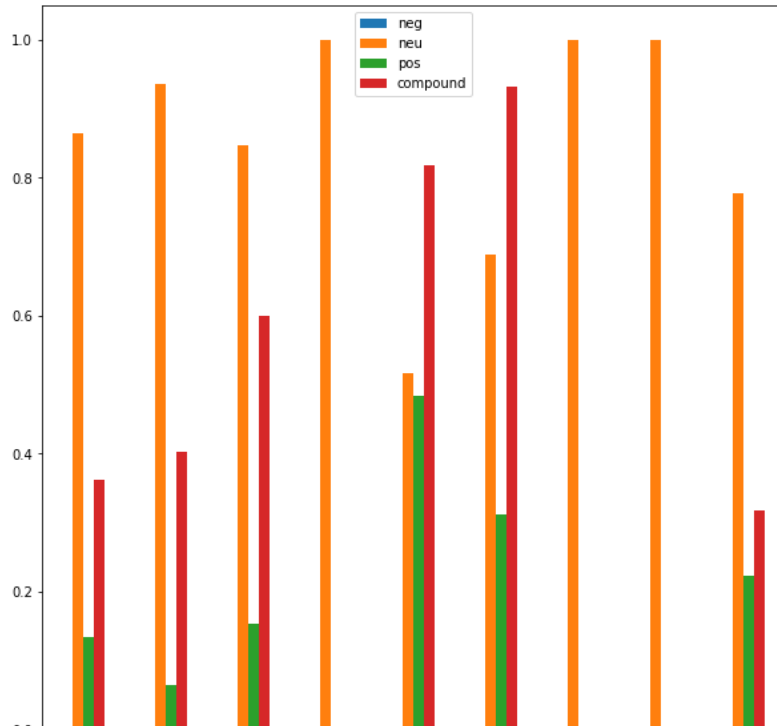
Valores:
neutros: [0.865, 0.935, 0.846, 1.0, 0.516, 0.689, 1.0, 1.0, 0.777]    -> 0.85
pos:    [0.135, 0.065, 0.154, 0.0, 0.484, 0.311, 0.0, 0.0, 0.223]    -> 0.15
neg:    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]    -> 0.0
Compound:    -> [0.3612, 0.4019, 0.5994, 0.0, 0.8176, 0.9313, 0.0, 0.0, 0.3182]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

1 valores
2 valores1=pd.DataFrame(valores)
3 valores2 = valores1.mean()

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

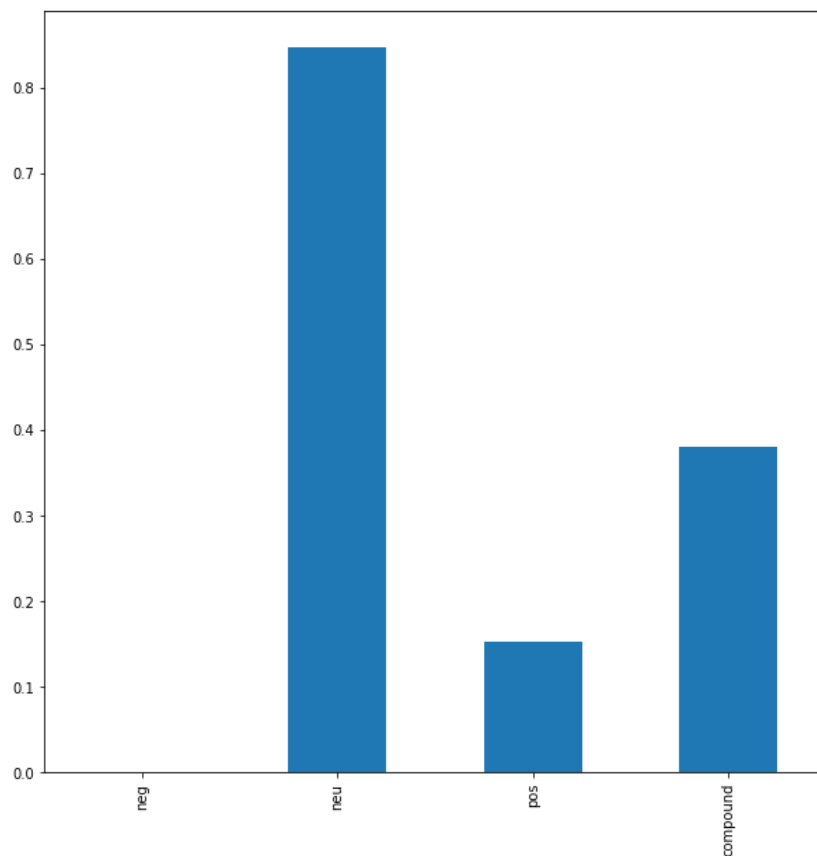
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7bfd5ddacd0>
```



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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7bfd6df7cd0>
```



Vemos claramente que el texto lo podemos considerar como neutro, nada negativo y la vertiente positiva nos ayudan a decir que el texto es en terminos generales es positivo

¿Qué es compuesto en Vader? La puntuación compuesta es la suma de puntuaciones positivas, negativas y neutras que luego se normaliza entre -1 (negativo más extremo) y +1 (positivo más extremo).

Cuanto más se acerque la puntuación de Compound a +1, mayor será la positividad del texto.

En este caso 0.4 por lo que lo consideraremos como positivo

**Conclusion**

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 interpretación 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 11:28

