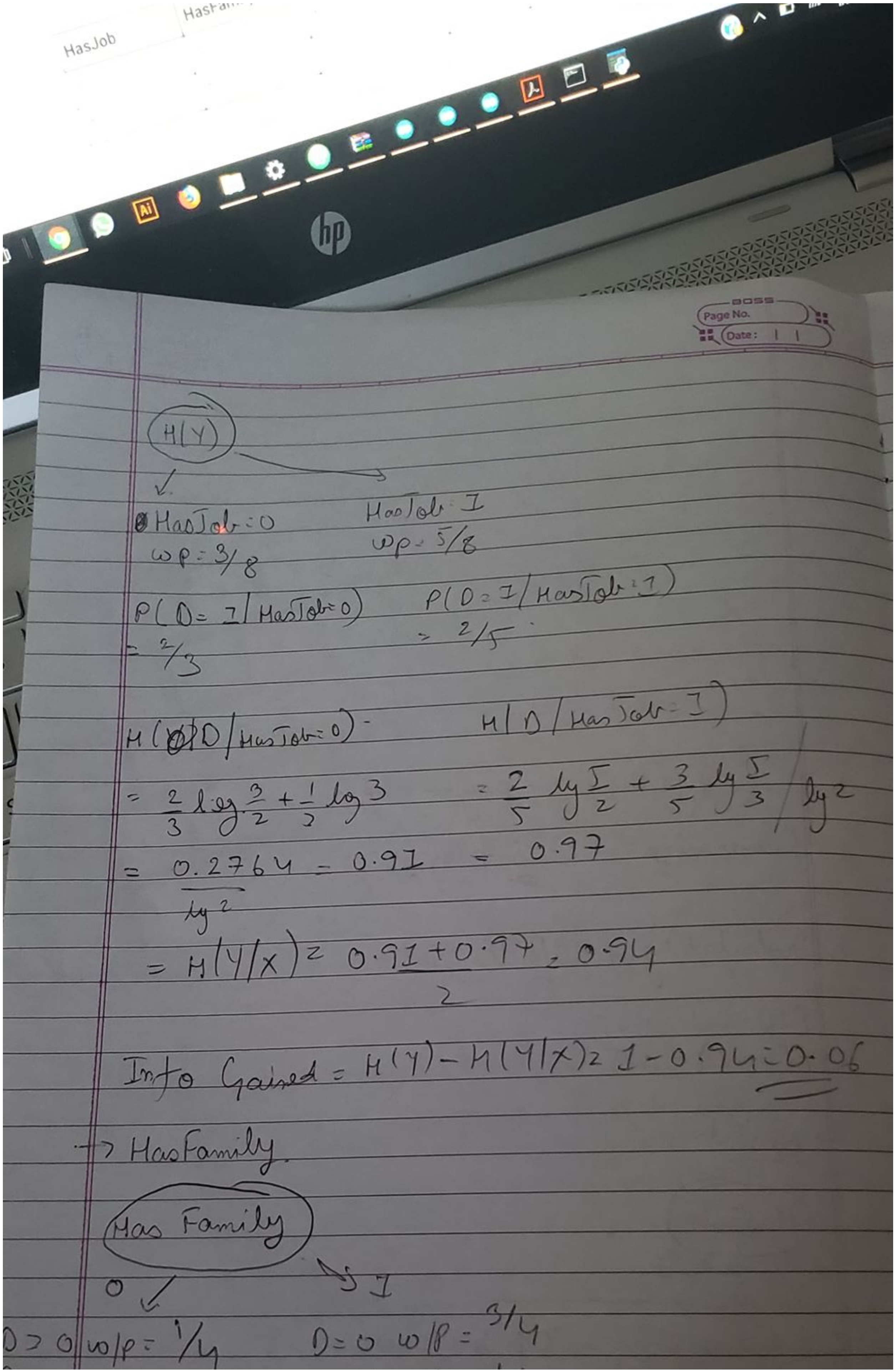
Part 1:- Entropy



$H(Y)$

HasJob = 0 $\quad wp = 3/8$

HasJob = 1 $\quad wp = 5/8$

$P(D=1 / \text{HasJob}=0) = 2/3$

$P(D=1 / \text{HasJob}=1) = 2/5$

$H(D / \text{HasJob}=0)$

$= \frac{2}{3}\log\frac{3}{2} + \frac{1}{3}\log 3$

$= \frac{0.2764}{\log 2} = 0.91$

$H(D / \text{HasJob}=1)$

$= \frac{2}{5}\log\frac{5}{2} + \frac{3}{5}\log\frac{5}{3} \Big/ \log 2$

$= 0.97$

$= H(Y/X) = \frac{0.91+0.97}{2} = 0.94$

Info Gained $= H(Y) - H(Y|X) = 1 - 0.94 = \underline{0.06}$

→ HasFamily

Has Family

0 ↙ ↘ 1

$D=0 \quad w/p = 1/4$

$D=0 \quad w/p = 3/4$

$$\begin{tikzcd} & H(Y) \arrow[dl] \arrow[dr] & \\ \text{HasFam} = 0 & & \text{HasFam} = 1 \end{tikzcd}$$

$w/P = 0.5$ $\quad$ $w/p = 0.5$

$P(D=1 \mid \text{HasFam}=0)$ $\quad$ $P(D=1 \mid \text{HasF}=1)$

$= 3/4$ $\quad$ $= 1/4$

$H(D \mid \text{HasFam}=0)$ $\quad$ $H(D \mid \text{HasFam}=1)$

$= \frac{3}{4}\log\frac{4}{3} + \frac{1}{4}\log 4$ $\quad$ $= \frac{3}{4}\log\frac{4}{3} + \frac{1}{4}\log 4$

$= 0.81$ $\quad$ $= 0.81$

$H(Y \mid X) = 0.81 \times 0.5 + 0.81 \times 0.5 = 0.81$

Info Gained $= ~~0~~\ 1 - 0.81 = \boxed{0.19}$

→ IsAbove 30 Years.

$$\begin{tikzcd} & \text{IsAbove30Years} \arrow[dl] \arrow[dr] & \\ 0 & & 1 \end{tikzcd}$$

$D=0$ $w/P =$ ~~0.5~~ $1/2$ $\quad$ $D=0$ $w/p = 3/6 = 1/2$

$D=1$ $w/P = 1/2$ $\quad$ $D=1$ $w/p = 3/6 = 1/2$

$$\begin{tikzcd} & H(Y) \arrow[dl] \arrow[dr] & \\ \text{IsAbo30} = 0 & & \text{IsAbov30} = 1 \end{tikzcd}$$

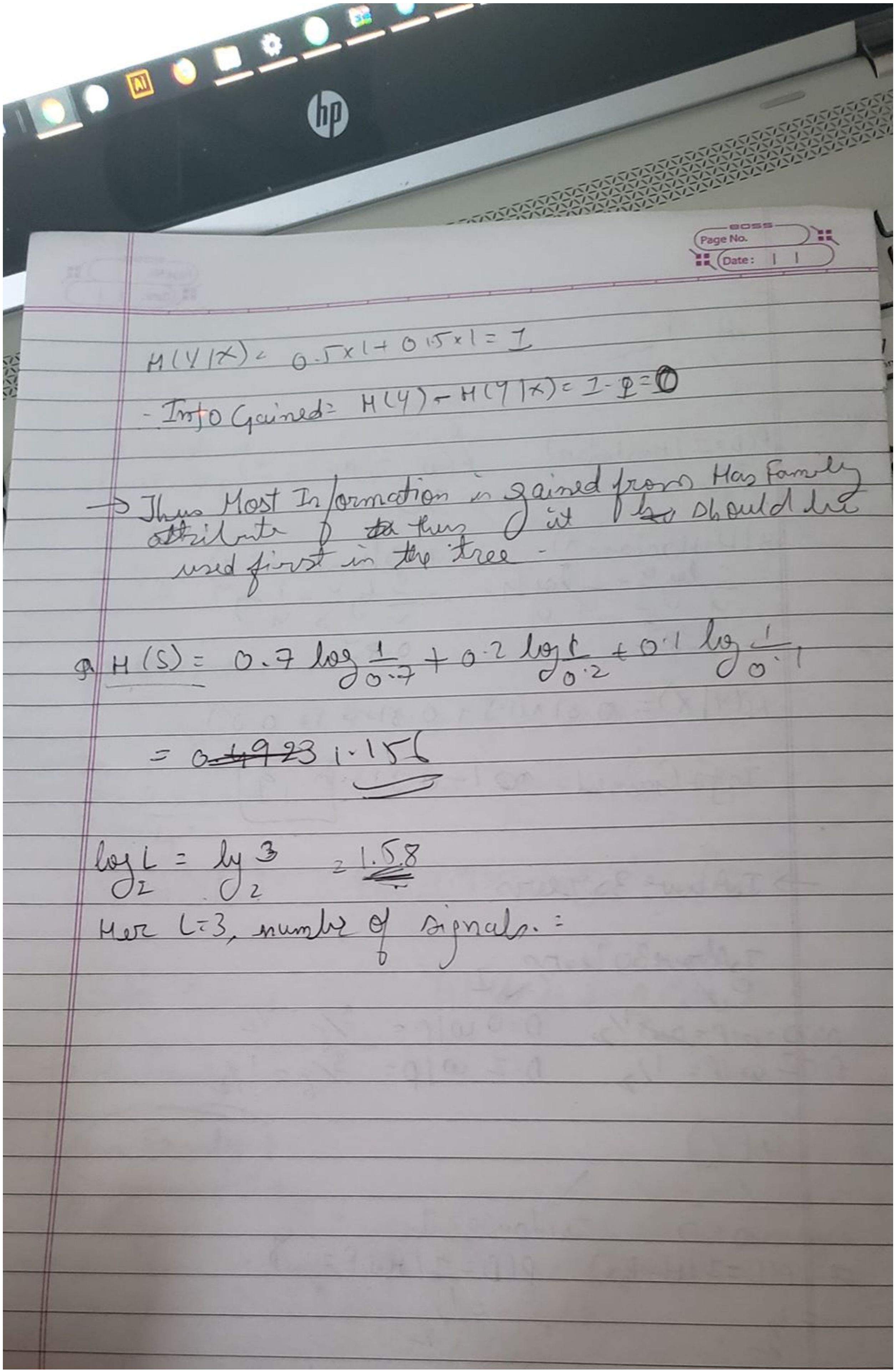
$P(D=1 \mid \text{HasF}=0)$ $\quad$ $P(D=1 \mid \text{HasF}=\cdot)$

Information gained from the HasJob is 0.06
Information gained from the HasFamily is 0.19
Information gained from the IsAbove30Years is 0
Thus according to calculations information gained from the HasFamily is maximum i.e. 0.19.
So, it will be the best feature for the first split.

Q2 Given a signal of three symbols S=(A, B, C) and P(A)=0.7, P(B)=0.2, P(C)=0.1, What is the entropy of S? What does it mean according to the Source coding Theorem ?
→ By solving the above problem the values of entropy for S H(S) = 1.156, and according to source coding theorem the value of H(S) = Summationof(p*log(1/p)) <= logN.
And by solving that we get log3 = 1.58.

Thus the source coding theorem proves that the bits of the 3 mentioned signal is between the range of (1.156, 1.58), it cannot be less that 1.156 and cannot exceed 1.58.

Below attached is the copy of the calculation page for entropy.



$H(Y|X) = 0.5 \times 1 + 0.5 \times 1 = \underline{1}$

- $\underline{\text{Info Gained}} = H(Y) - H(Y|X) = \underline{1} - \underline{1} = \boxed{0}$

→ Thus Most Information is gained from Has Family attribute & ~~the~~ thus it ~~has~~ should be used first in the tree.

Q $\underline{H(S)} = 0.7 \log \frac{1}{0.7} + 0.2 \log \frac{1}{0.2} + 0.1 \log \frac{1}{0.1}$

$= ~~0.4923~~ \ \underline{\underline{1.156}}$

$\log_2 L = \log_2 3 = \underline{\underline{1.58}}$

Here $L = 3$, number of signals =

Part 2: NLP

1. What is the difference between a Bag Of words Model in NLP and a Word2vec Model, discuss advantages of one over the other?

→ Bag of Words and Word2Vec model both tend to different aspects of characterising the text from the document.
Bag of words is basically the extraction of information from a document like words, unigram etc. Where as Word2Vec model is a type of data structure where information can be arranged in matrix format.
In Bag of words we basically extract information from the document in a unigrams and then we arrange them in the different permutations of multigram, bigram to extract information. And in Word2Vec model we use the bag of word to arrange them into vector and get a data-structure out of it. Basically a Co-occurrence matrix.
The disadvantage of the Bag of Word model is that it loses the sequence of the model thus results in ignorance of semantics from the sentence.
Whereas in vector representation the sentence does not loss its semantics since its arranged in the array, thus one can extract information just by changing the bit in that array.
Since Word2Vec is creation of distributional representations of the words, it is very much easy to see the text in organised manner without repeating the words i.e. saving space
Also making Word2Vec gives an advantage to find the similarities between the sentences.
To make Word2Vec first we need to construct Bag of Vector.

2. What is a word vector? What is a word Embedding? On what factors does the word embedding of a word depend (explain it from a NLP perspective)

→ Word Vector is a group of models related to each other that are used to produce the word embedding. It is a 2 layer neural network trained to reconstruct the linguistic context of the words. It takes input from the large corpus of the words and arranges into data structure.
Word embedding is the unity of the set of language modeling and future learning techniques from the NLP. Here words are mapped to vectors. I.e. Word embedding is nothing but a text converted into number that makes sense to the computer and thus computer can perform different tasks.
Word embedding is basically divided into two parts, Frequency based and Prediction based thus word embedding of a word is dependent on Count Vector, TF-IDF, Vector Co-Occurrence Vector, Bag of Words.

3. What is a corpus in NLP? How is the vocabulary of a model different from the corpus?

→ Corpus is a large set of the structured text that is used for the statistical analysis of the text. Corpus includes different type of data for text like tags, sentences etc. Whereas the vocabulary is a collection of words without repetition. Also corpus comprises of some semactical details of a word like say there is a same word but is noun form and verb form, one can easily differentiate it using corpus. Whereas in case of vocabulary one cannot do that.

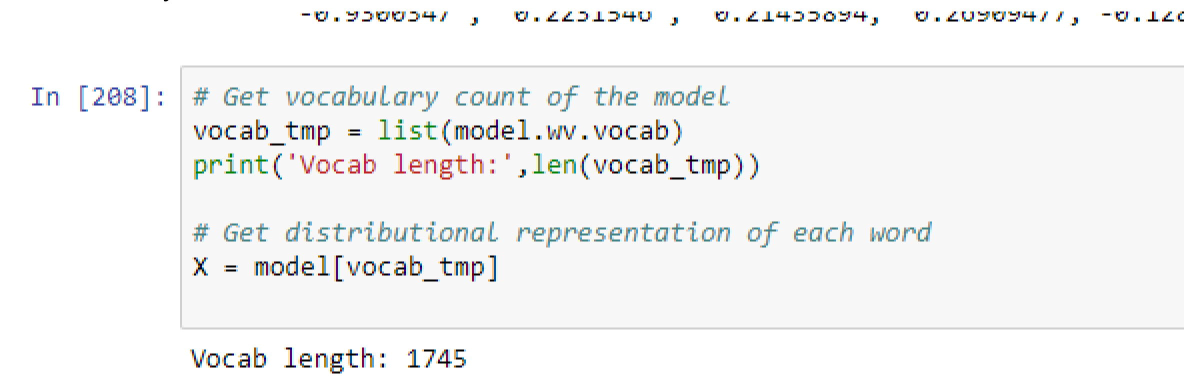
4. Train a word2vec model on the corpus consisting of the text in the novel Pride and Prejudice.

a) Write a short summary describing your preprocessing and modeling technique.

→ For this model I am first removing the warnings. And the i'm cleaning the sentences by removing the tags, converting all the letters in lowercase using split() method, also removing all english stop words, Joining the words back into one string separated by space, appending the emoticons to the end. Then making a corpus from bag of words and training the word2vec model. After training the model we will be testing the model on different functions like similar, mostsimiliar, doesnt_match and finally evaluating the embedding model number, vocabulary count and a PCA Embedding graph.

b) Report the vocabulary count, embedding size, number of training iterations in your modelling

→ Vocabulary count for this model is :- 1745



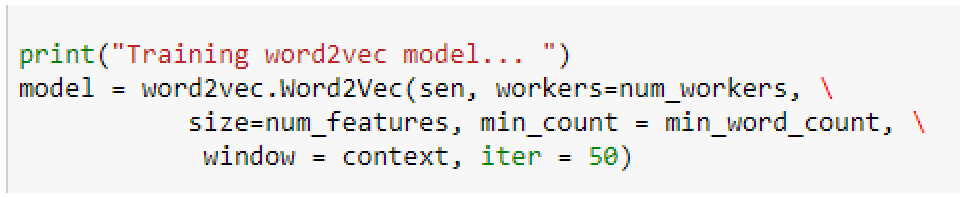
```
-0.9300547 , 0.2231540 , 0.21455894, 0.20909477, -0.12
```

```
In [208]: # Get vocabulary count of the model
          vocab_tmp = list(model.wv.vocab)
          print('Vocab length:',len(vocab_tmp))

          # Get distributional representation of each word
          X = model[vocab_tmp]

          Vocab length: 1745
```

Number of training iterations:- 50



```
print("Training word2vec model... ")
model = word2vec.Word2Vec(sen, workers=num_workers, \
            size=num_features, min_count = min_word_count, \
             window = context, iter = 50)
```

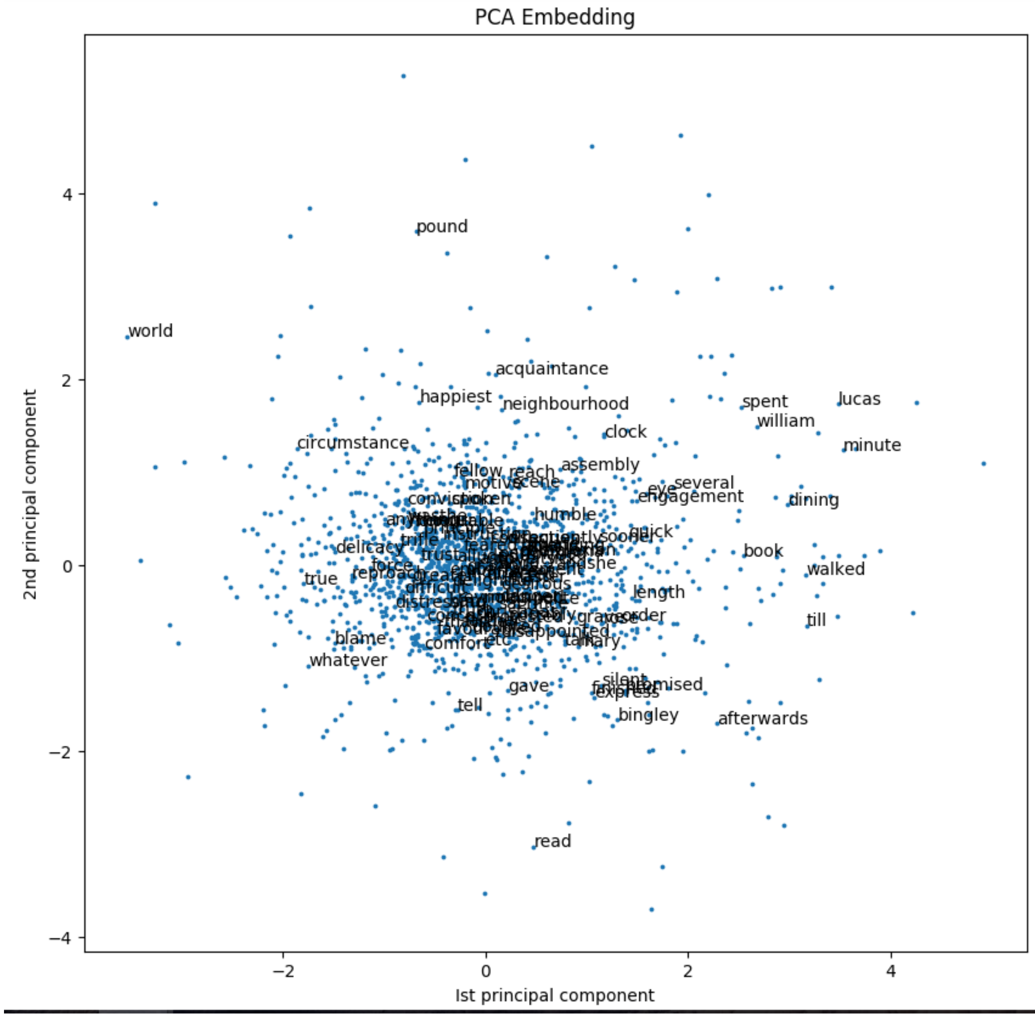
Number of embedding size:- (1745, 500)

c) Report and explain your observation from the visualization of word vectors

→ From the graph we come to conclusion on relation between one word to other.

Like say the word world doesn't even match the test, which will result in the opposite answers in the testimonial i.e doesnt_match function will take off 'world' when 'world place book ' are tested together, similarly for other words too this will be the case. This embedding graph also helps us to interpret different pattern for the words like which to use when.

The result of the graph is attached below.



d) Report any five intrinsic evaluation results that you performed using your model

Testing Similiarity

```
# Get cosine similarity of words
from sklearn.metrics.pairwise import cosine_similarity

model.similarity('girl','boy')
```

```
0.69198932991509654
```

```
model.similarity('woman', 'men')
```

```
0.83635626223417825
```

```
model.similarity('family', 'daughter')
```

```
0.18927051314399757
```

```
model.similarity('death', 'little')
```

```
0.42759598757792727
```

```
model.similarity('one', 'single')
```

```
0.53739901863610628
```

→

Testing doesnt match

```
model.doesnt_match('man woman till girl'.split())
```

```
'till'
```

```
model.doesnt_match('man world till'.split())
```

```
'till'
```

```
model.doesnt_match('man woman girl'.split())
```

```
'girl'
```

```
model.doesnt_match('earth saturn love'.split())
```

```
'love'
```

```
model.doesnt_match('mars jupiter men'.split())
```

```
'men'
```

```
model.most_similar("word")
```

```
[('interruption', 0.6156398057937622),
 ('lip', 0.5808097124099731),
 ('saying', 0.5720919370651245),
 ('speak', 0.5649688243865967),
 ('couldnot', 0.5605394244194031),
 ('valuable', 0.551924467086792),
 ('depend', 0.5424071550369263),
 ('without', 0.5264490842819214),
 ('anything', 0.525322437286377),
 ('upon', 0.5175555348396301)]
```

```
model.most_similar("till")
```

```
[('visitor', 0.8308674097061157),
 ('saturday', 0.8094621896743774),
 ('sunday', 0.781827986240387),
 ('returning', 0.7711752653121948),
 ('dressing', 0.768431544303894),
 ('along', 0.7605604529380798),
 ('got', 0.7460013628005981),
 ('chaise', 0.7387632131576538),
 ('downstairs', 0.734905481338501),
 ('hastily', 0.732915461063385)]
```

```
model.most_similar("girl")
```

```
[('extremely', 0.6928129196166992),
 ('boy', 0.6919894218444824),
 ('king', 0.6907191276550293),
 ('youngest', 0.6818444132804871),
 ('tall', 0.6741129159927368),
 ('younger', 0.6674612760543823),
 ('neighbourhood', 0.6519386768341064),
 ('son', 0.6438463926315308),
 ('married', 0.6389352679252625),
 ('sixteen', 0.6349356770515442)]
```

```
model.most_similar("man")
```

```
[('woman', 0.887626051902771),
model.most_similar("man")
```

```
[('woman', 0.887626051902771),
 ('men', 0.7938704490661621),
 ('person', 0.771087646484375),
 ('accomplished', 0.7617754340171814),
 ('totally', 0.7365685701370239),
 ('fortune', 0.7360684871673584),
 ('feature', 0.735831260681152 3),
 ('people', 0.7333755493164062),
 ('represented', 0.7166950106620789),
 ('influence', 0.7125266194343567)]
```

```
model.most_similar(positive=['husband','man'], negative=['woman'])
```

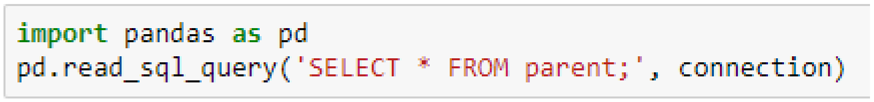
```
[('married', 0.6512598991394043),
 ('get', 0.6474043130874634),
 ('glad', 0.6298869848251343),
 ('introduce', 0.6025815010070801),
 ('ball', 0.5817509889602661),
 ('fond', 0.5778319835662842),
 ('jones', 0.5764567852020264),
 ('going', 0.5746958255767822),
 ('unless', 0.5700502395629883),
 ('fun', 0.5680896043777466)]
```

References:-

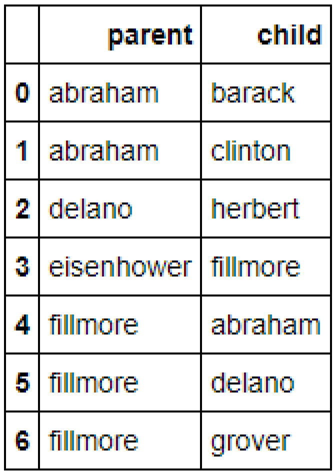
https://en.wikipedia.org/wiki/Word2vec
https://en.wikipedia.org/wiki/Word_embedding
https://en.wikipedia.org/wiki/Embedding
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
https://en.wikipedia.org/wiki/Text_corpus
https://machinelearningmastery.com/gentle-introduction-bag-words-model/
https://www.quora.com/In-NLP-what-is-the-difference-between-a-Lexicon-and-a-Corpus
https://github.com/ikhlaqsidhu/data-x/blob/master/07b-tools-word2vec_add_missing_si/Word2Vec.pptx.pdf
https://github.com/ikhlaqsidhu/data-x/blob/master/07a-tools-nlp-sentiment_add_missing_si/NLP1-slides_v2_afo.pdf

Part 3: SQL

1. Simple SELECTS (on the parents table)
1. SELECT all records in the table.

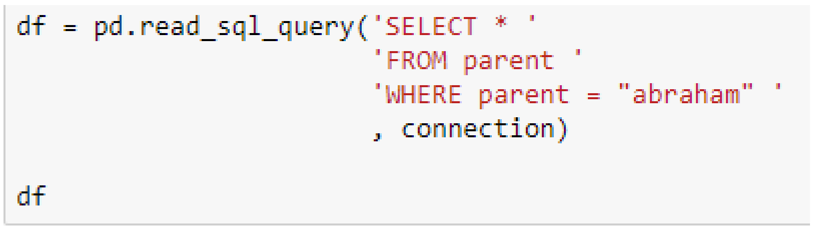


```
import pandas as pd
pd.read_sql_query('SELECT * FROM parent;', connection)
```



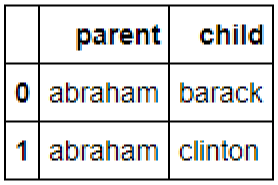
	parent	child
0	abraham	barack
1	abraham	clinton
2	delano	herbert
3	eisenhower	fillmore
4	fillmore	abraham
5	fillmore	delano
6	fillmore	grover

2. SELECT child and parent, where abraham is the parent.



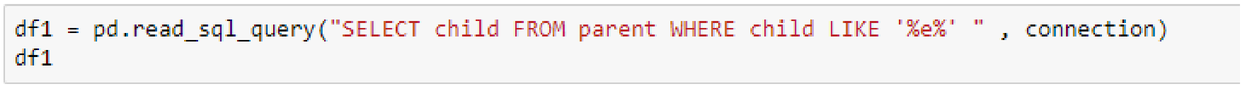
```
df = pd.read_sql_query('SELECT * '
                       'FROM parent '
                       'WHERE parent = "abraham" '
                       , connection)

df
```

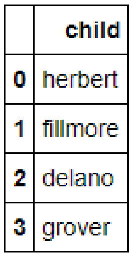


	parent	child
0	abraham	barack
1	abraham	clinton

3. SELECT all children that have an 'e' in their name (hint: use LIKE and '%e%').

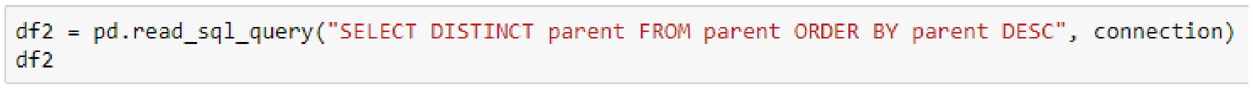


```
df1 = pd.read_sql_query("SELECT child FROM parent WHERE child LIKE '%e%' " , connection)
df1
```

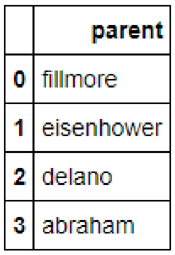


	child
0	herbert
1	fillmore
2	delano
3	grover

4. SELECT all unique parents (use SELECT DISTINCT) and order them by name, descending order (i.e. fillmore first)



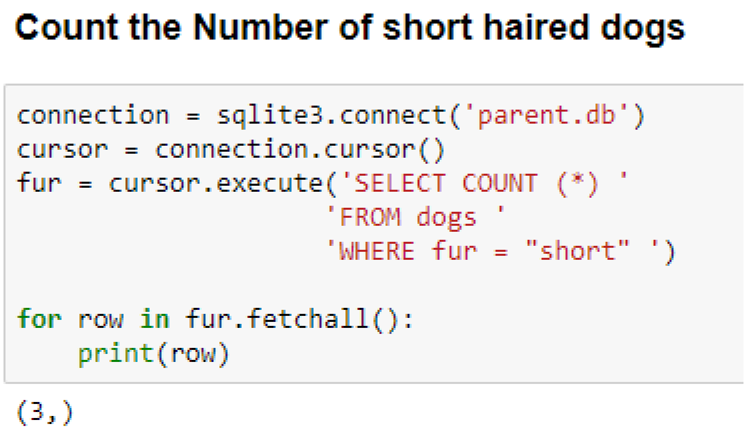
```
df2 = pd.read_sql_query("SELECT DISTINCT parent FROM parent ORDER BY parent DESC", connection)
df2
```



	parent
0	fillmore
1	eisenhower
2	delano
3	abraham

2) JOINS

1. COUNT the number of short haired dogs



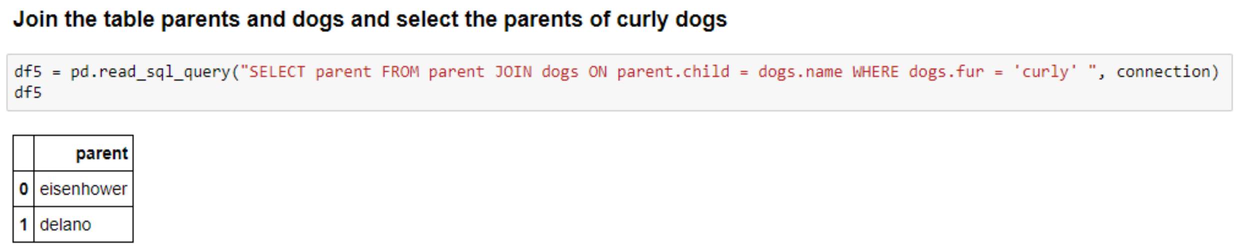
Count the Number of short haired dogs

```
connection = sqlite3.connect('parent.db')
cursor = connection.cursor()
fur = cursor.execute('SELECT COUNT (*) '
                     'FROM dogs '
                     'WHERE fur = "short" ')

for row in fur.fetchall():
    print(row)
```

```
(3,)
```

2. JOIN tables parents and dogs and SELECT the parents of curly dogs.

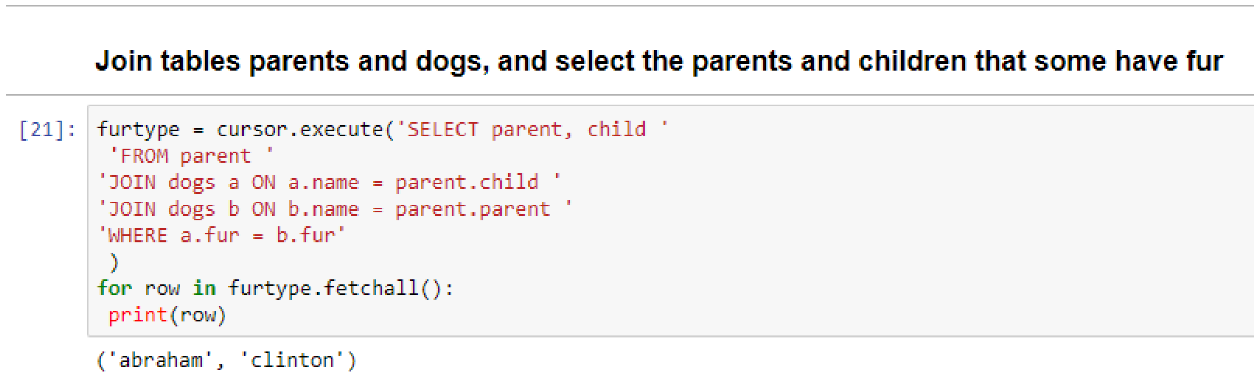


Join the table parents and dogs and select the parents of curly dogs

```
df5 = pd.read_sql_query("SELECT parent FROM parent JOIN dogs ON parent.child = dogs.name WHERE dogs.fur = 'curly' ", connection)
df5
```

	parent
0	eisenhower
1	delano

3. JOIN tables parents and dogs, and SELECT the parents and children that have the same fur type. Only show them once.



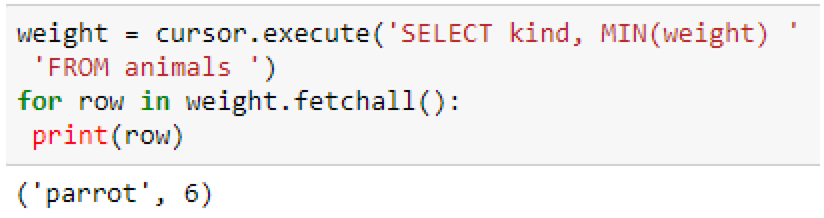
Join tables parents and dogs, and select the parents and children that some have fur

```
[21]: furtype = cursor.execute('SELECT parent, child '
       'FROM parent '
      'JOIN dogs a ON a.name = parent.child '
      'JOIN dogs b ON b.name = parent.parent '
      'WHERE a.fur = b.fur'
       )
      for row in furtype.fetchall():
       print(row)
```

```
      ('abraham', 'clinton')
```

3) Aggregate functions, numerical logic and grouping

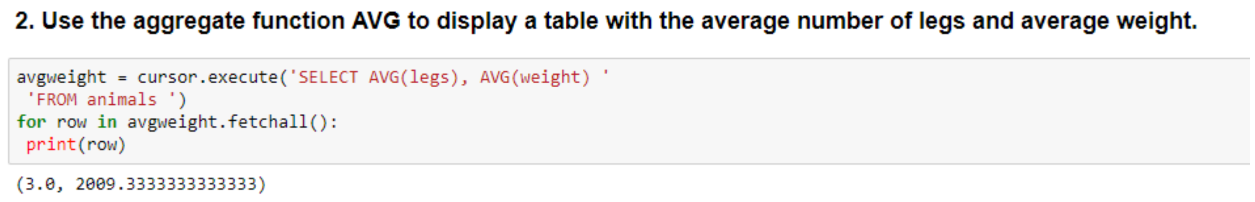
1. SELECT the animal with the minimum weight. Display kind and min_weight.



```
weight = cursor.execute('SELECT kind, MIN(weight) '
 'FROM animals ')
for row in weight.fetchall():
 print(row)
```

```
('parrot', 6)
```

2. Use the aggregate function AVG to display a table with the average number of legs and the average weight.

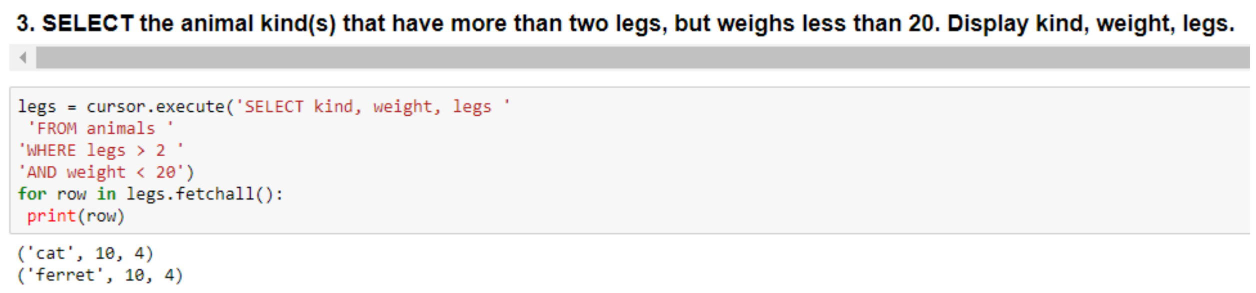


2. Use the aggregate function AVG to display a table with the average number of legs and average weight.

```
avgweight = cursor.execute('SELECT AVG(legs), AVG(weight) '
 'FROM animals ')
for row in avgweight.fetchall():
 print(row)
```

```
(3.0, 2009.3333333333333)
```

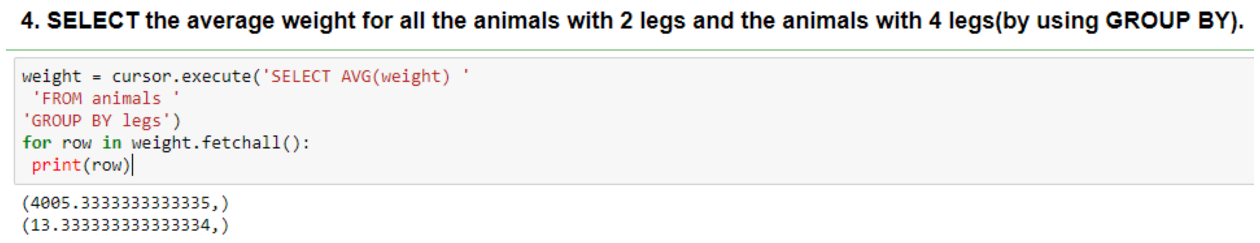
3. SELECT the animal kind(s) that have more than two legs, but weighs less than 20. Display kind, weight, legs.


3. SELECT the animal kind(s) that have more than two legs, but weighs less than 20. Display kind, weight, legs.

```
legs = cursor.execute('SELECT kind, weight, legs '
 'FROM animals '
'WHERE legs > 2 '
'AND weight < 20')
for row in legs.fetchall():
 print(row)
```

```
('cat', 10, 4)
('ferret', 10, 4)
```

4. SELECT the average weight for all the animals with 2 legs and the animals with 4 legs
(by using GROUP BY).


4. SELECT the average weight for all the animals with 2 legs and the animals with 4 legs(by using GROUP BY).

```
weight = cursor.execute('SELECT AVG(weight) '
 'FROM animals '
'GROUP BY legs')
for row in weight.fetchall():
 print(row)
```

```
(4005.3333333333335,)
(13.333333333333334,)
```