How to break Wordle with Beam and BigQuery

Iñigo San Jose Visiers

Software Engineer @ Google

BEAM SUMMIT

Austin, 2022

Agenda

- What, why and how?
- Solving Wordle
- Conclusions

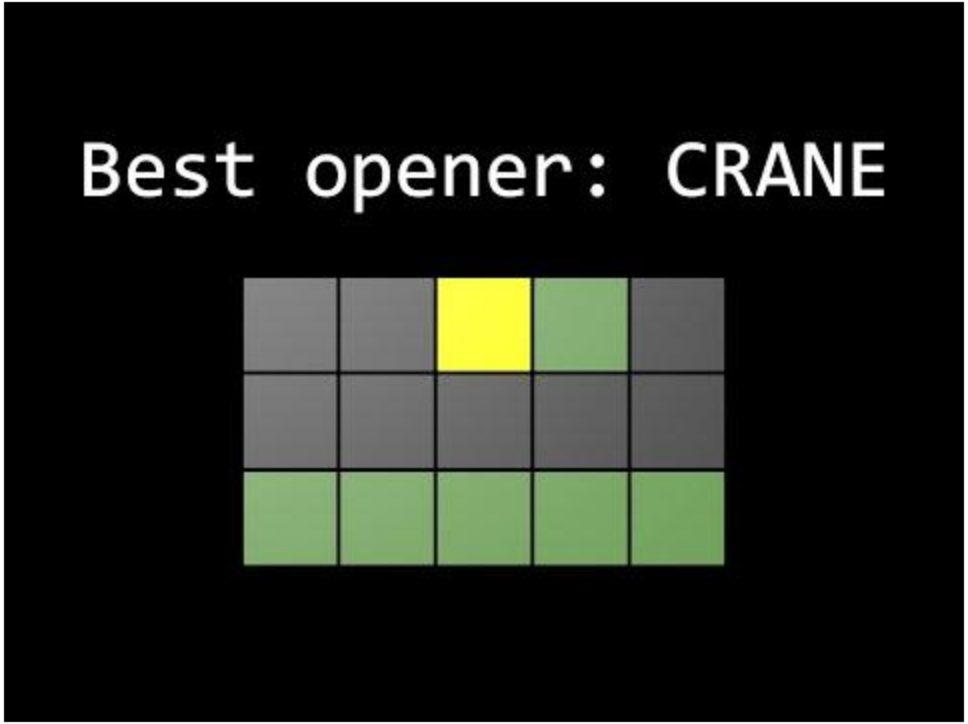
What, what not why & how

- The what and what not
- The why
- The how

What and what not

- We are going to compute all* three word combinations and score them
- We care about the average result over all possible answers
- **This is not a Wordle Solver**, but an analysis
- We aim for best success ratio, not least attempts

Worlde Solver by *3Blue1Brown*



Why?

Why?

WHY NOT?

Why?

- This is a good example of divide & conquer
- It shows how to prepare our data to speed up our analysis

Why?

- This is a good example of divide & conquer
- It shows how to prepare our data to speed up our analysis
- I was bored and needed an excuse to stop working

How?

- Since this is a highly parallelizable task, Beam is a good fit
- We don't know enough about the end result data, we need something to analyze
- BigQuery

Solving Wordle

- Problem statement
- The Beam Side
- The BigQuery Side

Problem Statement

Problem Statement - Space calculation

- Wordle allows ~13000 words to be played
- That's 2.182×10^{12} possible 3-word combinations
- We have to be smarter than this

Problem Statement - Reducing the space

1 - Filter words with duplicate letters — HELLO ✗ LOGIC ✓

2 - Only combine words with no common letters — LOGIC, GREAT ✗ LOGIC, HANDY ✓

3 - Remove duplicates — LOGIC, HANDY ✗ HANDY, LOGIC ✓

Problem Statement - Benefits

1. From 2 Trillion combinations, we go down to 144 Million (~15250 times less)

2. Calculating a word's score is faster

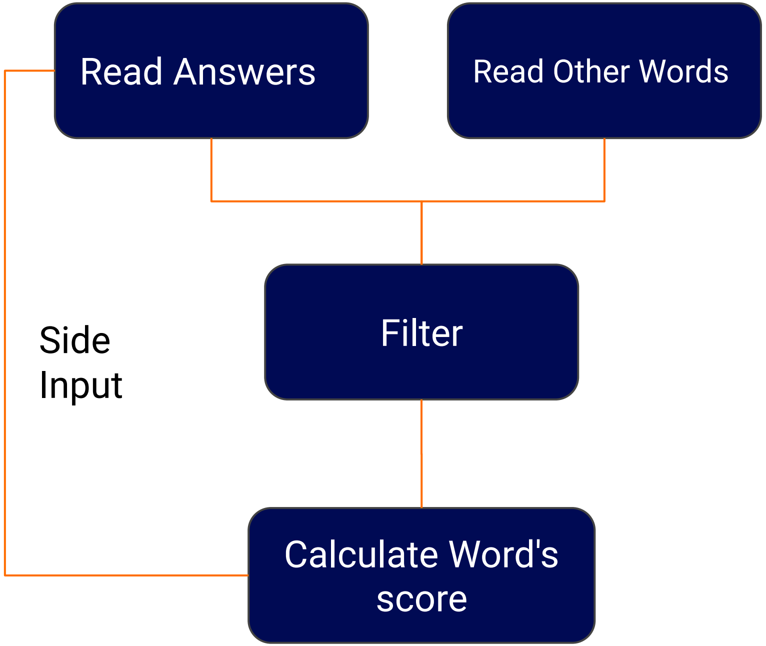
3. Combining scores is commutative and associative

The Beam side

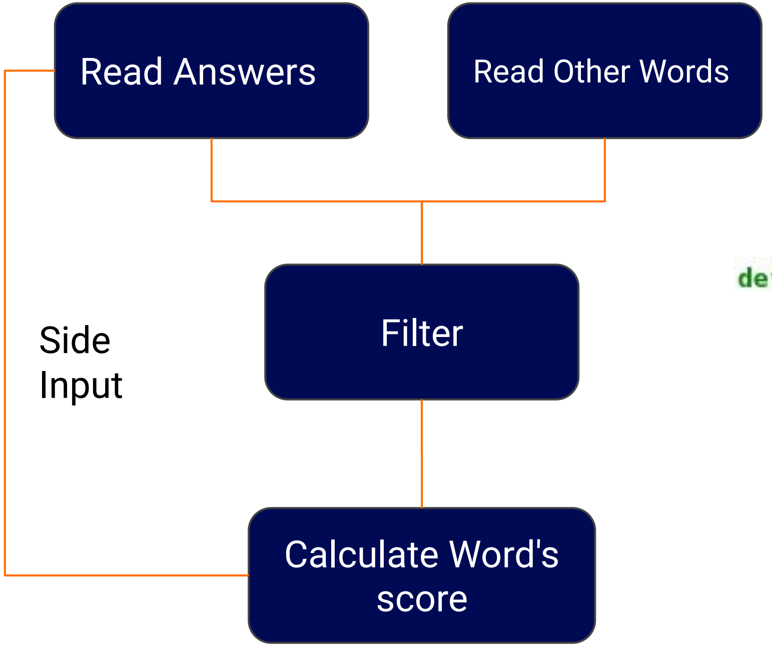
The Beam Side - Planning the pipeline

- Read words
- Filter words with duplicate letters
- Calculate a word's score
- Join the words and scores
- Filter Duplicates
- Write to BigQuery

The Beam Side - Reading, Filtering & Scoring

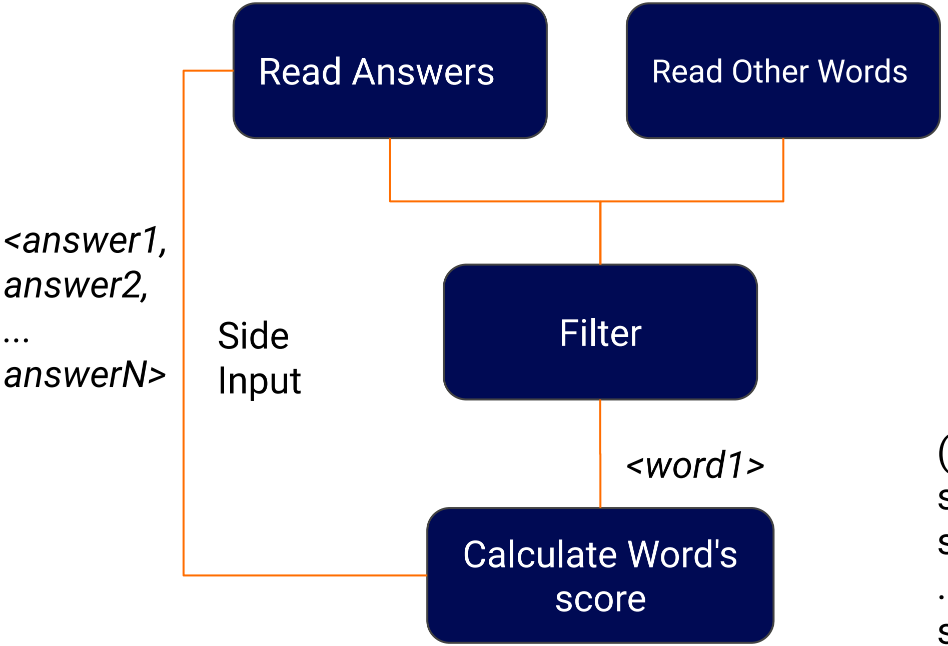


The Beam Side - Reading, Filtering & Scoring



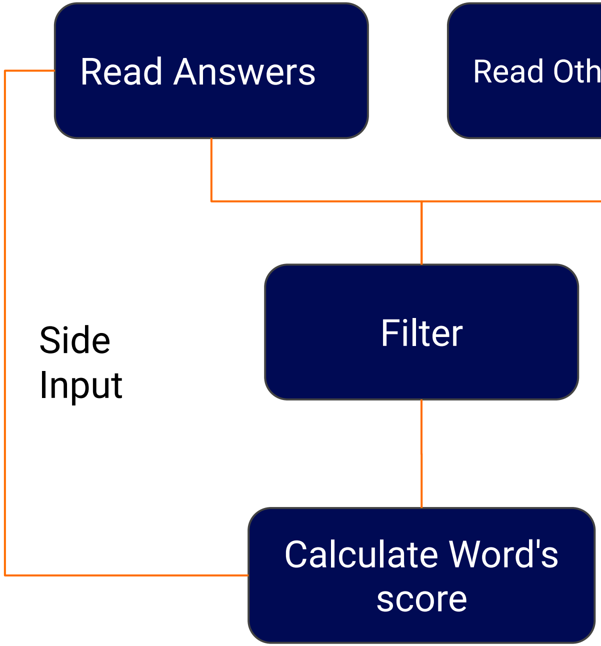

```python
def single_letter(word):
    s = set()
    for l in word:
        s.add(l)
    return len(s) == len(word)
```

The Beam Side - Reading, Filtering & Scoring



(word1,
score(word1, answer1) +
score(word1, answer2) +
...
score(word1, answerN))

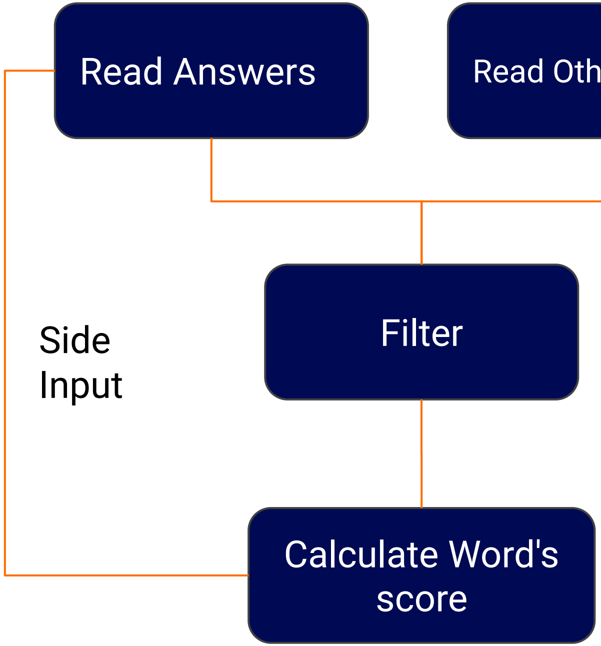
The Beam Side - Reading, Filtering & Scoring




```python
class WordleRow(DoFn):
    def process(self, word, answers):
        greens, yellows = 0, 0
        for answer in answers:
            for i, letter in enumerate(word):
                if letter == answer[i]:
                    greens += 1
                elif letter in answer:
                    yellows += 1
        yield self._format_result(word, yellows, greens)

    def _format_result(self, word, yellows, greens):
        return (word, yellows, greens, 1)
```

The Beam Side - Reading, Filtering & Scoring



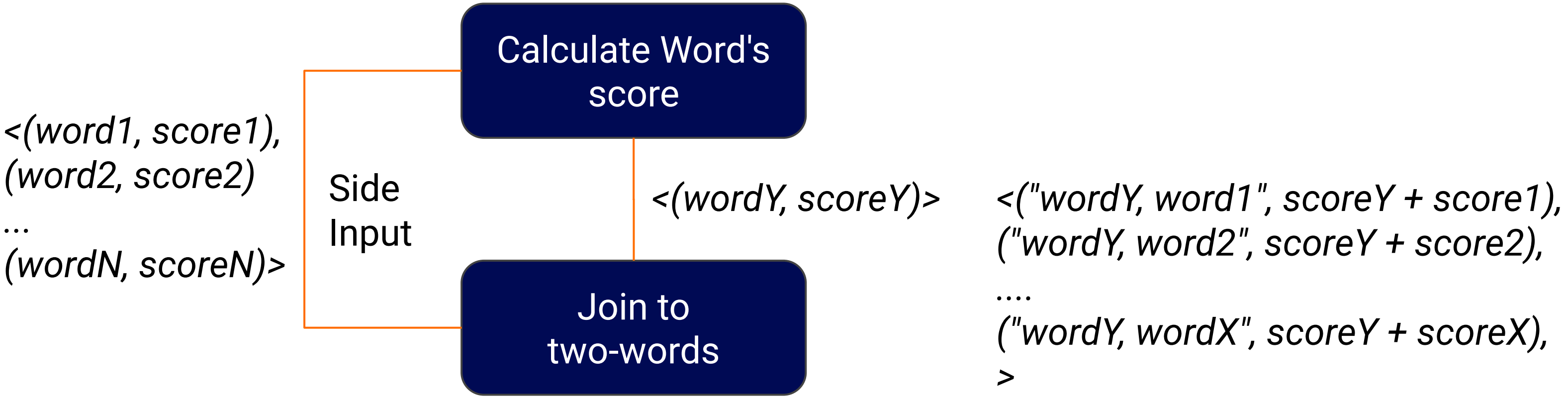
Iterate through all answers for each element

```python
class WordleRow(DoFn):
    def process(self, word, answers):
        greens, yellows = 0, 0
        for answer in answers:
            for i, letter in enumerate(word):
                if letter == answer[i]:
                    greens += 1
                elif letter in answer:
                    yellows += 1
        yield self._format_result(word, yellows, greens)

    def _format_result(self, word, yellows, greens):
        return (word, yellows, greens, 1)
```

Format as tuple

The Beam Side - Combining words



The Beam Side - Combining words

<(word1, score1),
(word2, score2)
...
(wordN, scoreN)>

Side
Input

Calculate Word's score

<(wordY, scoreY)>

Join to two-words

<("wordY, word1", scoreY + score1),
~~*("wordY, word2", scoreY + score2),*~~
....
("wordY, wordX", scoreY + scoreX),
>

The Beam Side - Combining words

Calculate Word's score

Side Input

<(word1, score1), (word2, score2)
...
(wordN, scoreN)>

<(wordX, scoreX)>

Join to two-words

<("wordY, word1", scoreY + score1),
~~*("wordY, word2", scoreY + score2),*~~
....
("wordY, wordX", scoreY + scoreX),
>,
<"wordX, wordY", scoreX + scoreY>

Distinct

The Beam Side - Combining words

<(word1, score1),
(word2, score2)
…
(wordN, scoreN)>

Side Input

Calculate Word's score

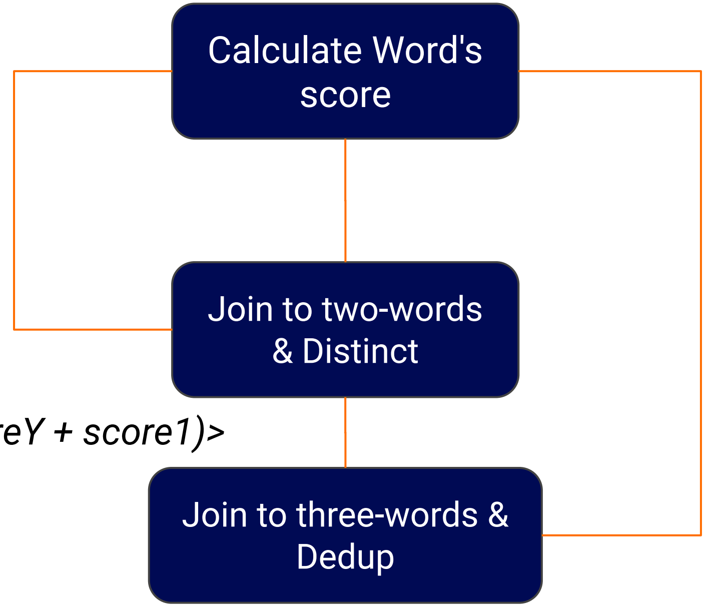
<(wordX, scoreX)>

Join to two-words

Distinct

<("wordY, word1", scoreY + score1),
~~*("wordY, word2", scoreY + score2),*~~
….
("wordY, wordX", scoreY + scoreX),
>,
~~*<"wordX, wordY", scoreX + scoreY>*~~

The Beam Side - Combining words



<(word1, score1),
(word2, score2)
...
(wordN, scoreN)>

<("wordY, word1", scoreY + score1)>

The Beam Side - Combining words

```python
def combine_words_new(main, side_words, size=3):
    def _letter_intersection(main_dict, side_word):
        for l in side_word:
            if l in main_dict:
                return True
        return False

    def _combine_tuples(word, t1, t2):
        return (word, t1[1] + t2[1], t1[2] + t2[2], t1[3] + t2[3])

    main_word = main[0]
    main_dict = {}
    for l in main_word:
        main_dict[l] = 1

    if size == 3:
        list_words = main_word.split(",")

    for side in side_words:
        side_word = side[0]
        intersection = _letter_intersection(main_dict, side_word)

        if not intersection:
            if size == 3:
                words = list_words + [side_word]
                new_word = ",".join(sorted(words))
            elif main_word > side_word:
                new_word = f"{side_word},{main_word}"
            else:
                new_word = f"{main_word},{side_word}"
            yield _combine_tuples(new_word, main, side)
```

The Beam Side - Combining words

```python
def combine_words_new(main, side_words, size=3):
    def _letter_intersection(main_dict, side_word):
        for l in side_word:
            if l in main_dict:
                return True
        return False

    def _combine_tuples(word, t1, t2):
        return (word, t1[1] + t2[1], t1[2] + t2[2], t1[3] + t2[3])

    main_word = main[0]
    main_dict = {}
    for l in main_word:
        main_dict[l] = 1

    if size == 3:
        list_words = main_word.split(",")

    for side in side_words:
        side_word = side[0]
        intersection = _letter_intersection(main_dict, side_word)

        if not intersection:
            if size == 3:
                words = list_words + [side_word]
                new_word = ",".join(sorted(words))
            elif main_word > side_word:
                new_word = f"{side_word},{main_word}"
            else:
                new_word = f"{main_word},{side_word}"
            yield _combine_tuples(new_word, main, side)
```

Side input

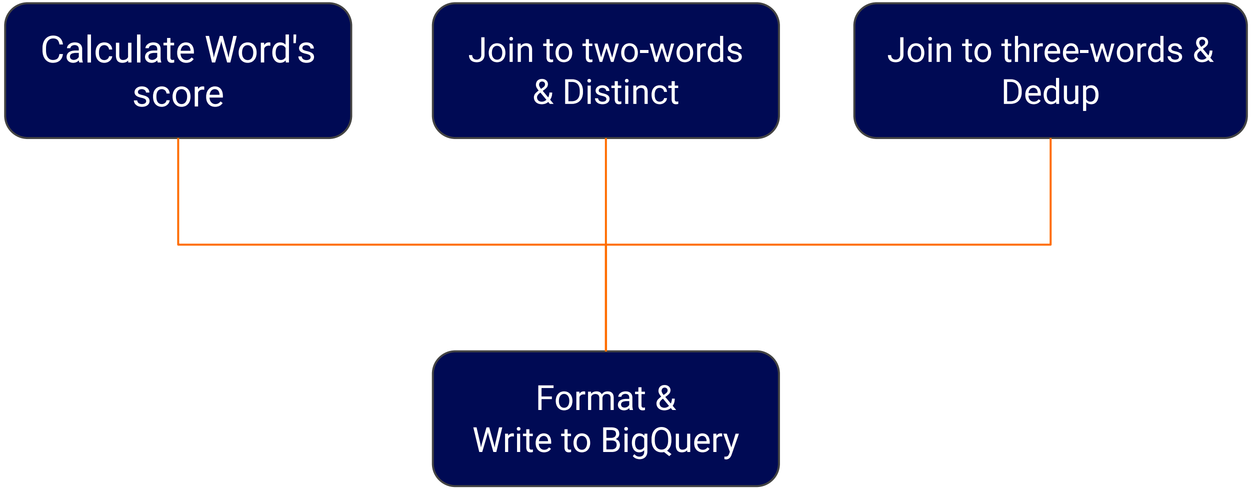
Do the words have common letters?

NO

Format the combination for deduping

Sum the scores

The Beam Side - Writing to BigQuery



The BigQuery side

The BigQuery Side - How to analyze the data

The BigQuery Side - How to analyze the data

- How do we score the combinations?

- What is our strategy?

The BigQuery Side - How to analyze the data

- Greens are more valuable than yellows, but less when there are more
  - With three words, greens are 1.75 times more valuable
  - With two words, greens are 2.25 times more valuable

- We want the best three-word combination that performs well with two words
  - Out of the best 3-combinations, we create a the possible two combinations and rank them

The BigQuery Side - How to analyze the data

Row	words	yellows	yellow_avg	greens	green_avg	amount_words	total_words
1	gazed,jumby,snick	4096	1.7693304535637149	2505	1.08207343412527	3	2315
2	forby,muzak,pinch	4352	1.879913606911447	2320	1.0021598272138228	3	2315
3	bufty,glisk,moved	4352	1.879913606911447	2624	1.1334773218142549	3	2315
4	gauzy,voxel,wrick	4352	1.879913606911447	2657	1.1477321814254859	3	2315
5	shaky,vibex,would	4352	1.879913606911447	2671	1.15377969762419	3	2315
6	chink,pudgy,zaxes	4352	1.879913606911447	2419	1.0449244060475162	3	2315
7	dumpy,gawks,zilch	4352	1.879913606911447	1916	0.827645788336933	3	2315
8	fable,pudgy,shock	4352	1.879913606911447	2945	1.2721382289416847	3	2315
9	bludy,finch,gopak	4352	1.879913606911447	2468	1.0660907127429806	3	2315
10	bungy,major,whift	4352	1.879913606911447	2477	1.0699784017278617	3	2315
11	bowat,midgy,pluck	4352	1.879913606911447	2491	1.0760259179265659	3	2315
12	fuzed,thong,wispy	4608	1.9904967602591792	2308	0.99697624190064793	3	2315
13	cinqs,judge,wormy	4608	1.9904967602591792	2313	0.99913606911447084	3	2315
14	aping,botch,dumky	4608	1.9904967602591792	2322	1.003023758099352	3	2315
15	chimb,flunk,vaped	4608	1.9904967602591792	2325	1.0043196544276458	3	2315
16	grind,spumy,thack	4608	1.9904967602591792	2837	1.2254859611231101	3	2315

The BigQuery Side - First Query and Results

```sql
SELECT
  words,
  greens,
  yellows,
  green_avg,
  yellow_avg,
  1.75 * green_avg + yellow_avg AS weighted_score
FROM
  `table`
WHERE
  amount_words=3
ORDER BY 1.75 * green_avg + yellow_avg  DESC
```

The BigQuery Side - First Query and Results

```sql
SELECT
  words,
  greens,
  yellows,
  green_avg,
  yellow_avg,
  1.75 * green_avg + yellow_avg AS weighted_score
FROM
  `table`
WHERE
  amount_words=3
ORDER BY 1.75 * green_avg + yellow_avg  DESC
```

Row	words	greens	yellows	green_avg	yellow_avg	weighted_score
1	count,pride,shaly	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
2	coady,print,shule	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
3	crude,point,shaly	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
4	crine,poult,shady	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
5	coaly,pride,shunt	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
6	crudy,point,shale	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
7	chant,prude,soily	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
8	coude,print,shaly	3588	5434	1.5498920086393089	2.3473002159827212	5.0596112311015116
9	douce,print,shaly	3584	5438	1.5481641468682505	2.3490280777537795	5.0583153347732175
10	dault,pricy,shone	3546	5476	1.5317494600431965	2.3654427645788338	5.0460043196544273
11	dhole,pricy,saunt	3546	5476	1.5317494600431965	2.3654427645788338	5.0460043196544273
12	dhole,print,saucy	3546	5476	1.5317494600431965	2.3654427645788338	5.0460043196544273
13	drice,phony,sault	3546	5476	1.5317494600431965	2.3654427645788338	5.0460043196544273
14	count,drape,shily	3538	5484	1.52829373650108	2.3688984881209505	5.0434125269978409
15	crape,doily,shunt	3538	5484	1.52829373650108	2.3688984881209505	5.0434125269978409
16	count,drily,shape	3538	5484	1.52829373650108	2.3688984881209505	5.0434125269978409

The BigQuery Side - Analyzing first results

- Getting more than 5 weighted score seems good enough, but maybe there's a difference in the two-word combinations

The BigQuery Side - Analyzing first results

- Getting more than 5 weighted score seems good enough, but maybe there's a difference in the two-word combinations

"count,pride,shaly", 5.06
...
"jumps,kylix,vozhd", 2.66

The BigQuery Side - Analyzing first results

- Getting more than 5 weighted score seems good enough, but maybe there's a difference in the two-word combinations

"count,pride,shaly", 5.06
...
~~"jumps,kylix,vozhd", 2.66~~

"count,pride", ?
"count,shaly", ?
"pride,shaly", ?

The BigQuery Side - The (heavy) query

```
WITH double_words AS (
  SELECT
    ARRAY[
      CONCAT(SPLIT(words, ',')[OFFSET(0)], ',', SPLIT(words, ',')[OFFSET(1)]),
      CONCAT(SPLIT(words, ',')[OFFSET(0)], ',', SPLIT(words, ',')[OFFSET(2)]),
      CONCAT(SPLIT(words, ',')[OFFSET(1)], ',', SPLIT(words, ',')[OFFSET(2)])
    ] word_array,
    words as three_words,
    yellow_avg + 1.75 * green_avg as three_score
  FROM
    `table`
  WHERE
    amount_words = 3
    AND yellow_avg + 1.75 * green_avg > 5
  )

SELECT
  w.words,
  yellow_avg + 2.25 * green_avg weighted,
  yellow_avg,
  green_avg,
  three_words,
  three_score
FROM
  `table` w,
  double_words d,
  UNNEST(d.word_array) words_2
WHERE
  w.words IN (words_2)
  AND amount_words = 2
ORDER BY
  yellow_avg + 2.25 * green_avg DESC,
  d.three_score DESC
```

Split the 3-words

The BigQuery Side - The (heavy) query

```
WITH double_words AS (
  SELECT
    ARRAY[
      CONCAT(SPLIT(words, ',')[OFFSET(0)], ',', SPLIT(words, ',')[OFFSET(1)]),
      CONCAT(SPLIT(words, ',')[OFFSET(0)], ',', SPLIT(words, ',')[OFFSET(2)]),
      CONCAT(SPLIT(words, ',')[OFFSET(1)], ',', SPLIT(words, ',')[OFFSET(2)])
    ] word_array,
    words as three_words,
    yellow_avg + 1.75 * green_avg as three_score
  FROM
    `table`
  WHERE
    amount_words = 3
    AND yellow_avg + 1.75 * green_avg > 5
  )

SELECT
  w.words,
  yellow_avg + 2.25 * green_avg weighted,
  yellow_avg,
  green_avg,
  three_words,
  three_score
FROM
  `table` w,
  double_words d,
  UNNEST(d.word_array) words_2
WHERE
  w.words IN (words_2)
  AND amount_words = 2
ORDER BY
  yellow_avg + 2.25 * green_avg DESC,
  d.three_score DESC
```

Split the 3-words

Filter

The BigQuery Side - The results

Row	words	weighted	yellow_avg	green_avg	three_words	three_score
1	prate,soily	4.4116630669546435	1.7719222462203024	1.1732181425485961	dunch,prate,soily	5.0126349892008637
2	crine,slaty	4.4015118790496759	1.7637149028077754	1.172354211663067	crine,dough,slaty	5.0060475161987039
3	soily,trade	4.3880129589632828	1.8095032397408208	1.1460043196544276	punch,soily,trade	5.0022678185745137
4	crine,sault	4.350215982721382	1.83585313174946	1.1174946004319655	crine,podgy,sault	5.0044276457883363
5	briny,slate	4.3098272138228939	1.6963282937365012	1.1615550755939525	briny,pouch,slate	5.007991360691145
6	brine,slaty	4.3098272138228939	1.6963282937365012	1.1615550755939525	brine,pouch,slaty	5.007991360691145
7	chore,saint	4.2981641468682508	1.8362850971922247	1.0941684665226783	chore,duply,saint	5.0032397408207343
8	crone,saith	4.2960043196544273	1.838012958963283	1.09244060475162	crone,duply,saith	5.0019438444924411
9	crate,shily	4.286069114470842	1.723110151187905	1.1390928725701943	crate,pound,shily	5.0385529157667381
10	crate,shily	4.286069114470842	1.723110151187905	1.1390928725701943	bound,crate,shily	5.0144708423326136
11	crate,shily	4.286069114470842	1.723110151187905	1.1390928725701943	crate,mound,shily	5.0064794816414686
12	sadly,trine	4.2841252699784018	1.7969762419006479	1.1053995680345572	pouch,sadly,trine	5.0103671706263508
13	pricy,slate	4.26695464362851	1.7127429805615551	1.1352051835853132	hound,pricy,slate	5.0119870410367167
14	praty,slice	4.26695464362851	1.7127429805615551	1.1352051835853132	hound,praty,slice	5.0119870410367167
15	price,slaty	4.26695464362851	1.7127429805615551	1.1352051835853132	hound,price,slaty	5.0119870410367167
16	shily,trace	4.2666306695464362	1.7386609071274297	1.1235421166306696	pound,shily,trace	5.026889848812095

The BigQuery Side - The winner

PRATE, SOILY, DUNCH

The BigQuery Side - The winner

PRATE, SOILY, DUNCH

Honorable Mention

CRATE, SOILY, BUNDH

Can we do better?

Questions?

https://github.com/InigoSJ