

ה-Beef בין קנדרייק למאր לדרייק: מי יותר טוב?

טקסט Conteinos – עיבוד שפה טבעית וניתוח רשות



* An art piece about the song "Not Like Us" written by Kendrick about Drake, posted on Reddit Miguelpaco

מגישיים:

אמיר יטיב 207128513

Micah Yafe 206376675

תוכן עניינים

3	1. בחירת הנושא והצגת הנתונים
4	2. ניתוח רשותות
4	נתוניים כלליים על הרשות
5	יזואלייזציה
5	מטריקות עיקריות
7	מדדי מרכזיות
10	קהילות
11	3. עיבוד שפה
11	עיבוד טקסט מקדים
12	ניתוח סנטימנט
14	מילוט מפתח
15	ניתוח נושאי
17	ענן מילים
19	4. נספחים

* העבודה היא 18 עמודים בלבד – לא להיכל מהנספחים

1. בחירת הנושא והציגת הנתונים

Beef



to have a [grudge](#) or [start](#) one with [another](#) person.

([50 cent: life's on the line](#)) "Beef u dont want [none](#) so dont start none"

* Urban Dictionary

הרשאות החברתיות משמשות במא לשיח ציבורי ולהשמעת דעתות מגוונות של המשתמשים בדיון בנושאים שונים בחלוקת, כמו ריב או ויכוח מתוקשר בין שני מפורטים – "כוף". דוגמאות מפורטים ל"ביפים" מהשנים האחרונות:

- בטכנולוגיה: אילון מאסק נגד ג'ף בזוס
- בקולנוע: וין דיזל נגד "זה רוק"
- בספורט: רונאלדו נגד מס'

המונ "ביפים" שייצים לעולם המוזיקה ובפרט בזאנר ההיפ-הופ. בפרויקט שלנו בחרנו להתמקד ב"ביפ" **הספרטיבי בין הרابر קנדרייק למאר לרابر דרייק**.

סביר הריב המתוקשר בין קנדרייק למאר לדרייק יש המונ פעילות ברשות החברתיות. מבחינת ניתוח רשות – מעניין לראות איך המשתמשים מתנהגים, איך הם מתפלגים ומתקשרים ביניהם והאם אפשר לחלק אותם לקבוצות ברורות.

ניתוח שפה טבעית וסלג הוא תחום שימושו אוטומטית לבוחר בנושא זה. אנחנו חשבים שהשימוש במודלים של עיבוד שפה יעזר לנו להבין יותר טוב את הנושא – לבחון את הדעות והידע הקיים שלנו וגם לאגלות מגמות שלא הכרנו.

סוגי הנתונים של פפני

עכבר פרוייקט זה בחרנו ליצא נתונים גלובליים היישר מהרשת החברתית "Reddit". רשת חברתיות זו היא פלטפורמה מעוניינת למחקר כי השיח בה הוא סביב "קהילות" ("Subreddit") בנושאים שונים.

כדי ליצור בסיס נתונים כולל על ה"ביפ" של קנדרייק ודרייק, בחרנו לאסוף:

- פוסטים מוביילים מהסאברדייט "Drake/kendrickLamar"/z" שמכילים את המילה "Drake"
- פוסטים מוביילים מהסאברדייט "Drizzy" Kendrick" שמכילים את המילה "Kendrick"

עכבר החלק הראשון של ניתוח רשות חברתיות ריכזו את 50 הפוסטים המוביילים מכל קטגוריה – הכותבים שלהם, הסאברדייט שבו נכתבו, התגובה. (כותבי הפוסטים יהיו הצמתים ותגובה תוצג על ידי קשת מכונת – סה"כ 50 צמתים ו-612 קשות).

עכבר החלק השני של ניתוח הטקסט עליה הוכיח את המאגר שלנו. אספנו את 200 הפוסטים המוביילים בכל סאברדייט, על כל תגיותיהם. סך הכל בסיס נתונים של 8111 מסמכים (פוסטים/תגיות) - 3150 מסמכים מהסאברדייט של דרייק ו-4961 מסמכים מהסאברדייט של קנדרייק.

2. ניתוח רשות

נתונים כלליים על הרשות

הצמתים ברשות: מייצגים משתמשי רדייט שככתבו את 50 הפוסטים המוביילים בסאברדייטים "r/Drizzy" ו-"r/KendrickLamar".

הקשות שמחברות ביןיהם: הן תגיות על פוסטים שהמשתמשים הולו. ככלומר אם משתמש הגיב למשתמש אחר תחבר בינוין קשה. יש שימושות לכיוון של הקשר – המשתמש המגיב והמשתמש שעליו הגיבו, لكن בחרנו גרף מסווג מכוון.

משקלים: בחרנו בקשרות לא ממושקלות כי לא ראיינו צורך בלהת תוכנה לקשת שמייצגת תגובה למשתמש.

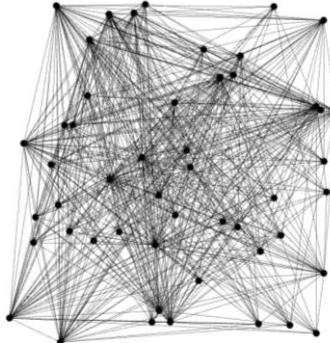
תוכנות ברשות: שביל לענות על השאלה "מי יותר טוב?" החלטנו שהשוו לשומר עבור כל משתמש איפה הוא כתב את הפוסט שלו – בסאברדייט של קנדרייק או של דרייק. שמרנו זאת כתוכנה עבור כל קודקוד, ובמה שיר נראה זאת בגרף באמצעות צבעים.

שאלות מחקר: בגרף כמו שלנו שמייצג רשות חברתית מעניין לשאול

- למי הקהל יותר תומך? מי הצליח להשפיע על דעת הקהלה בדרדייט
- איך מתנהגים התומכים של דרייק לעומת התומכים של קנדרייק? האם הם מייצרים אינטראקציות ביניהם או שמדובר בקהילות סגורות?
- אילו מילוטים סלנג "תפסו" שיש סביבה "ביפ"? איזה מילוטים בולטים אפשר לזהות ומה המשמעות שלהם?
- האם כל השיח סובב סביב אותם נושאים או שאפשר לחלק אותו לסוגי שיח שונים?

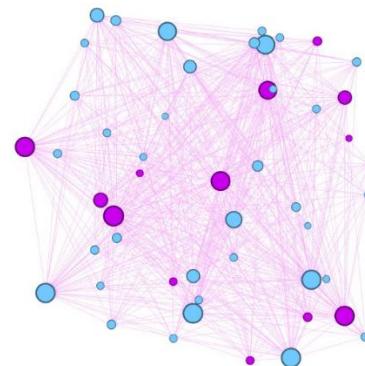
ווייזואליזציה

כדי לייצר וייזואליזציה של הרשת השתמשנו ב-Gephi. כך נראה הגרף הגלמי:



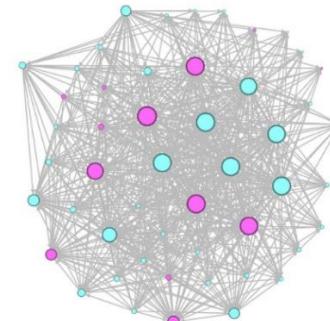
משמעות לא אינפורטטיבי. אלה הפעולות שביצענו כדי לשפר את הויזואליזציה של הגרף:

- a. צביעת הקודקודים לפי קטגוריה (הסאברדייט שבו כתבו)
- סאברדייט של קנדרייך = תכלת, סאברדייט של דרייך = סגול. מיד ניתן לראות שקיבלנו יותר משתמשים שכתו בסאברדייט של קנדרייך מאשר בסאברדייט של דרייך! אולי הוא יותר פופולארי ומושך יותר ?traffic
- b. גודל הקודקודים לפי הדרגה – קודקוד גדול הוא בעל דרגה גבוהה (נכנסת וייצאת ביחיד). ניתן לראות מי הקודקודים ברשת שיש להם יותר פעילות – הגיבו הרבה או קיבלו הרבה תגובות.



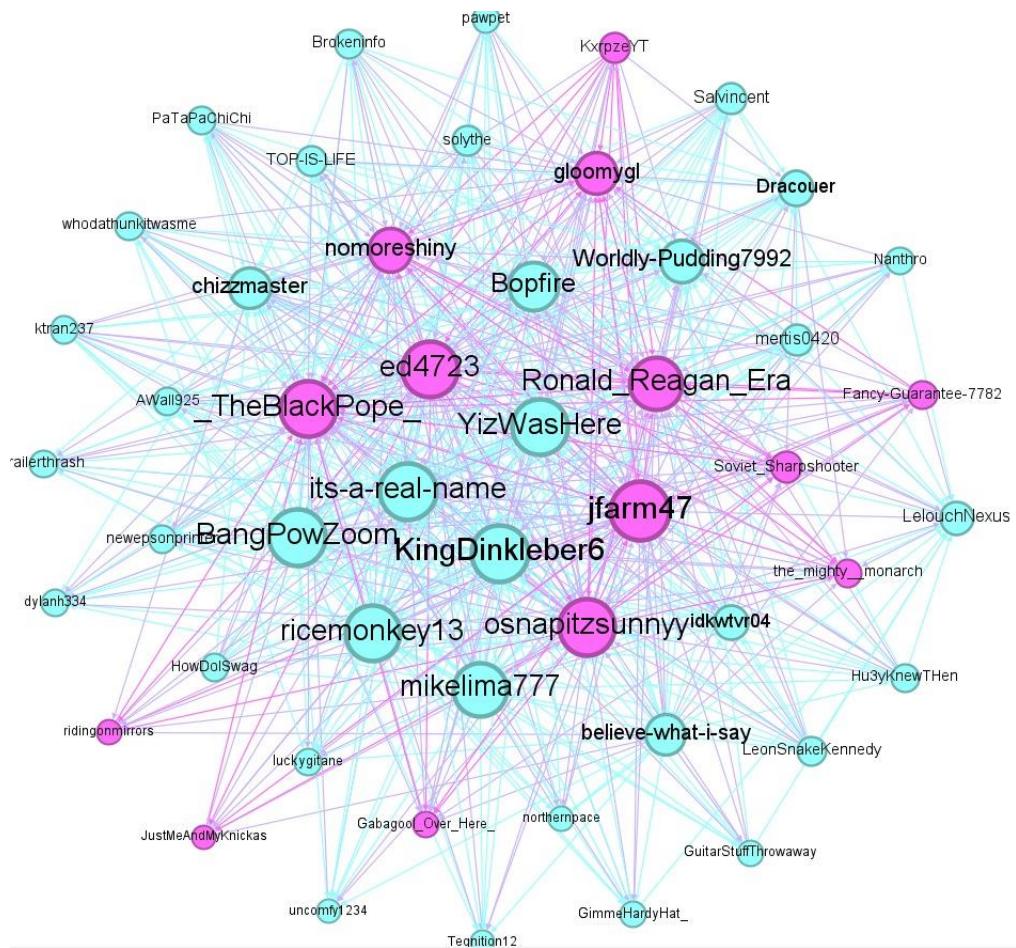
נכנסנו לבדוק מי הקודקוד 'KingDinkleber6' שיש לו דרגה גבוהה מאוד (48, כמעט מקסימלית) ודרגה כניסה נמוכה של 34. הוא כתוב פוטט בסאברדייט של קנדרייך עם מילוי שצוחקים על דרייך והוא קיבל הסכמה נרחבת בתגובה.

- g. שיטת הפרישה של הגרף –
ניסינו כמה אלגוריתמים שונים. הנה לדוגמה 2 Force Atlas 2:



מהר מאוד הבנו שאין המון אופציית שמתאים לנו ובחרנו ב Fruchterman Reingold שנלמד בכיתה.

ד. **הוספה Label לכל קודקוד** – עוזר לנו לדעת מי משתמשים הבודקים. הנה הוייזואלייזציה הסופית עם פרישת Fruchterman Reingold ושמות משתמשים:



מטריקות עיקריות

קוטר הגרף הוא 3 – כלומר המרחק המקסימלי בין 2 קודקודים בגרף הוא באורך 3 צלעות בלבד! כולם די קרובים.

כפיות הגרף היא 0.25 – ערך זה התקבל מ Gephi אך גם בחישוב פשוט יכולנו להגיע לאותה תוצאה: כפיות היא כמות הקששות בגרף ביחס לגרף מלא על אותה קבוצת קודקודים.

יש 50 צמתים ו-612 קשתות. בגרף מלא ומכoon עם 50 צמתים היי $2,450$ קשותות ($49 * 50$).
לנו יש 612 קשותות, שזה רבע מבגרף מלא. לכן הצפיפות 0.25 .

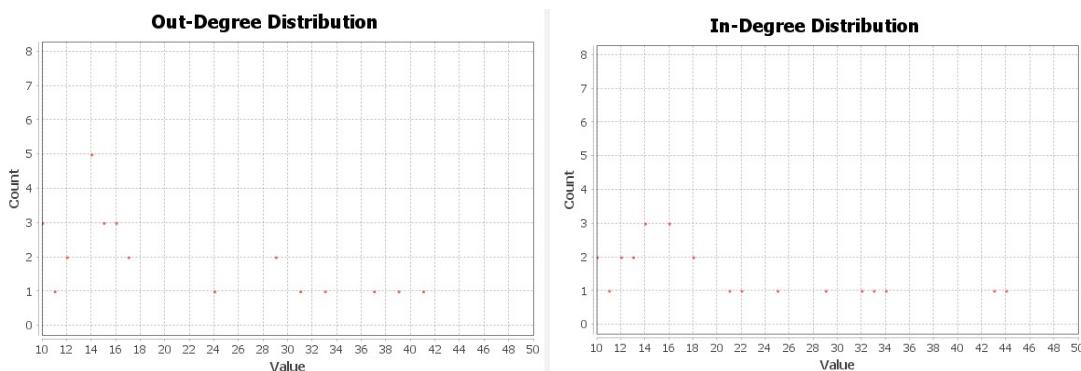
הגרף הוא גראף קשבי

אנחנו יודעים זאת כי גראף לא קשור הקוטר הוא אינסוף. ואצלנו רק 3 😊 במקורה שלנו זה מלמד שלכטובי 50 הפוסטים המובילים יש גם פעילות רבה ומוגנת מבחינות התגוכות, ולא ניתן להפריד את הרשת לרכיבי קשריות נפרדים.

הדרגה הממוצעת ברשת כפי שמייצג Gephi מציג היא 12.24.

במבט ראשון זה נראה לנו מאוד מוזר כי הדרגה המינימלית שאנו רואים בפירות היא 11. אבל חשוב לציין שככל קשת יוצאה היא גם כניסה ולכן מספרת בדרגה של שני הצמתים שלה. אם מחשבים 612 צמתים בגרף חלקו 50 קודקודים מקבלים 12.24.

התעמקנו לראות איך מתפלגות הדרגות הנכנסות והיצאות:



משמעותו 2 משתמשים שאין להם תגוכות מבין משתמשי הגרף (דרגה כניסה 0), ו-3 משתמשים שלא הגיעו כלל על פוסטים של משתמשי הגרף (דרגה יציאה 0).

יש גם משתמשים סופר-פופולריים: כמעט כולם הגיעו להם (דרגות כניסה 43/44), ומשתמשים סופר-פעילים: הגיעו לפחות לפוסטים של רוב משתמשי הגרף (דרגות יציאות 37/39/41).

גשרים ונקודות חיתוך

מכיוון שמדובר בגרף מכוון אין משמעות למושג גשר, אבל ניתן לבדוק אם יש צמתים שהורדה שלהם תגדיל את מספר רכיבי הקשריות – כלומר צמתים שהם cut-points.

נקודות חיתוך חיפשנו באמצעות Python. טענו את הגרף לפיזיון וביקשנו לקבל סקירה כללית באמצעות פונקציה info מספריית ach. קיבלנו שאין בגרף שום נקודות חיתוך (התבלה קבוצה ריקה). זה הגיוני כי הגרף מאד קשור ומראה פעילות רבה, צפינו שלא יהיה ניתן לפרק אותו בклות לרכיבי קשריות.

```
Graph with 50 nodes and 612 edges
Articulation points: []
There are no cutpoints

Process finished with exit code 0
```

מדדי מרכזיות

מדדי המרכזיות של מדרנו בכיתה הם:

- מدد לצפיפות, כמה קשרים יש לצומת הזאת בגרף. – **Degree Centrality**
- ממד שבודק כמה הנקודות בגרף (רגילה או מנורמלת) – **Betweenness Centrality**
- הן "קשרות" ומחווקות בין רכיבים שונים, מודד את המרכזיות של הצומת במסלול שמחבר בין כל שני צמתים אחרים בגרף.
- ממד קלות שבה ניתן להגיע בין **Closeness Centrality** (רגילה או הרמוני) – ממד לצמתים האחרים בגרף. הצומת לצמתים האחרים בגרף.

הריצנו את הפונקציות המתאימות מספרית אח וקיבלונו כפלט את רשימת המרכזיות של כל הקודקודים לפי כל אחד מהמדדים:

```
degree_centrality: {'believe-what-i-say': 0.5918367346938775, 'ktran237': 0.3061224489795918, 'Ho
betweenness_centrality: {'believe-what-i-say': 0.01567176063880247, 'ktran237': 0.000183995385012
closeness_centrality: {'believe-what-i-say': 0.7101449275362319, 'ktran237': 0.5903614457831325,
```

הקודקודים שנמצאו במרכזים בגרף:

- 0.9999999999999999 – הצומת 'jfarm47' עם ממד מרכזיות של 99.99999999999999. נסלח לעצמנו ונעה ל.1. הצומת זהה מחובר לכל 49 הצמתים האחרים בגרף. בתמונה מתוך Gephi ניתן לראות שיש כמה צמתים עם דרגה גבוהה מאוד:

Id	Degree	Closeness Centrality	Betweenness Centrality
jfarm47	49	1.0	72.805964
KingDinkleber6	48	1.0	89.22209
its-a-real-name	47	1.0	27.856353
BangPowZoom	47	0.97561	37.206837
ed4723	47	0.974359	37.344571
YizWasHere	47	0.967742	49.914694
osnapitzsunnyy	47	0.942857	45.300012
TheBlackPope	46	1.0	14.583808
ricemonkey13	46	0.945946	39.533532
mikelima777	44	0.901961	22.246228

- גם במידה זה אנחנו מזהים אותם צמתים מובילים. מעניין לראות צמתים עם דרגות יחסית גבוהות (29,32) אבל Betweenness של 0.0. זאת מושם שהם מחוברים לקודקודים רכיבים אבל כנראה "עוקפים" אותם בכל המסלולים והם נעשים מיותרים.

Id	Betweenness Centrality	Degree
Worldly-Pudding7992	0.0	32
believe-what-i-say	0.0	29
newepsonprinter	0.0	15
ktran237	0.0	15
HowDolSwag	0.0	15
trailerthrash	0.0	15
luckygitane	0.0	14
dylanh334	0.0	14
Gabagool_Over_Here_	0.075499	13
ridingonmirrors	0.075499	12
JustMeAndMyKnickas	0.075499	11
Teqnition12	0.075499	11
uncomfy1234	0.075499	11
Brokeninfo	0.130038	17
pawpet	0.130038	15
the_mighty_monarch	0.287037	15
whodathunkitwasme	0.291667	16
GimmeHardyHat_	0.412079	13
northernpace	0.412079	12
PaTaPaChiChi	0.421705	16
AWall925	0.421705	15
GuitarStuffThrowaway	0.446561	13
Nanthro	0.561856	15
solythe	0.639303	18

• – גם במדד מרכזיות אנחנו מזהים את אותם צמתים מוכילים. זה לא מפתיע כי מדובר בכיתה שהմינים הולכים ביחד וקשה מאוד למצוא דוגמאות לצמתים עם שונה בין המינים האלה.

10 הצמתים המוכילים ב- Closeness Centrality Betweenness Centrality

Id	Closeness Centrality	Id	Betweenness Centrality
jfarm47	1.0	KingDinkleber6	89.22209
KingDinkleber6	1.0	jfarm47	72.805964
its-a-real-name	1.0	YizWasHere	49.914694
TheBlackPope	1.0	osnapitzsunnyy	45.300012
BangPowZoom	0.97561	ricemonkey13	39.533532
ed4723	0.974359	ed4723	37.344571
YizWasHere	0.967742	BangPowZoom	37.206837
ricemonkey13	0.945946	Bopfire	30.620035
osnapitzsunnyy	0.942857	Ronald_Reagan_Era	27.906614
mikelima777	0.901961	its-a-real-name	27.856353

קהילות

מהסתכלות על הגרף הצפוף שלנו והדרגות הגכוות של רוב הקודקודים, אנחנו משערים שלא נצליח לחלק לקהילות. ניסינו לחלק את הגרף לקהילות על ידי שימוש ב K-Core דרך פיתון. ניסינו להכנס כמה ערכי K ועבור אף אחד מהם לא הצליחו ליצור קהילות. הגרף באמת מאוד מלויד כמו ששיעורנו.

```
K-core decomposition statistics:  
k=1: nodes=50, edges=612  
k=2: nodes=50, edges=612  
k=3: nodes=50, edges=612  
k=4: nodes=50, edges=612  
k=5: nodes=50, edges=612
```

ניתן ללמוד מכך שהקהילה שלנו מאוד מלוידת. האלגוריתם לא הצליח להפריד אפילו בין תומכי של קנדרייק לתומכי של דרייק. זאת מושם שהמשתמשים המעורבים בתקשורת סביר ה"ביף" מאוד פעילים ומגיבים על פוסטים משני הצדדים של המתרס (משני הסאברדייטים).

3. עיבוד שפה

עיבוד טקסט מקדים

לכל הדרך אנחנו עובדים עם שני סוגים של קורפוסים:

- פופטים מהסابرדייט "Drake" שמכילים את המילה "KendrickLamar"
 - פופטים מהסابرדייט "Drizzy" שמכילים את המילה "Kendrick".
 זאת על מנת לחזור את הדעה של שני הצדדים על ה"כיף".

בהתחלת שלפננו בעזרת ה-API את 100 הפוסטיטים המוכילים מהסאברדי של דרייק שמכילים את המילה קנדיריק, ולהפוך.

אבל קיבלנו 2 בעיות:

1. בחלק גדול מהפוסטים לא היה טקסט
 2. חיפשו את המילים 'drake' ו-'kendrick' באותיות קטנות בלבד

וזה מיקדנו את הדרישה שלנו לפוסטים שמכילים "self text" – כלומר טקסט בגין הפוסט, וגם הרחכנו את החיפוש כך שלא יהיה רגש לאותיות קטנות או גודלות.

שלבי עיבוד מקדים שכיצן:

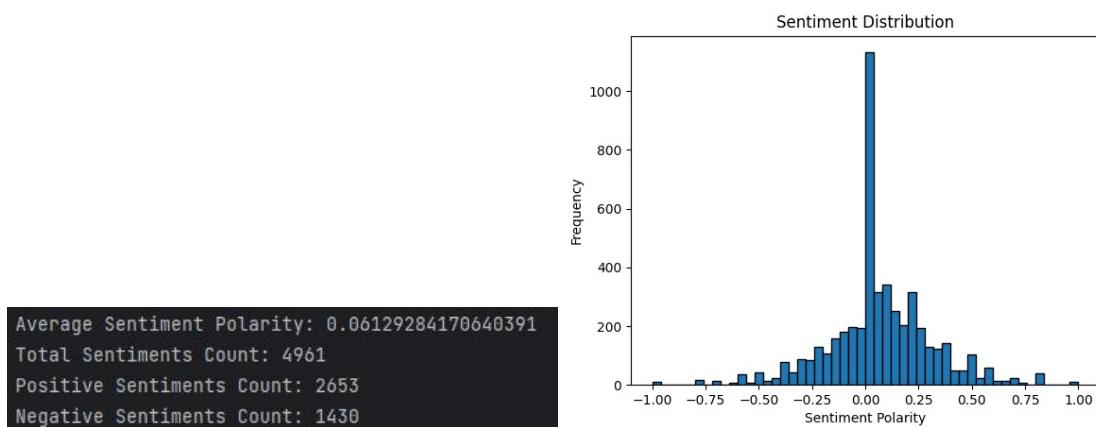
- אוטיות קטנות – המרת כל האותיות לאותיות קטנות. אין לנו ערך להבחנה בין אוטיות גדולות לקטנות בניתוח שלנו, למשל נרצה שהמילים "drake" ו-"Drake" יתפסו כזאות.
 - Tokenization – פירוק וחלוקת הטקסט לאבני בניין בעלות משמעות "tokens".
 - שלב מיוחד – שמירה על המילה `not`. לפני שנסיר `stop words` אנחנו רוצים לשמור על המילה `not` בগল הסנטימנט שלה וגם בגלל השיר "`us not like us`" שהוא השיר המפורסם ביותר ש肯דريك כתב על דרייק. אנחנו מוצפים לראות הרבה התייחסות ברדייט לשיר, לכן גם נצף את המילים כך "`us_not_like_us`" כדי לשמור על המשמעות שלהן.
 - הסרת `stop words` – הורדת מילים נפוצות וחסרות משמעות כמו "`the`", "`and`", "`is`".
עזר לנו להתמקד בעיקר – במילים בעלות מסר וסנטימנט.
 - Lemmatizing – החזרת מילים לצורת המקור שלהם – הורדה של הטיה, רכיבים וסיומות מקובלות בשפה האנגלית שאינן מוסיפות משמעות למילה.
 - שלב מיוחד – זיהינו שככל הופסרים הסמל של גרש ('') הוחלף ברכף האותיות `•` החלנו להסיר את רצף האותיות זהה בשלב העיבוד המוקדם.

ניתוח סנטימנט

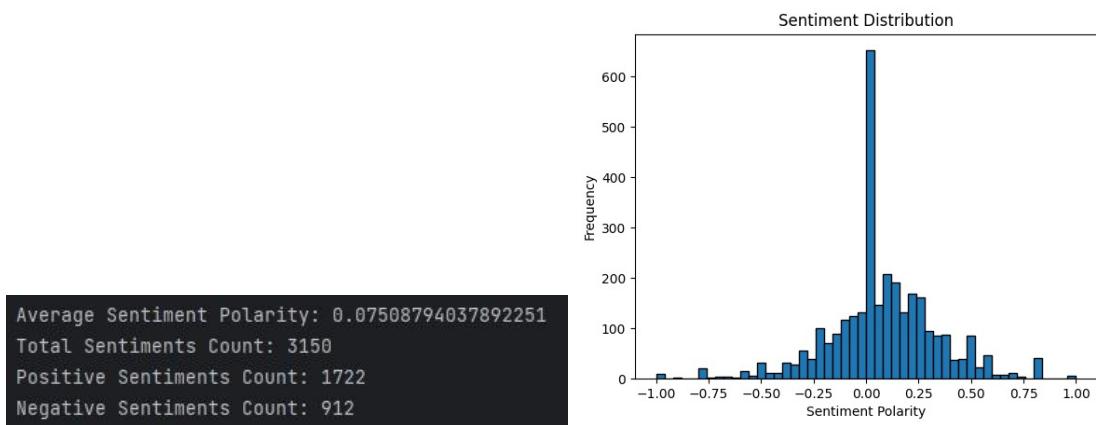
ביצענו ניתוח סנטימנט בעזרת ספריית TextBlob כפי שנלמד בכיתה. הפונקציה שהפעלנו מחשבת עבור כל פוסט בקורס סנטימנט בין -1 (שלילי) ל 1 (חיובי).
חרנו וראינו שהמודל של TextBlob מואמן בעיקר על ביקורת סרטים, כלומר הוא מואמן היטב על סlang אנגלית ועל רגשות חיוביים ושליליים. לכן אנחנו מעריכים שיעד לסוג היבט את הפוסטים שלנו מכיוון שהם מכילים סlang.

תוצאות:

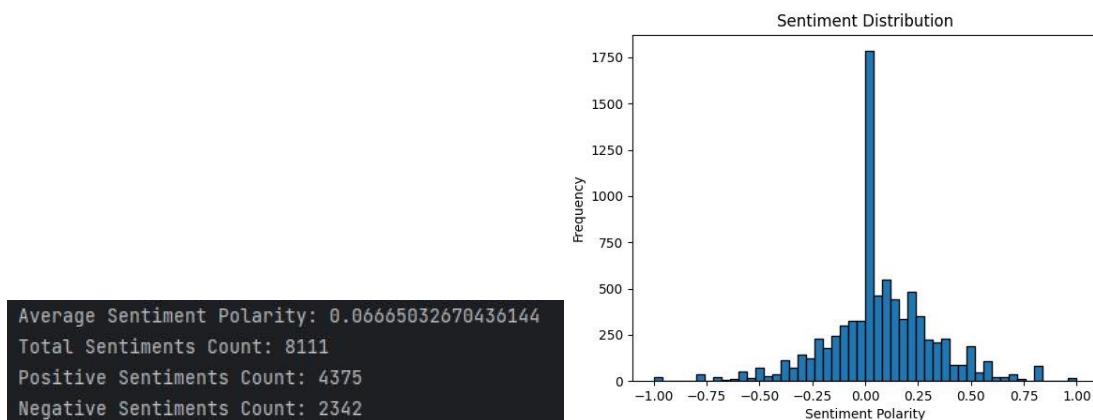
- בקורס של הפוסטים שמדוברים על זרייק (מתוך הסאכדייט של קנדז'יק):



- בקורס של הפוסטים שמדוברים על קנדז'יק (מתוך הסאכדייט של דרייק):



- זאת התפלגות הכללית של הפוסטים על דרייק וקנדרייק ביחד:



מסקנות:

- **הטקסטים שמדוברים על קנדרייק חיוביים יותר מהtekסטים שמדוברים על דרייק!**
 (סנטימנט 0.075 על קנדרייק לעומת סנטימנט 0.061 על דרייק)
- הסנטימנט הכללי של הטקסטים שלנו נראה ניטרי מואוד (0.067) אף על פי שמספר הטקסטים שנמצאו חיוביים גדול כמעט פי 2 ממספר הטקסטים שנמצאו שליליים.
 זאת מושם שמרכיבת הפוסטים סוגרנטרליים והם "מושכים" לכיוון האפס, כפי שנראה בהתפלגות.
- ציפינו שהמודל של TextBlob יידע לסוג טוב יותר את הסנטימנטים ואנחנו מאוכזבים לגלות שמרכיבת הפוסטים דוחה כנטרליים בנושא כל כך טען ורגשי.
 להבא לא נתלה תקוותנו בספרייה חינמית.

מילות מפתח

תחילה, אלה מילים שאנו מוצאים למקומות בשפה:

- Diss – התבאות של אומן כלפי אומן אחר כדי "לרדת" עליו. אצל רארפרים זה מתייחס במיוחד באמצעות שירים שהם כתובים אחד על השני.
- FAN – ראש תיבות ל "F**ker" – כינוי גנאי של קנדרייק כלפי דרייק. קנדרייק טוען שדרייק הוא מוזר ו "שונה מアイינו". קנדרייק נהוג להציג את דרייק כشخص מהככל.
- us_not_like – שם השיר המפורסם ביותר ב"ביפ" – קנדרייק צוחק בו על דרייק שהוא שונה מוכלים ו- "FAN".
- Minor / Pedophile – קנדרייק מכנה את דרייק פדופיל כי הוא יוצא עם קטינות ואפילו שר על זה.
- Short – דרייק צוחק על קנדרייק שהוא נמוך.
- Beat – דרייק טוען על קנדרייק שהוא מכח את אישתו (לא מבוסס מציאות).
- Lie (על כל הטענות שלה) – המונח מהшиб ב"ביפ" סובב סביב שקרים וסקרנים.
- Daughter – יש דבר חזק על זה שלדרייק יש בת שהוא נטש והחביא מהציבור כל השניים (גם זה לא ממש מבוסס מציאות).
- Liebel / SOO – חברות התקליטים של דרייק, שקנדרייק טוען שלוקחת לו 50% מהרווחים.
- Control – שם השיר של ביג שון שבו קנדרייק התחיל את ה"ביפ" עם דרייק.

כדי למצוא מילים מפתח השתמשנו בפונקציה Vectorizer שהופכת את כל המילים לקטור – שיטת Bag Of Words. הפונקציה מבצעת ספירה פשוטה של המילים וממחישה את המילים הבולטות ביותר בטקסט.

אחרי הרצת הקוד התקבל הפלט הבא:

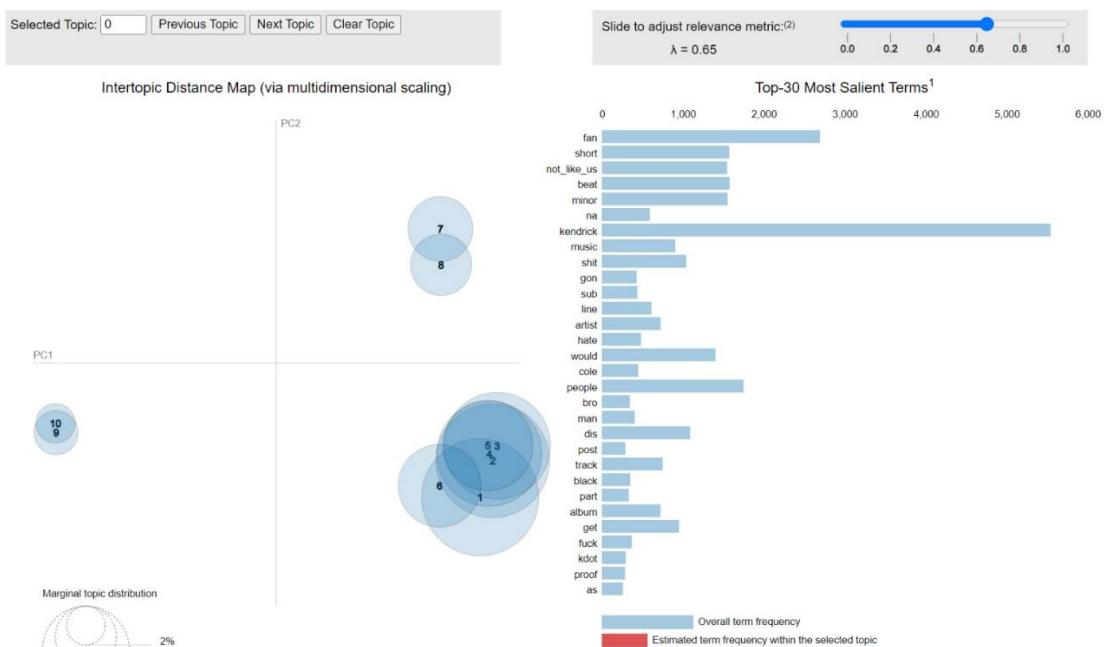
```
Top Key Words: ['also', 'beat', 'beef', 'dis', 'drake', 'even', 'fan', 'get', 'got', 'kendrick', 'know', 'like', 'make', 'minor', 'music', 'one', 'people', 'really', 'said', 'say', 'saying', 'shit', 'short', 'song', 'thing', 'think', 'time', 'track', 'way', 'would']
```

בנוסף לשמותיהם של הרארפרים, ניתן למצוא בין מילים מפתח:

זה מאוד מתאים למה ששיעורנו!

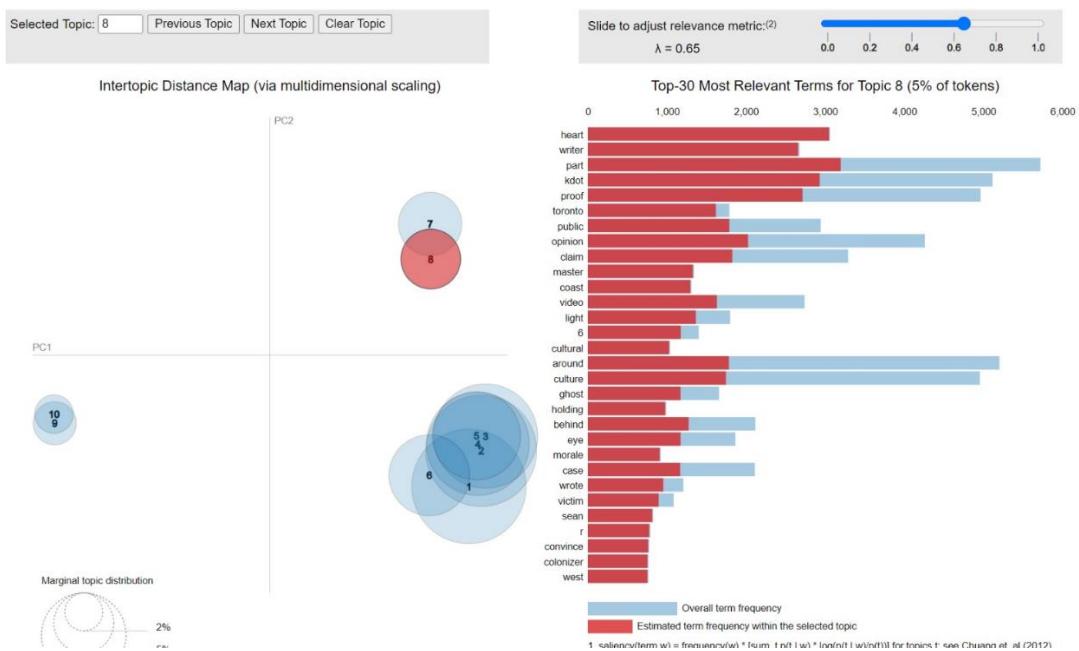
ניתוח נושאי

עבור מידול הנושאים השתמשנו באלגוריתם LDA שנלמד בכיתה, מספרייה שנקראות. השיטה LdaModel יצרה פלט של עמוד HTML שבו מוצגת החלוקה לנושאים Gensim. באופן ויזואלי.



- בצד שמאל ניתן לראות את הנושאים השונים שנמצאו לפי גודל ודימויו זה זהה – בחרנו שרירותית לחלק ל-10 נושאים, אולם קל להזות בגרף שקיימים 3 נושאי על.
- בצד ימין ניתן לראות את 30 המילים הנפוצות בכל נושא יחד עם ההתפלגות הכללית שלהן בין כל המילים.
- הפרמטר למדה (λ) מייצג את היחס בין כמה המילה תזרה בתוך הנושא לעומת התדרות הכללית שלה. אין ערך אופטימלי אוניברסלי עבור למדה, אבל מקובל להשתמש בערך בין 0.6 ל-0.7 – لكن בחרנו 0.65.

כך נראה תצוגה כאשר "עומדים" עם העכבר על אחד הנושאים שהתקבלו:



פוסטים רבים מכילים סלנג וקללות אשר לא תורמים להבנת הנושאים מתוך הטקסט, אולם ניתן להזמין 3 נושאי על:

1. **שיח אלים ו"ירידות" בין שני המחנות:** (נושאים 9-10)

מילים בולטות לפי סדר הופעתן [**'not_like_us'**, '**'fan'**', '**'minor'**', '**'beat'**', '**'short'**]

(מעניין לראות ש3 המילים הראשונות צוחקות על דרייק ו-2 המילים האחרונות צוחקות על קנדרייק. זה עולה בקנה אחד עם הטענה שלנו שקנדרייק הוא המלך!)
2. **שיח מקצועי "נטו":** (נושאים 1-6)

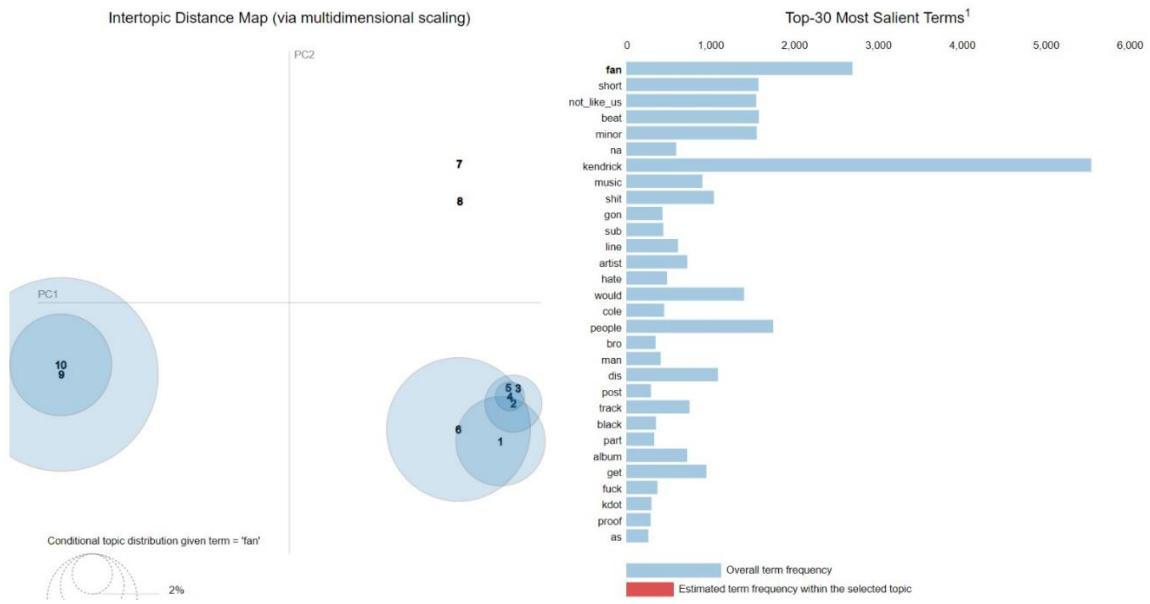
[**'music'**', '**'artist'**', '**'album'**', '**'rap'**', '**'make'**', '**'think'**', '**'like'**]

כנראה פוסטים שמדוברים על מוזיקה חדשה שיוצאת, מאופיינים במילים עם סנטימנט חיובי ואין הרבה קללות.
3. **פסים אישיים מן העבר (לא אלימים):** (נושאים 7-8)

[**'heart'**', '**'writer'**', '**'kdot'**', '**'proof'**', '**'toronto'**', '**'opinion'**', '**'NA'**', '**'cultural'**]

ניתן למצוא כאן את טורונטו (עיר הולדתו של דרייק), K.Dot (הכינוי היישן של קנדרייק) יסודות התצוגה שמספק ה-HTML מסויימות לנו לחזק את הטענה שהشيخ בנושא זה אינו אלים. אם "עומדים" על מילה מסוימת עם העכבר ניתן לראות באיזה נושאים היא מופיעה. כשאנו מצביעים על מילים אלימות וקללות ניתן לראות שלא מופיעות כלל בנושא זה (נקודות 7-8).

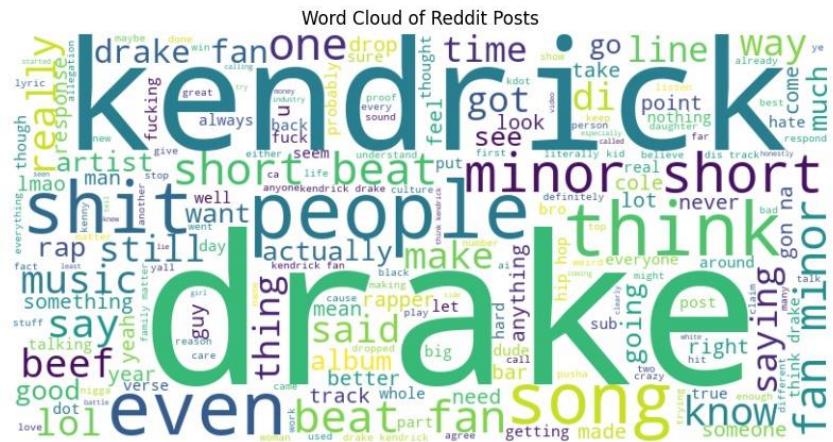
לדוגמה בחירת המילה FAN (שמייצגת עלבון על דרייק):



ענין מילאים

יצרנו ענן מילים באמצעות הספרייה WordCloud כמו שנלמד בכיתה.

ען מיל'ם ראשוני:



הקורפוס שלנו בכוונה מרכיב רק פוסטים שמכילים את המילים "kendrick" או "drake" ולכך הhoeפה שלhn בענן המילים לא תורמת לנו דבר.

נרצה לסנן החוצה את המילים "drake" ו- "kendrick" כדי לתת מקום לביטוי של מושגים אחרים.



חקרונו כמה מילים מופיעות פעמיים בענן שיצרנו. מדובר שכשאין מספיק מילים שחווזרות על עצמן מספיק כדי להופיע בענן, הפונקציה מכפילה מילים תדיות כדי למלא את המסר.

דזוקא את הביטוי "us_like_not" אנחנו לא מצליחים ליצור במלואו – ניתן לראות שענן המילים סופר את המילים בנפרד ו"like" מקבלת מקום מרכזי.

מסקנות:

- ניתן לראות בכירור בענן את המילים שנמצאו כמיילות מפתח בקורפוס: .FAN, short, beat, minor
- ענן המילים הוא טבעי ולא בעל סנטימנט ברור (למרות שניתן למצאו בו מילים בעלות סנטימנט שלילי, כמו קללות).
- בעזרה ענן המילים גילינו גם ביטויים שלא הכרנו – כמו "kdot". מחקר ברדיט גילינו ש"Dot.K" היה שם של קנדרייק למאיר כשהיה רابر מתחילה בשנות ה2000.

4. נספחים

קוד להורדת הנתונים מרדית באמצעות API לקובץ אקסל של קשרות וצמתיים:

```
import praw
import re
import networkx as nx
import csv
from collections import defaultdict

Initialize PRAW with your credentials #
)reddit = praw.Reddit
,"client_id="eurM_n9rWag6gquXPgSVwg
,"client_secret="7t0wCpR3TxqW5jFSy0W_-L56_k7rRg
,"user_agent="Amir_Michal by Michal_Amir
,"username="Michal_Amir
"password="12qw3456
(
Function to collect posts and comments #
:def collect_data(subreddit_name, query, limit=100)
subreddit = reddit.subreddit(subreddit_name)
[] = data
:for submission in subreddit.search(query, limit=limit)
submission.comments.replace_more(limit=0)
:()for comment in submission.comments.list
))data.append
,'author': comment.author.name if comment.author else 'deleted'
{text': comment.body'
,parent_id': comment.parent_id'
,submission_id': submission.id'
 subreddit': subreddit_name # Add the subreddit field'
({
```

```
return data

Collect data from relevant subreddits #
data_kl = collect_data('kendricklamar', 'Kendrick', limit=200)
data_drake = collect_data('Drizzy', 'Drake', limit=200)
```

```
Combine the data from both subreddits #
data = data_kl + data_drake
```

```
Clean the text #
:def clean_text(text)
()text = text.lower
text = re.sub(r'\s+', ' ', text) # Remove extra spaces
text = re.sub(r'[^w\s]', "", text) # Remove punctuation
return text

:for entry in data
entry['text'] = clean_text(entry['text'])
```

```
Construct the network #
()G = nx.Graph
```

```
Add nodes #
:for entry in data
author = entry['author']
:'if author != 'deleted'
G.add_node(author)
```

```
Create edges based on thread participation and parent-child relationships #
{} = submission_dict
```

```
Group comments by submission #
:for entry in data
    submission_id = entry['submission_id']
    :if submission_id not in submission_dict
        [] = submission_dict[submission_id]
        submission_dict[submission_id].append(entry)
```

```
Create edges for users in the same submission thread #
edge_weights = defaultdict(int)
:()for submission_id, comments in submission_dict.items
    authors = [comment['author'] for comment in comments if comment['author'] != 'deleted']
    :for i in range(len(authors))
        :for j in range(i + 1, len(authors))
            :if authors[i] != authors[j]
                edge_weights[(authors[i], authors[j])] += 1
```

```
Add edges with weight #
:()for (author1, author2), weight in edge_weights.items
    G.add_edge(author1, author2, weight=weight)
```

```
Filter to get around 50 nodes #
[50:]nodes_to_keep = list(G.nodes)
H = G.subgraph(nodes_to_keep)
```

```
Filter edges to reduce their number, keep the top weighted edges #
edges = sorted(H.edges(data=True), key=lambda x: x[2]['weight'], reverse=True)
edges_to_keep = edges[:len(edges)//2] # Keep only the top half edges
```

```
Create a new graph with the filtered edges #
():H_filtered = nx.Graph
```

```
H_filtered.add_nodes_from(H.nodes(data=True))

H_filtered.add_edges_from([(u, v) for u, v, w in edges_to_keep])
```

```
Analyze the network #

Degree Centrality #

degree_centrality = nx.degree_centrality(H_filtered)
```

```
Betweenness Centrality #

betweenness_centrality = nx.betweenness_centrality(H_filtered)
```

```
Define the file paths #

'nodes_file_path = 'C:/Users/yativ/OneDrive/Desktop/nodes2.csv'
'edges_file_path = 'C:/Users/yativ/OneDrive/Desktop/edges2.csv'
```

```
Export Nodes #

:with open(nodes_file_path, 'w', newline='') as f
writer = csv.writer(f)

writer.writerow(['Id', 'Label', 'DegreeCentrality', 'BetweennessCentrality',
'Subreddit'])

:for node in H_filtered.nodes

Find the subreddit for the node #

subreddits = set(entry['subreddit'] for entry in data if entry['author'] == node)
subreddit_list = ', '.join(subreddits)

writer.writerow([node, node, degree_centrality[node],
betweenness_centrality[node], subreddit_list])
```

```
Export Edges #

:with open(edges_file_path, 'w', newline='') as f
writer = csv.writer(f)

writer.writerow(['Source', 'Target'])

:for edge in H_filtered.edges
```

```
writer.writerow([edge[0], edge[1]])
```

קוד לטעינת האקסלים כגרף לפיזון:

```
import pandas as pd
```

```
import networkx as nx
```

```
Load your nodes and edges CSV files #
```

```
nodes_df = pd.read_csv("C:/Users/yativ/OneDrive/Desktop/new nodes.csv")
```

```
edges_df = pd.read_csv("C:/Users/yativ/OneDrive/Desktop/edges2.csv")
```

```
Print column names to verify #
```

```
print("Nodes CSV columns:", nodes_df.columns)
```

```
print("Edges CSV columns:", edges_df.columns)
```

```
Define the correct column names based on your CSV files #
```

```
'node_id_col = 'Id
```

```
'node_label_col = 'Label
```

```
'node_subreddit_col = 'Subreddit
```

```
'edge_source_col = 'Source
```

```
'edge_target_col = 'Target
```

```
Create a graph #
```

```
()G = nx.Graph
```

```
Add nodes with attributes #
```

```
:()for index, row in nodes_df.iterrows
```

```
G.add_node(row[node_id_col], label=row[node_label_col],  
subreddit=row[node_subreddit_col])
```

```
Add edges #
```

```
:()for index, row in edges_df.iterrows
```

```
G.add_edge(row[edge_source_col], row[edge_target_col])
```

```
Find articulation points #
articulation_points = list(nx.articulation_points(G))
print("Articulation points:", articulation_points)
```

```
Mark articulation points in the graph #
:()for node in G.nodes
G.nodes[node]['articulation_point'] = node in articulation_points
```

```
Save the modified graph back to a file #
nx.write_gexf(G, "C:/Users/yativ/OneDrive/Desktop/iWish1.gexf")
```

מדי המרכזיות מתוך פיתון:

```
import networkx as nx
import numpy as np

"gexf_file = "C:/Users/yativ/OneDrive/Desktop/iWish1.gexf"
:def load_graph_from_gexf(gexf_file)
return nx.read_gexf(gexf_file)

:def sort_centrality_measures(centrality_measures)
{} = sorted_measures
:()for measure, values in centrality_measures.items
sorted_measures[measure] = sorted(values.items(), key=lambda item: item[1],
reverse=True)
return sorted_measures

:def measure_centrality(graph)
{} = centrality_measures
```

```
Degree Centrality #
centrality_measures['degree_centrality'] = nx.degree_centrality(graph)

Betweenness Centrality #
centrality_measures['betweenness_centrality'] = nx.betweenness_centrality(graph)

Closeness Centrality #
centrality_measures['closeness_centrality'] = nx.closeness_centrality(graph)

return centrality_measures

:Example usage #
:"if name == "main
File path #

Load graph from GEXF file #
G = load_graph_from_gexf(gexf_file)

Measure centrality indices #
centrality_indices = measure_centrality(G)

Print centrality indices #
sorted_centrality_indices = sort_centrality_measures(centrality_indices)

Print sorted centrality indices #
:()for measure, values in sorted_centrality_indices.items
print(f"{measure}: {values}\n")
```

פלט המרכזיות מטור פיתון:

```
degree_centrality: [('jfarm47', 0.999999999999999), ('KingDinkleber6',  
0.9795918367346939), ('BangPowZoom', 0.9591836734693877), ('ed4723',  
0.9591836734693877), ('osnapitzsunnyy', 0.9591836734693877), ('YizWasHere',  
0.9591836734693877), ('its-a-real-name', 0.9591836734693877), ('ricemonkey13',  
0.9387755102040816), ('TheBlackPope', 0.9387755102040816), ('mikelima777',  
0.8979591836734693), ('Ronald_Reagan_Era', 0.8775510204081632), ('Bopfire',  
0.7959183673469387), ('nomoreshiny', 0.673469387755102), ('Worldly-Pudding7992',  
0.6530612244897959), ('gloomygl', 0.6326530612244897), ('chizzmaster',  
0.6122448979591836), ('believe-what-i-say', 0.5918367346938775), ('idkwtvr04',  
0.4693877551020408), ('Dracouer', 0.4693877551020408), ('LelouchNexus',  
0.42857142857142855), ('Salvincent', 0.3877551020408163), ('mertis0420',  
0.3877551020408163), ('KxrpzeyT', 0.36734693877551017), ('solythe',  
0.36734693877551017), ('TOP-IS-LIFE', 0.36734693877551017),  
('Soviet_Sharpshooter', 0.36734693877551017), ('LeonSnakeKennedy',  
0.3469387755102041), ('Brokeninfo', 0.3469387755102041), ('Hu3yKnewTHen',  
0.32653061224489793), ('PaTaPaChiChi', 0.32653061224489793), ('Fancy-Guarantee-  
7782', 0.32653061224489793), ('whodathunkitwasme', 0.32653061224489793),  
('ktran237', 0.3061224489795918), ('HowDolSwag', 0.3061224489795918), ('pawpet',  
0.3061224489795918), ('Nanthro', 0.3061224489795918), ('AWall925',  
0.3061224489795918), ('newepsonprinter', 0.3061224489795918),  
('the_mighty_monarch', 0.3061224489795918), ('trailerthrash', 0.3061224489795918),  
('dylanh334', 0.2857142857142857), ('luckygitane', 0.2857142857142857),  
('GuitarStuffThrowaway', 0.26530612244897955), ('GimmeHardyHat',  
0.26530612244897955), ('Gabagool_Over_Here_', 0.26530612244897955),  
('northernpace', 0.24489795918367346), ('ridingonmirrors', 0.24489795918367346),  
('JustMeAndMyKnickas', 0.22448979591836732), ('Teqnition12',  
0.22448979591836732), ('uncomfy1234', 0.22448979591836732)]
```

```
betweenness_centrality: [('jfarm47', 0.043355285079189884), ('KingDinkleber6',  
0.04046261766584579), ('osnapitzsunnyy', 0.040316025873116126), ('YizWasHere',  
0.03999410671801151), ('BangPowZoom', 0.03902654789037128), ('ed4723',  
0.03902654789037128), ('its-a-real-name', 0.03902654789037128), ('ricemonkey13',  
0.03718540321149368), ('TheBlackPope', 0.03667013163089378), ('mikelima777',  
0.035455194736936084), ('Ronald_Reagan_Era', 0.026362956734820715), ('Bopfire',  
0.02299671393924458), ('believe-what-i-say', 0.01567176063880247), ('Worldly-  
Pudding7992', 0.011857959125519783), ('nomoreshiny', 0.011830730248402333),  
('gloomygl', 0.009646349303990994), ('chizzmaster', 0.007385548634853373),  
('LelouchNexus', 0.005667279950423224), ('Dracouer', 0.0036996724362220726),  
('idkwtvr04', 0.0031154213895606122), ('Salvincent', 0.001063537396042207),  
('mertis0420', 0.001063537396042207), ('Soviet_Sharpshooter',  
0.0009500956530120322), ('the_mighty_monarch', 0.0007231267435701335),
```

('LeonSnakeKennedy', 0.0006456387832678175), ('KxrPzeYT', 0.0006375312503797776), ('solythe', 0.0006375312503797776), ('TOP-IS-LIFE', 0.0006375312503797776), ('Hu3yKnewTHen', 0.0005956187752646162), ('Fancy-Guarantee-7782', 0.0005956187752646162), ('GuitarStuffThrowaway', 0.0004648216001324659), ('northernpace', 0.00043739127316296846), ('GimmeHardyHat', 0.00043739127316296846), ('Nanthro', 0.0004032064460601837), ('whodathunkitwasme', 0.0003768057339485911), ('luckygitane', 0.00035233636130891586), ('Brokeninfo', 0.00033356114016820524), ('newepsonprinter', 0.0002201193971380307), ('trailerthrash', 0.0002201193971380307), ('ridingonmirrors', 0.00019767759320476962), ('Gabagool_Over_Here_', 0.0001924433442730417), ('ktran237', 0.00018399538501238805), ('pawpet', 0.00018399538501238805), ('PaTaPaChiChi', 0.00018399538501238805), ('AWall925', 0.00018399538501238805), ('JustMeAndMyKnickas', 0.00014765758520156831), ('Teqnition12', 0.00014765758520156831), ('uncomfy1234', 0.00014765758520156831), ('HowDolSwag', 7.05536419822134e-05), ('dylanh334', 7.05536419822134e-05)]

closeness_centrality: [('jfarm47', 1.0), ('KingDinkleber6', 0.98), ('BangPowZoom', 0.9607843137254902), ('ed4723', 0.9607843137254902), ('osnapitzsunny', 0.9607843137254902), ('YizWasHere', 0.9607843137254902), ('its-a-real-name', 0.9607843137254902), ('ricemonkey13', 0.9423076923076923), ('TheBlackPope', 0.9423076923076923), ('mikelima777', 0.9074074074074074), ('Ronald_Reagan_Era', 0.8909090909090909), ('Bopfire', 0.8305084745762712), ('nomoreshiny', 0.7538461538461538), ('Worldly-Pudding7992', 0.7424242424242424), ('gloomygl', 0.7313432835820896), ('chizzmaster', 0.7205882352941176), ('believe-what-i-say', 0.7101449275362319), ('idkwtvr04', 0.6533333333333333), ('Dracouer', 0.6533333333333333), ('LelouchNexus', 0.6363636363636364), ('Salvincent', 0.620253164556962), ('mertis0420', 0.620253164556962), ('KxrPzeYT', 0.6125), ('solythe', 0.6125), ('TOP-IS-LIFE', 0.6125), ('Soviet_Sharpshooter', 0.6125), ('LeonSnakeKennedy', 0.6049382716049383), ('Brokeninfo', 0.6049382716049383), ('Hu3yKnewTHen', 0.5975609756097561), ('PaTaPaChiChi', 0.5975609756097561), ('Fancy-Guarantee-7782', 0.5975609756097561), ('whodathunkitwasme', 0.5975609756097561), ('ktran237', 0.5903614457831325), ('HowDolSwag', 0.5903614457831325), ('pawpet', 0.5903614457831325), ('Nanthro', 0.5903614457831325), ('AWall925', 0.5903614457831325), ('newepsonprinter', 0.5903614457831325), ('the_mighty_monarch', 0.5903614457831325), ('trailerthrash', 0.5903614457831325), ('dylanh334', 0.583333333333334), ('luckygitane', 0.583333333333334), ('GuitarStuffThrowaway', 0.5764705882352941), ('GimmeHardyHat', 0.5764705882352941), ('Gabagool_Over_Here_', 0.5764705882352941), ('northernpace', 0.5697674418604651), ('ridingonmirrors', 0.5697674418604651), ('JustMeAndMyKnickas', 0.5632183908045977), ('Teqnition12', 0.5632183908045977), ('uncomfy1234', 0.5632183908045977)]

הורדת נתונים עבור ניתוח רשותות:

```
import praw
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from textblob import TextBlob
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

Initialize Reddit API (replace 'your_client_id', 'your_client_secret', 'your_user_agent' # with actual values)

```
)reddit = praw.Reddit
, "client_id="eurM_n9rWag6gquXPgSVwg
, "client_secret="7t0wCpR3TxqW5jFSy0W_-L56_k7rRg
, "user_agent="Amir_Michal by Michal_Amir
, "username="Michal_Amir
"password="12qw3456
()
```

Define a function to collect comments and posts #

```
:def collect_reddit_data(subreddit_name, search_query, limit=1000)
subreddit = reddit.subreddit(subreddit_name)
posts = subreddit.search(search_query, limit=limit)
```

```
[] = data
```

```
:for post in posts
```

```
if post.is_self: # Check if the post is a text post
    data.append({'title': post.title, 'body': post.selftext, 'score': post.score})
    post.comments.replace_more(limit=0)
    for comment in post.comments.list:
        data.append({'title': None, 'body': comment.body, 'score': comment.score})

return pd.DataFrame(data)
```

```
Collect data from Kendrick Lamar's subreddit #
kendrick_data = collect_reddit_data('KendrickLamar', 'Drake', limit=200)
```

```
Collect data from Drake's subreddit (Drizzy) #
drake_data = collect_reddit_data('Drizzy', 'Kendrick', limit=200)
```

```
Save to CSV files #
kendrick_data.to_csv('kendrick_data.csv', index=False)
drake_data.to_csv('drake_data.csv', index=False)
```

```
print("Data collection complete. Files saved as 'kendrick_data.csv' and
'drake_data.csv'.")
```

```
Define preprocessing functions #
def preprocess_text(text):
    if text is None:
        """
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
        text = re.sub(r'@\w+|\#', '', text)
        text = text.lower()
        text = re.sub(r'\d+', '', text)
        text = re.sub(r'^\w\s', '', text)
    return text
```

```

:def tokenize_text(text)
    return word_tokenize(text)

:def remove_stopwords(tokens)
    stop_words = set(stopwords.words('english'))
    return [word for word in tokens if word not in stop_words]

:def lemmatize_tokens(tokens)
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(token) for token in tokens]

Preprocess the collected data # #
nltk.download('punkt') #
nltk.download('stopwords') #
nltk.download('wordnet') #

kendrick_data['cleaned_body'] = kendrick_data['body'].apply(preprocess_text)
kendrick_data['tokens'] = kendrick_data['cleaned_body'].apply(tokenize_text)
kendrick_data['tokens'] = kendrick_data['tokens'].apply(remove_stopwords)
kendrick_data['tokens'] = kendrick_data['tokens'].apply(lemmatize_tokens)

drake_data['cleaned_body'] = drake_data['body'].apply(preprocess_text)
drake_data['tokens'] = drake_data['cleaned_body'].apply(tokenize_text)
drake_data['tokens'] = drake_data['tokens'].apply(remove_stopwords)
drake_data['tokens'] = drake_data['tokens'].apply(lemmatize_tokens)

Display the preprocessed data #
print(kendrick_data.head())
print(drake_data.head())

```

```

:def get_sentiment(text)
    return TextBlob(text).sentiment.polarity

kendrick_data['sentiment'] = kendrick_data['cleaned_body'].apply(get_sentiment)
drake_data['sentiment'] = drake_data['cleaned_body'].apply(get_sentiment)

Display sentiment analysis results #
print(kendrick_data[['cleaned_body', 'sentiment']].head())
print(drake_data[['cleaned_body', 'sentiment']].head())

Vectorize the text data #
count_vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
kendrick_count_data = count_vectorizer.fit_transform(kendrick_data['cleaned_body'])
drake_count_data = count_vectorizer.fit_transform(drake_data['cleaned_body'])

Fit LDA model #
lda = LatentDirichletAllocation(n_components=10, random_state=0)
kendrick_lda = lda.fit(kendrick_count_data)
drake_lda = lda.fit(drake_count_data)

Display the top words for each topic #
:def display_topics(model, feature_names, no_top_words)
:for topic_idx, topic in enumerate(model.components_)
    print("Topic %d:" % (topic_idx))
    print(" ".join([feature_names[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))

no_top_words = 10
print("Kendrick Data Topics:")
display_topics(kendrick_lda, count_vectorizer.get_feature_names_out(), no_top_words)
print("Drake Data Topics:")
display_topics(drake_lda, count_vectorizer.get_feature_names_out(), no_top_words)

```

```

Word Cloud #

all_kendrick_words = ' '.join([text for text in kendrick_data['cleaned_body']])
all_drake_words = ' '.join([text for text in drake_data['cleaned_body']])

kendrick_wordcloud = WordCloud(width=800, height=500, random_state=21,
max_font_size=110).generate(all_kendrick_words)

drake_wordcloud = WordCloud(width=800, height=500, random_state=21,
max_font_size=110).generate(all_drake_words)

plt.figure(figsize=(10, 7))
plt.imshow(kendrick_wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('Word Cloud for Kendrick Data')
()plt.show

plt.figure(figsize=(10, 7))
plt.imshow(drake_wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('Word Cloud for Drake Data')
()plt.show

Sentiment Distribution #

plt.figure(figsize=(10, 6))
plt.hist(kendrick_data['sentiment'], bins=50, color='blue', edgecolor='black')
plt.title('Sentiment Distribution for Kendrick Data')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')
()plt.show

plt.figure(figsize=(10, 6))

```

```

plt.hist(drake_data['sentiment'], bins=50, color='blue', edgecolor='black')
plt.title('Sentiment Distribution for Drake Data')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')
()plt.show

```

עיבוד מילים:

```

import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

```

Download required NLTK data files # #

```

nltk.download('punkt') #
nltk.download('stopwords') #
nltk.download('wordnet') #

```

File paths (replace with your actual file paths) #

```

file_paths = ['drake_data.csv', 'kendrick_data.csv']
output_paths = ['processed_file1.csv', 'processed_file2.csv']

```

Preprocess the text data #

```
:def preprocess_text(text)
```

Convert to lower case #

```
()text = text.lower
```

Tokenize #

```
tokens = word_tokenize(text)
```

Remove stop words #

```
stop_words = set(stopwords.words('english'))
```

```

tokens = [word for word in tokens if word.isalnum() and word not in stop_words]

Lemmatize #
()lemmatizer = WordNetLemmatizer

tokens = [lemmatizer.lemmatize(word) for word in tokens]
return ''.join(tokens)

:for file_path, output_path in zip(file_paths, output_paths)

Read CSV file #
df = pd.read_csv(file_path)

Apply preprocessing to the 'body' column #
df['processed_body'] = df['body'].apply(preprocess_text)

Save the processed data to a new CSV file #
df.to_csv(output_path, index=False)

Display the processed data (this line is for debugging purposes, to check the #
processed data)
print(f"Processed data for {file_path}:")
print(df.head())

print(f"Processed data saved to {output_paths}")

```

ניתוח סטטימנט:

```

import pandas as pd
from textblob import TextBlob
import matplotlib.pyplot as plt

Load the data #
file_path = 'Kendrick Posts.csv' # Adjust this path to your file location

```

```
data = pd.read_csv(file_path)

Function to get sentiment polarity #
:def get_sentiment(text)
    return TextBlob(text).sentiment.polarity

Apply sentiment analysis #
data['sentiment'] = data['processed_body'].apply(get_sentiment)

Calculate average sentiment #
()average_sentiment = data['sentiment'].mean

Count total sentiments #
total_sentiments = len(data)

Count positive and negative sentiments #
[0]positive_sentiments = data[data['sentiment'] > 0].shape
[0]negative_sentiments = data[data['sentiment'] < 0].shape

Print results #
print("Average Sentiment Polarity:", average_sentiment)
print("Total Sentiments Count:", total_sentiments)
print("Positive Sentiments Count:", positive_sentiments)
print("Negative Sentiments Count:", negative_sentiments)

Display sentiment scores #
print(data[['processed_body', 'sentiment']].head())

Plot sentiment distribution #
plt.hist(data['sentiment'], bins=50, edgecolor='black')
plt.title('Sentiment Distribution')
```

```
plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
()plt.show
```

המילים הנפוצות ביותר:

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
```

Load the data #

```
file_path = 'All Posts like_us.csv' # Adjust this path to your file location
data = pd.read_csv(file_path)
```

Ensure you have the necessary NLTK data # #

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

Preprocess the data #

```
:def preprocess_text(text)
tokens = word_tokenize(text.lower())
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.isalnum() and word not in
stop_words]
return ''.join(filtered_tokens)
```

Apply preprocessing to each post #

```
data['cleaned_text'] = data['processed_body'].apply(preprocess_text)
```

Key words extraction #

```
vectorizer = CountVectorizer(max_features=30)
X = vectorizer.fit_transform(data['cleaned_text'])
```

```
Extract top key words #
()keywords = vectorizer.get_feature_names_out
print("Top Key Words:", keywords)
```

ניתוח נושאי LDA:

```
import pandas as pd
from gensim import corpora
from gensim.models.ldamodel import LdaModel
import pyLDAvis.gensim_models
```

```
Step 1: Load the Data #
'file_path = 'All Posts like_us.csv'
df = pd.read_csv(file_path)
```

```
'Assuming your preprocessed text data is in a column named 'text #
()texts = df['processed_body'].tolist
```

```
Step 2: Prepare the Dictionary and Corpus #
Tokenize the text #
tokenized_texts = [text.split() for text in texts]
```

```
.Create a dictionary representation of the documents #
dictionary = corpora.Dictionary(tokenized_texts)
```

```
Filter out extremes to limit the number of features #
dictionary.filter_extremes(no_below=15, no_above=0.5, keep_n=100000)
```

```
.Create a Bag-of-Words representation of the documents #
```

```
corpus = [dictionary.doc2bow(text) for text in tokenized_texts]
```

Step 3: Build the LDA Model #

```
Set training parameters #
```

```
num_topics = 3 # The number of topics you want to extract
```

```
passes = 10 # Number of passes through the corpus during training
```

Train the LDA model #

```
lda_model = LdaModel(corpus=corpus, id2word=dictionary,  
num_topics=num_topics, passes=passes)
```

Step 4: Visualize the Topics #

```
Prepare the visualization #
```

```
lda_vis = pyLDAvis.gensim_models.prepare(lda_model, corpus, dictionary)
```

Save the visualization to an HTML file #

```
pyLDAvis.save_html(lda_vis, 'lda_visualization.html')
```

```
print("LDA model built and visualization saved as 'lda_visualization.html'.")
```