Lab 3

Pattern Discovery from Cleaned YouTube Text

Course Title: Data Mining

Course Code: CSE 477

Semester: Fall 2025

Section: 01

Submitted by

Name: Nushrat Jaben Aurnima

ID: 2022-2-60-146

Date: 02/11/2025

Submitted to

Amit Mandal

Lecturer, Department of Computer Science and Engineering

East West University

Table of Contents

Introduction

The YouTube text-mining workflow which we developed in this lab continues to operate on clean text to identify new patterns. Through my work with the cleaned comments and captions datasets I built a basket style pipeline which reveals token combinations and advanced relationships between them. The analysis starts with basic frequency analysis of unigrams before moving to co-occurrence graphs and then to frequent itemset mining and association rule analysis which includes support confidence and lift metrics. The robust evaluation included tests that eliminated short tokens while using stemming instead of lemmatization and different minimum support thresholds (0.30, 0.20, 0.10, 0.15, 0.05). The results section contains individual analyses of comments and captions as well as a combined dataset which includes visualizations and CSV files that will serve as input for the following laboratory exercise.

Objective

- Validate cleaned comments and captions datasets to ensure cleaned tokens are listed.
- Convert each token list into a transaction to drop baskets with <3 tokens and remove duplicates.
- Summarize dataset structure with a histogram of basket lengths (avg/min/max).
- Compute unigram frequencies and pairwise co-occurrences filter pairs with count ≥3.
- Visualize patterns with a bar chart (top co-occurring pairs), a unigram bar for comparison, and a network graph of co-occurrences.
- Export co-occurrence data to CSV for reproducibility.
- One-hot encode transactions and run Apriori at min_support = 0.30, 0.20, 0.10, 0.15, 0.05; keep 2- and 3-itemsets.
- Generate association rules; filter by confidence ≥ 0.60 and lift ≥ 1.20; plot support vs. confidence.
- Conduct robustness checks: stemming (vs. lemmatization) and length≥4 token filter.
- Apply the pipeline to captions, comments, and merged data; compare rule patterns across settings.
- Save frequent item sets, rules, and all plots; keep a clean folder for submission and reuse in the next lab

Tools and Libraries

- **Python 3.x** — For running all the code (processing and analysis).
- **pandas** — Creating DataFrames, and many more.
- **numpy** — Used in array operations.
- **collections.Counter & itertools** — `collections.Counter` helps quickly count how many times each item appears in a list or string. `itertools` provide memory-efficient tools for creating and combining iterators
- **matplotlib.pyplot** — used for basic visualizations (histograms, bar charts, scatter plot.)
- **wordcloud** — word cloud visualization from frequent items.
- **networkx** — co-occurrence graph construction and simple network visualization.
- **nltk (stopword, stem)** — For stop word removal and stemming variant check.
- **mlxtend.frequent_patterns** — Apriori and association_rules, thresholds set in `apriori_pipeline()` with `min_support`, `confidence`, and `lift`.
- **mlxtend.preprocessing** — `TransactionEncoder` for one-hot creation.
- **os / glob / pathlib** — For path handling and export of CSVs and PNGs into `OUT_DIR`.
- **gc** — For garbage collection to keep memory stable during repeated Apriori runs.

Dataset Description

The datasets used in this lab are the cleaned versions of the YouTube comments and captions created in the previous lab 2 *(Data Cleaning & Preprocessing for YouTube Text Mining)*. The files `cleaned_comments.csv` and `cleaned_captions.csv` hold text data that underwent tokenization and stopword removal and cleaning procedures. The records contain a column named cleaned_tokens which holds processed word lists that divide individual comments or caption segments. These cleaned text datasets from the previous lab serve as the foundation for pattern discovery in this new laboratory work. The research direction moved away from text cleaning toward investigating how words appear together and how tokens form associations. Each list of tokens was treated as a transaction, like a basket of items. This allowed the application of market basket analysis techniques. The analysis began with separate examination of comments and captions to find common words and token pairs before combining both datasets to detect overlapping and unique linguistic patterns.

Methodology

Setup and Data Recall

The cleaned datasets (*cleaned_comments.csv* and *cleaned_captions.csv*) created in the previous lab were re-used for this analysis. Each dataset was imported into pandas as a DataFrame containing a column named `cleaned_tokens`, which holds tokenized words derived from text preprocessing. The first step was to verify that the column existed and contained valid list-like entries. Rows where this column was missing or empty were removed.

Key Column	**Rows Dropped**	**Language Issues Detected**	**Example of `cleaned_tokens` Entries**
`cleaned_tokens`	38	Few emoji tokens removed during inspection	`['good', 'probably', 'need', 'updated', 'example', 'used', 'spotify', 'amp', 'twilo', 'outdated']['amazing']['ngl', 'giant', 'twilio', 'advertisement']`
`cleaned_tokens`	0	Primarily English text, minor repetitions due to transcript formatting	`['course', 'excellent', 'introduction', 'apis', 'beginner', 'updated', 'version', 'already', 'one', 'popular']['api', 'application', 'programming', 'interface', 'addition', 'able', 'interface', 'addition', 'able']['spotify', 'com', 'open', 'api']`

Table 1. Summary of Data Recall and Token Inspection

After validation, the `collections.Counter` function was applied to the token lists to print the top 20 most frequent unigrams. This provided an overview of the most common terms appearing in each dataset.

#	**comment_token**	**comment_count**	**caption_token**	**caption_count**
1	http	293	going	1878
2	href	289	right	701
3	watch	289	see	641
4	youtube	283	let	620
5	com	275	get	606
6	amp	274	message	567
7	www	272	want	546
8	wxsd	268	api	521
9	zgxjrw	268	like	514
10	api	201	one	484

11	video	138	thing	428
12	quot	126	back	388
13	spotify	116	look	373
14	course	108	make	340
15	twilio	106	say	326
16	like	106	know	322
17	apis	91	use	299
18	thank	87	could	296
19	get	81	got	280
20	using	75	code	265

Table 2: Top 20 most frequent unigrams for comments and captions

The mlxtend package was then installed and tested to ensure that its Apriori and association_rules functions were available for later stages of frequent-pattern mining.

Transaction Construction

Each token list was treated as a single basket (transaction), like the structure used in market basket analysis. Transactions containing fewer than three tokens were excluded to avoid trivial associations. For reference and tracking, a unique transaction ID was assigned to every basket.

Dataset	Total Transactions	Example Basket
Comments	788	`['good', 'probably', 'need', 'updated', 'example', 'used', 'spotify', 'amp', 'twilo', 'outdated']`
Captions	52	`['course', 'excellent', 'introduction', 'apis', 'beginner', 'updated', 'version', 'already', 'one', 'popular', 'api', 'internet', 'craig', 'dennis', 'back', 'instructor', 'opinion', 'best', 'developer', 'educator', ...]`

Table 3: Transaction Summary

Summary statistics including average, minimum, and maximum basket size were also printed to describe dataset density and variety.

Average Length	Minimum Length	Maximum Length
11.87	3	410
101.65	3	932

Table 4: Basket Length Statistics

A histogram of basket lengths was plotted to visualize how many words typically appeared in each transaction.

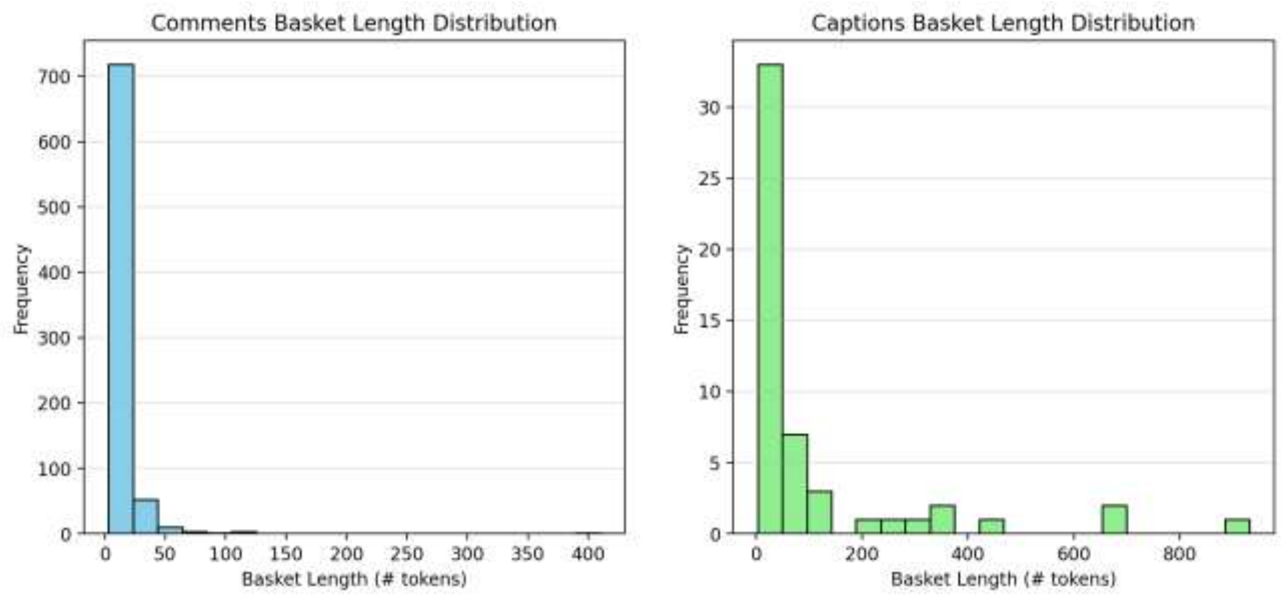

Figure 1: Histogram of basket lengths (comments and captions)

Manual Pattern Discovery

Before using algorithmic methods, manual exploration was conducted to understand token co-occurrence patterns. Pairwise word co-occurrences were counted across all baskets. The top 20-word pairs were visualized using a horizontal bar chart.

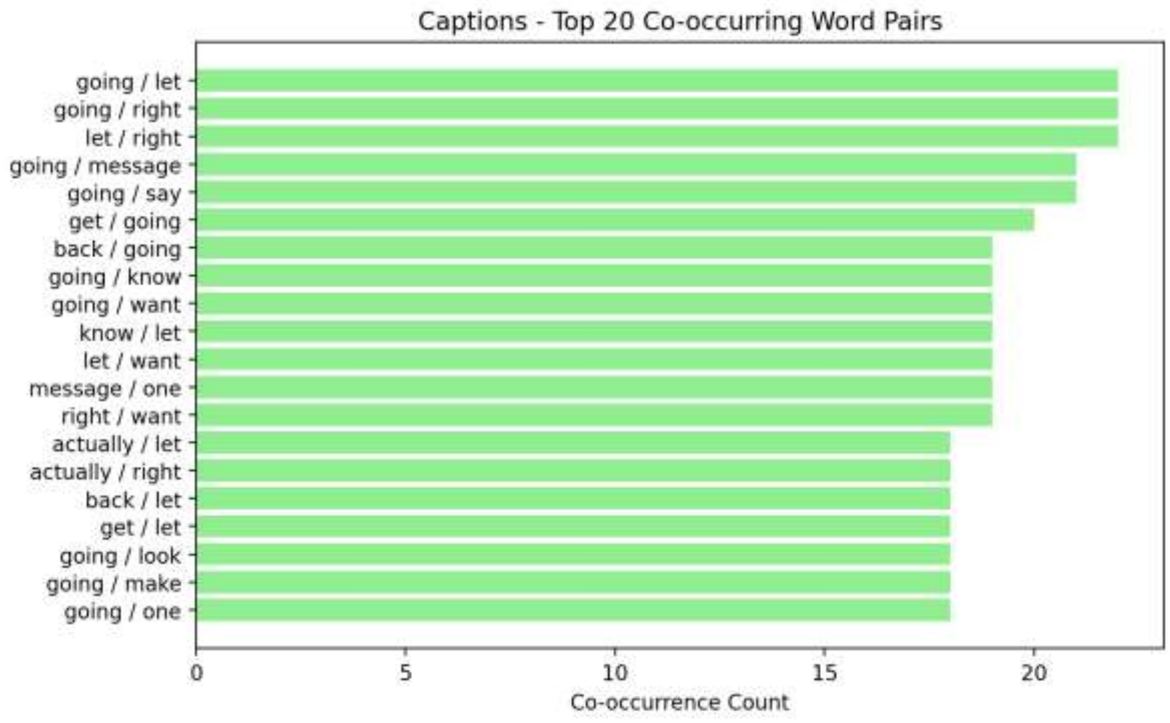

Figure 2: Top 20 co-occurring pairs (captions)

Then by a unigram frequency chart for comparison (removing stopword using *NTLK* Library). This helped identify overlapping patterns between frequent individual tokens and co-occurring pairs

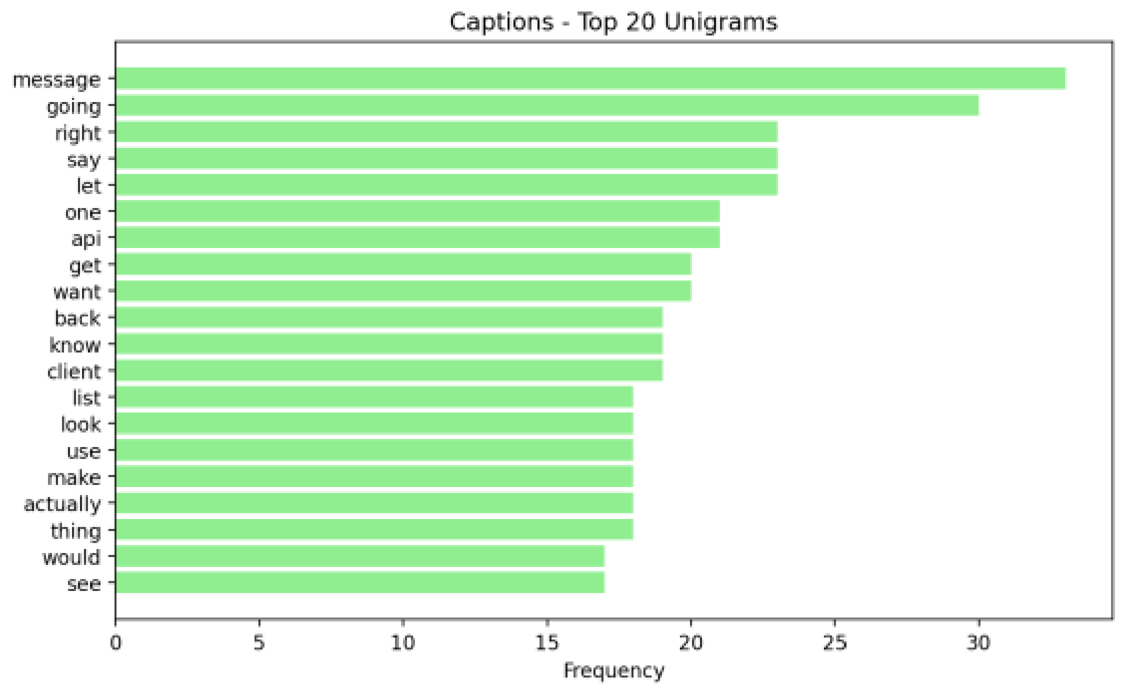

Figure 3: Top 20 Unigrams (captions)

A network graph was also created with NetworkX, where each node represented a word and edges indicated strong co-occurrence relationships. Finally, the co-occurrence results were exported to CSV for later reference.

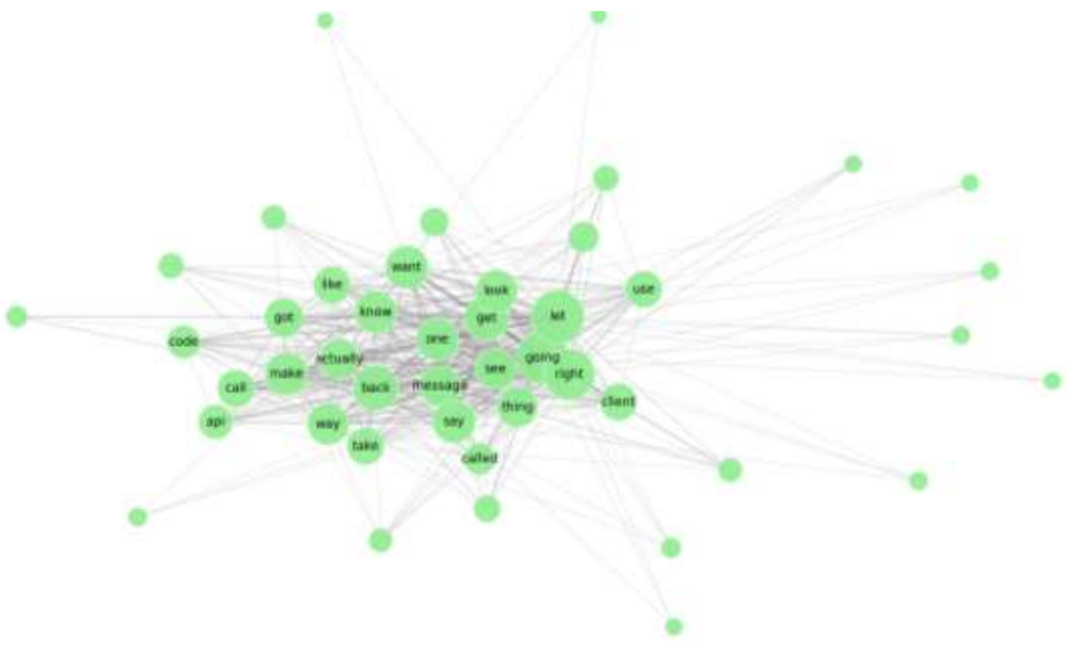

Figure 4: Network graph showing word co-occurrences (captions)

Algorithmic Pattern Discovery

The transactions were transformed into a one-hot encoded DataFrame suitable for Apriori analysis. Using mlxtend.frequent_patterns.apriori, frequent itemsets were generated at min_support = {0.30, 0.20, 0.10}. From these frequent itemsets, association rules were generated and filtered by confidence ≥ 0.6 and lift ≥ 1.2. The resulting rules were visualized using support vs. confidence scatter plots displayed the strength and reliability of discovered patterns. Separate runs were performed for both the comments and captions datasets.

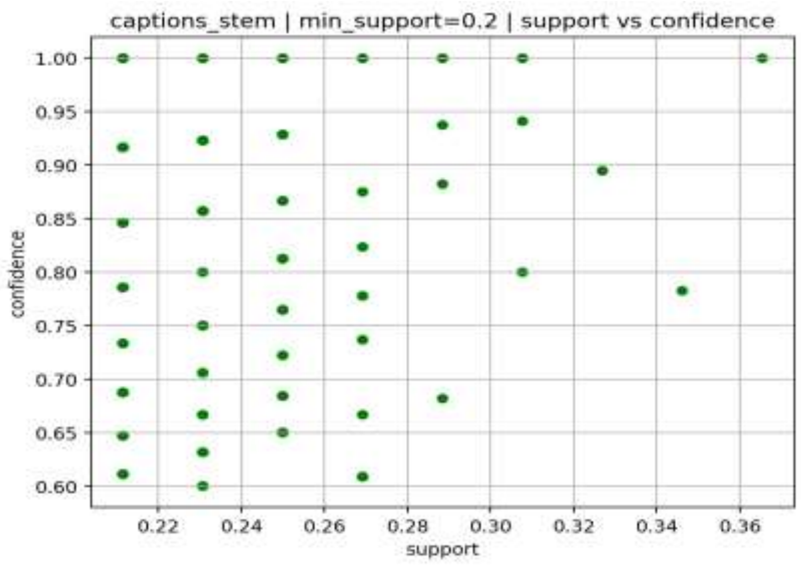

Figure 5 (a): Support vs. Confidence scatter plots

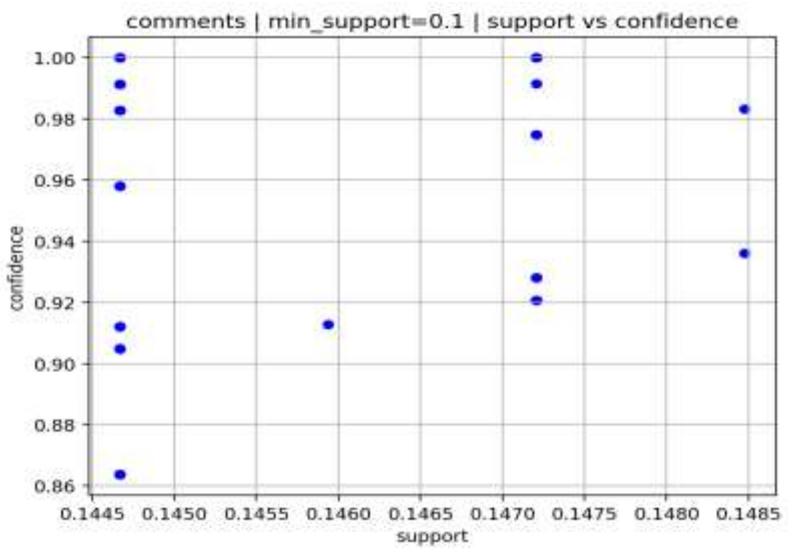

Figure 5 (b): Support vs. Confidence scatter plots

Exploration Using Variations

To examine how preprocessing and thresholds influence pattern discovery, several controlled experiments were conducted:

- **Varying Support**: Apriori was rerun with min_support = 0.15 and 0.05 to identify weaker but more inclusive associations.

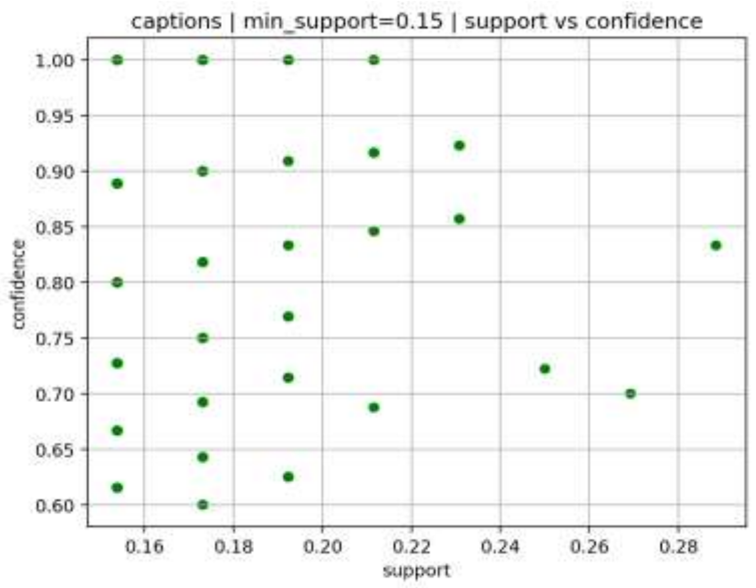

Figure 6: Support–Confidence scatter plot (captions)

- **Stemming:** The **PorterStemmer** replaced lemmatization to test how morphological simplification affects rule formation.

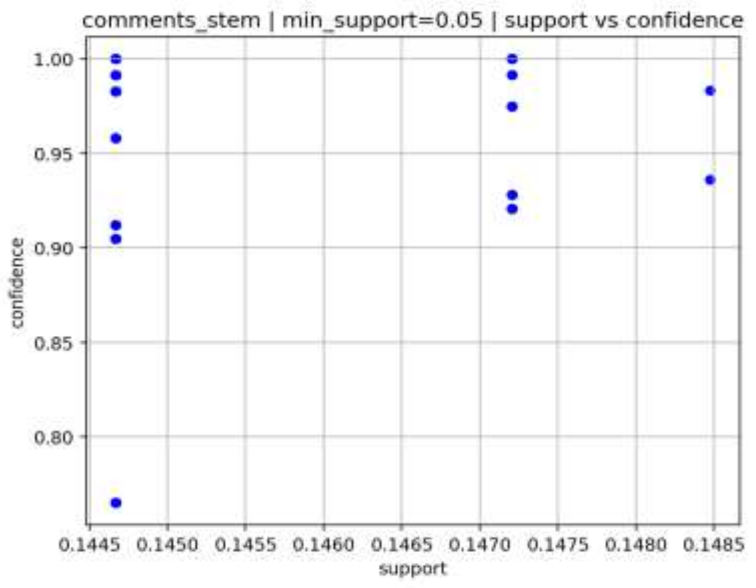

Figure 7: Support–Confidence scatter plot (comments)

- **Token Length Filtering:** Tokens shorter than four characters were excluded to reduce noise, and Apriori was rerun on the filtered transactions.

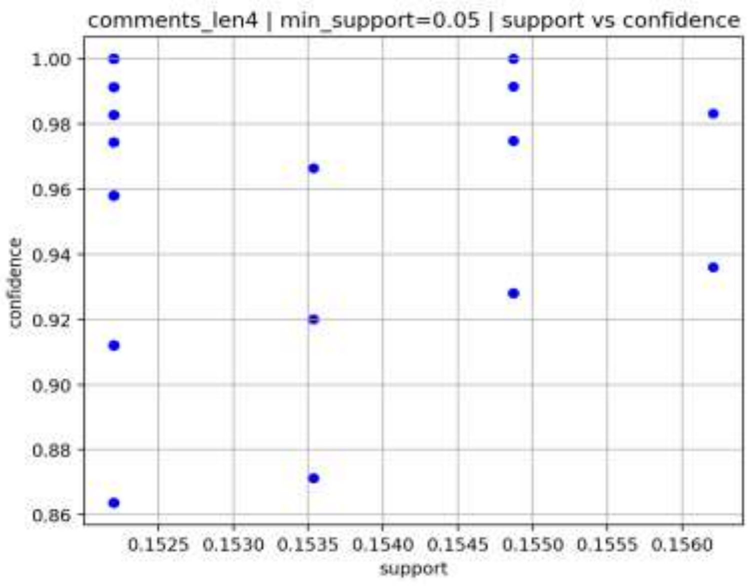

Figure 7: Support–Confidence scatter plot (comments)

- **Dataset Comparisons:** The entire pipeline was applied separately to comments, captions, and a merged dataset (combining both).

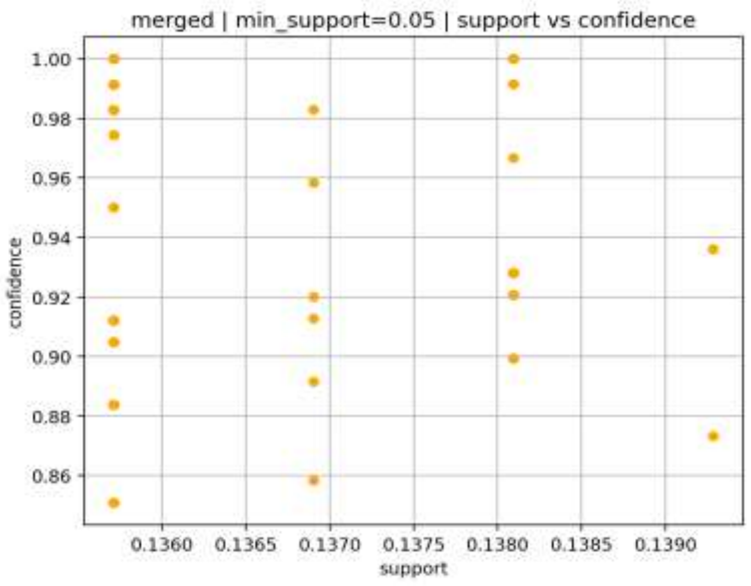

Figure 8: Merged Support–Confidence scatter plot

All variations of generated itemsets, rules and images were saved as CSV and PNGs files.

Pattern Interpretation and Visualization

Finally, patterns were interpreted using summary plots and visualizations. The top 10 two-itemsets (by support) and top 10 three-item rules (by confidence) were plotted for both datasets. A cluster graph was generated to highlight words that appeared together in high-lift rules.

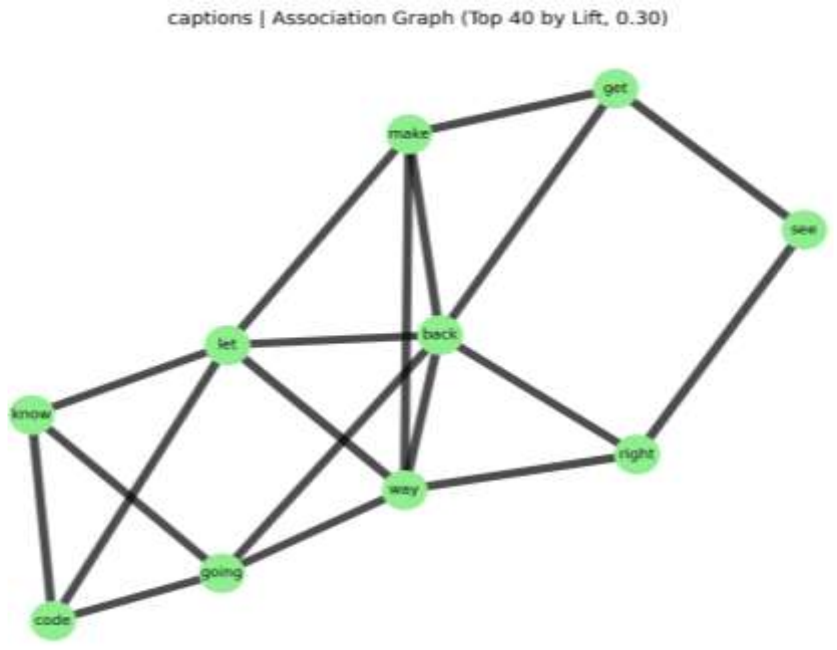

Figure 9 (b): Association Graph

A word cloud was created from the most frequent items.

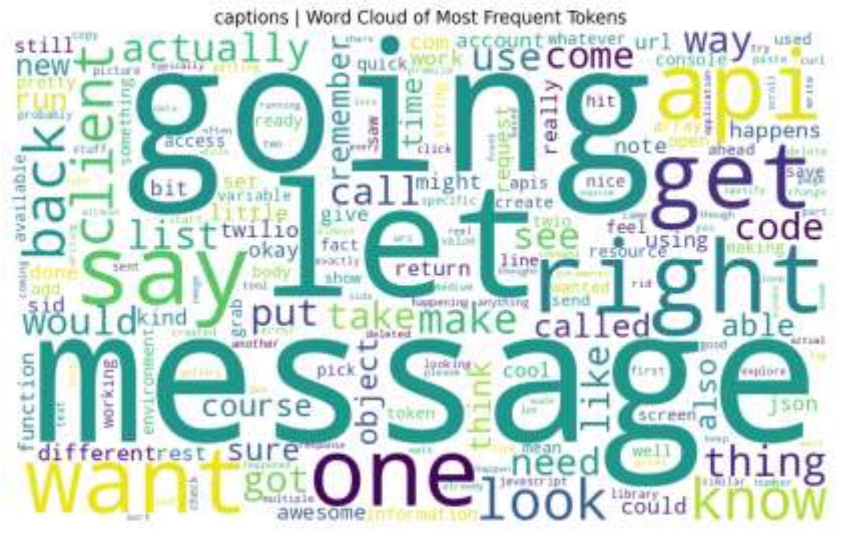

Figure 10: Word cloud of frequent items

Code Implementation

Loading & Cleaning Data

```
# Load cleaned data
comments = pd.read_csv("cleaned_comments.csv")
captions = pd.read_csv("cleaned_captions.csv")

# Verify cleaned column
assert 'cleaned_tokens' in comments.columns, "cleaned_tokens missing
in comments"
assert 'cleaned_tokens' in captions.columns, "cleaned_tokens missing
in captions"
print("Cleaned_tokens column found in both datasets.\n")

# Convert to list
def to_list(x):
    if pd.isna(x):
        return []
    if isinstance(x, list):
        return x
    try:
        return list(ast.literal_eval(x))
    except:
        return str(x).split()

comments['cleaned_tokens'] =
comments['cleaned_tokens'].apply(to_list)
captions['cleaned_tokens'] =
captions['cleaned_tokens'].apply(to_list)

# Drop empty tokens
before_comments = len(comments)
before_captions = len(captions)

comments = comments[comments['cleaned_tokens'].map(len) >
0].reset_index(drop=True)
captions = captions[captions['cleaned_tokens'].map(len) >
0].reset_index(drop=True)

print(f"Dropped {before_comments - len(comments)} empty rows from
comments.")
print(f"Dropped {before_captions - len(captions)} empty rows from
captions.\n")

# Sanity check
print("Sample tokens from comments:")
print(comments['cleaned_tokens'].head(3).to_list())
print("\nSample tokens from captions:")
print(captions['cleaned_tokens'].head(3).to_list())
```

Building Transactions

```
# Build transactions function
def make_transactions(df, min_len=3, dedup=False):
    transactions = []
    ids = []
    for i, tokens in enumerate(df['cleaned_tokens']):
        basket = tokens.copy()
        if dedup:
            basket = list(dict.fromkeys(basket))  # preserve order
but remove duplicates
        if len(basket) >= min_len:
            transactions.append(basket)
            ids.append(i)
    return transactions, ids

# Build transactions for comments
transactions_comments, tx_ids_comments = make_transactions(comments,
min_len=3, dedup=True)

# Build transactions for captions
transactions_captions, tx_ids_captions = make_transactions(captions,
min_len=3, dedup=True)

# Print
print("Transactions created successfully!")
print(f"Comments: {len(transactions_comments)} baskets")
print(f"Captions: {len(transactions_captions)} baskets")
```

Co-occurrence Counting

```
# --- Cell #7 ---
# Top 20 unigrams for comments
all_tokens_comments =
list(itertools.chain.from_iterable(comments['cleaned_tokens']))
unigram_counts_comments = Counter(all_tokens_comments)
top20_comments = unigram_counts_comments.most_common(20)
df_comments = pd.DataFrame(top20_comments,
columns=['token','count'])

# Top 20 unigrams for captions
all_tokens_captions =
list(itertools.chain.from_iterable(captions['cleaned_tokens']))
unigram_counts_captions = Counter(all_tokens_captions)
top20_captions = unigram_counts_captions.most_common(20)
df_captions = pd.DataFrame(top20_captions,
columns=['token','count'])

# Display
print("Top 20 Most Common Tokens")
display(pd.concat(
[df_comments.rename(columns={'token':'comment_token','count':'commen
t_count'}),

df_captions.rename(columns={'token':'caption_token','count':'caption
_count'})],
    axis=1
))

# --- Cell #9 ---
# Initialize English stop words list from NLTK
stop_words = set(stopwords.words('english'))

def count_pairs(baskets, min_count=3, stop_words=stop_words):
    """
    Count co-occurring word pairs across baskets (transactions).

    """
    c = Counter()
    for t in baskets:
        # Remove stopwords from each basket before counting
        t = [w for w in t if w not in stop_words]

        # Count each unique pair only once per basket
        c.update(itertools.combinations(sorted(set(t)), 2))

    # Filter pairs with count >= min_count and sort descending by
frequency
    pairs_sorted = sorted(
        [(a, b, n) for (a, b), n in c.items() if n >= min_count],
        key=lambda x: x[2],
        reverse=True
    )
    return pairs_sorted, c


# Compute frequent word pairs for comments and captions
pairs_comments, counter_comments =
count_pairs(transactions_comments, min_count=3)
pairs_captions, counter_captions =
count_pairs(transactions_captions, min_count=3)

# Display top 20 co-occurring pairs
df_comments = pd.DataFrame(pairs_comments[:20], columns=['word1',
'word2', 'count'])
df_captions = pd.DataFrame(pairs_captions[:20], columns=['word1',
'word2', 'count'])

display(
    pd.concat(
        [df_comments.add_prefix('comment_'),
         df_captions.add_prefix('caption_')],
        axis=1
    )
)
```

```
# --- Cell #10 ---
# Tiny helper
def _flatten(list_of_lists):
    return list(itertools.chain.from_iterable(list_of_lists))

def plot_top_pairs_and_unigrams(pairs, tokens_list, title, prefix,
color, out_dir="."):
    # Ensure output folder exists
    os.makedirs(out_dir, exist_ok=True)

    # Align token filtering with upstream pair counting (stopwords
removed)
    tokens = _flatten(tokens_list)
    tokens = [t for t in tokens if t not in stop_words and
isinstance(t, str) and t]

    # Filter stopwords for visualization
    filtered_pairs = [(a, b, n) for (a, b, n) in pairs if a not in
stop_words and b not in stop_words]

    # === Top 20 Co-occurring Pairs ===
    top_pairs_for_plot = filtered_pairs[:20]
    pair_labels = [f"{a} / {b}" for a, b, _ in top_pairs_for_plot]
    pair_counts = [c for _, _, c in top_pairs_for_plot]

    plt.figure(figsize=(8, 5))
    plt.barh(pair_labels[::-1], pair_counts[::-1], color=color)
    plt.title(f"{title} - Top 20 Co-occurring Word Pairs")
    plt.xlabel("Co-occurrence Count")
    plt.tight_layout()

    # Save
    out1 = os.path.join(out_dir, f"{prefix}_top20_pairs.png")
    plt.savefig(out1, dpi=200, bbox_inches="tight")
    plt.show()
    plt.close()
    print(f"Saved: {out1}")

    # === Top 20 Unigrams ===
    uni = Counter(tokens).most_common(20)
    if uni:
        uni_labels, uni_counts = zip(*uni)
        plt.figure(figsize=(8, 5))
        plt.barh(list(uni_labels)[::-1], list(uni_counts)[::-1],
color=color)
        plt.title(f"{title} - Top 20 Unigrams")
        plt.xlabel("Frequency")
        plt.tight_layout()

        # Save
        out2 = os.path.join(out_dir, f"{prefix}_top20_unigrams.png")
        plt.savefig(out2, dpi=200, bbox_inches="tight")
        plt.show()
        plt.close()
        print(f"Saved: {out2}")


# Save plots to OUT_DIR
df_pairs_comments = plot_top_pairs_and_unigrams(
    pairs_comments, transactions_comments,
    title="Comments",
    prefix="comments",
    color="skyblue",
    out_dir=OUT_DIR
)

df_pairs_captions = plot_top_pairs_and_unigrams(
    pairs_captions, transactions_captions,
    title="Captions",
    prefix="captions",
    color="lightgreen",
    out_dir=OUT_DIR
)

# --- Cell #11 ---
# Export to CSV
df_pairs_comments = pd.DataFrame(pairs_comments, columns=["word1",
"word2", "count"])
df_pairs_captions = pd.DataFrame(pairs_captions, columns=["word1",
"word2", "count"])

df_pairs_comments.to_csv(os.path.join(OUT_DIR,
"comments_cooccurrence_pairs.csv"), index=False)
df_pairs_captions.to_csv(os.path.join(OUT_DIR,
"captions_cooccurrence_pairs.csv"), index=False)

print("Saved:")
print("-", os.path.join(OUT_DIR, "comments_cooccurrence_pairs.csv"))
print("-", os.path.join(OUT_DIR, "captions_cooccurrence_pairs.csv"))

# --- Cell #12 ---
def plot_cooccurrence_network(
    pairs_sorted,
    title,
    color,                        # "skyblue" or "lightgreen"
    min_count=3,                  # keep edges with weight >= min_count
    max_edges=400,                # cap strongest edges for clarity
    min_degree=2,                 # drop low-degree leaves to reduce
clutter (set 1 to keep all)
    label_mode="topk",            # "topk" or "all"
    label_top_k=20,               # used if label_mode="topk"
    ban_tokens=None,              # iterable of tokens to drop entirely
    seed=42
):
    os.makedirs(OUT_DIR, exist_ok=True)  # ensure directory exists

    # Banlist cleanup
    ban = set(ban_tokens or [])
    url_like = lambda w: ("http" in w) or ("www" in w) or
(w.endswith((".com", ".net", ".org")))

    # Filter + cap edges
    edges = [(a, b, n) for (a, b, n) in pairs_sorted
             if n >= min_count and a not in ban and b not in ban and
not url_like(a) and not url_like(b)]
    if not edges:
        print(f"[{title}] No edges at threshold
min_count={min_count}.")
        return
    edges = sorted(edges, key=lambda x: x[2],
reverse=True)[:max_edges]

    # Build graph
    G = nx.Graph()
    G.add_weighted_edges_from(edges)

    # Optional prune low-degree nodes
    if min_degree > 1:
        keep = [n for n, d in G.degree() if d >= min_degree]
        G = G.subgraph(keep).copy()
        if G.number_of_edges() == 0:
            print(f"[{title}] All edges removed after
min_degree={min_degree}. Try min_degree=1.")
            return

    # Stats
    n_nodes, n_edges = G.number_of_nodes(), G.number_of_edges()
    stats_text = f"Words: {n_nodes} | Edges: {n_edges}"

    # Strength & labels
    strength = dict(G.degree(weight="weight"))
    if label_mode == "all":
        label_nodes = set(G.nodes())
    else:
        label_nodes = set(sorted(strength, key=strength.get,
reverse=True)[:min(label_top_k, len(G))])

    # Layout
    pos = nx.spring_layout(G, seed=seed, k=0.75 / (len(G) ** 0.35 +
1e-6), iterations=100)

    # Scaling
    s_vals = list(strength.values())
    s_min, s_max = min(s_vals), max(s_vals)
    node_sizes = [
        150 + (1500 - 150) * ((strength[n] - s_min) / (s_max - s_min
+ 1e-9))
        for n in G.nodes()
    ]
    w_vals = [d["weight"] for _, _, d in G.edges(data=True)]
    w_min, w_max = min(w_vals), max(w_vals)
    edge_widths = [0.6 + (4.0 - 0.6) * ((w - w_min) / (w_max - w_min
+ 1e-9)) for w in w_vals]
```

```
    # Draw
    plt.figure(figsize=(13, 8))
    nx.draw_networkx_edges(G, pos, width=edge_widths, alpha=0.25,
edge_color="gray")
    nx.draw_networkx_nodes(G, pos,
                           node_size=node_sizes, alpha=0.9,
                           node_color=color, linewidths=0.5,
edgecolors="white")
    nx.draw_networkx_labels(G, pos, labels={n: n for n in
label_nodes}, font_size=9)
    plt.title(title)
    plt.text(0.01, 0.98, stats_text, transform=plt.gca().transAxes,
             va="top", ha="left", fontsize=10)
    plt.axis("off")
    plt.tight_layout()

    # Save
    safe_title = title.lower().replace(" ", "_").replace("–", "-")
    out_path = os.path.join(OUT_DIR,
f"{safe_title}_cooccurrence_network.png")
    plt.savefig(out_path, dpi=220, bbox_inches="tight")
    print("Saved:", out_path)

    # Show saved image
    plt.show()
    plt.close()


# Recommended banlist for noise tokens
ban_tokens = {
    "http", "https", "www", "com", "amp", "quot", "href", "watch",
"v", "zxjw", "utm",
    "youtu", "youtube", "link", "click", "token", "id"
}

# Plot Network Graphs
plot_cooccurrence_network(
    pairs_comments,
    title="Comments – Co-occurrence Network",
    color="skyblue",
    min_count=3,
    max_edges=400,
    min_degree=2,
    label_mode="topk",
    label_top_k=25,
    ban_tokens=ban_tokens
)

plot_cooccurrence_network(
    pairs_captions,
    title="Captions – Co-occurrence Network",
    color="lightgreen",
    min_count=3,
    max_edges=400,
    min_degree=2,
    label_mode="topk",
    label_top_k=25,
    ban_tokens=ban_tokens
)
```

Apriori Analysis

```
# --- Cell #13 ---
# One-hot encode transactions
def to_onehot_df(transactions):
    te = TransactionEncoder()
    arr = te.fit(transactions).transform(transactions)
    return pd.DataFrame(arr, columns=te.columns_).astype(bool)
onehot_comments = to_onehot_df(transactions_comments)
onehot_captions = to_onehot_df(transactions_captions)

print("One-hot shapes:", onehot_comments.shape,
onehot_captions.shape)

# --- Cell #14 ---
# Apriori runner
def _supp_suffix(s: float) -> str:
    # 0.3 -> 'supp03', 0.15 -> 'supp015', 0.05 -> 'supp005'
    return "supp" + f"{s:.2f}".split(".")[1].ljust(2, "0")

def apriori_pipeline(onehot: pd.DataFrame, label: str,
                     support: float,
                     min_conf: float = 0.6,
                     min_lift: float = 1.2,
                     per_support_top_k: int = 200,
                     max_plot_points: int = 250,
                     do_plot: bool = True):
    os.makedirs(OUT_DIR, exist_ok=True)
    assert 0 < support <= 1.0, "support must be in (0,1]"
    onehot = onehot.astype(bool)
    suff = _supp_suffix(support)

    # Pre-filter single items by support
    item_support = onehot.mean(axis=0).sort_values(ascending=False)
    keep_cols = item_support[item_support >= support].index.tolist()

    fis_path   = os.path.join(OUT_DIR,
f"{label}_frequent_itemsets_{suff}.csv")
    rules_path = os.path.join(OUT_DIR,
f"{label}_association_rules_{suff}.csv")

    if not keep_cols:
        print(f"[{label}] s={support}: no 1-items meet support.
Skipping CSV save.")
        return # Exit function if no 1-items meet support

    # Restrict to top-K items
    if per_support_top_k is not None and len(keep_cols) >
per_support_top_k:
        keep_cols =
item_support.loc[keep_cols].iloc[:per_support_top_k].index.tolist()
    onehot_s = onehot[keep_cols]

    # Apriori up to length 3
    fis = apriori(onehot_s, min_support=support, use_colnames=True,
low_memory=True, max_len=3)

    if fis.empty:
        print(f"[{label}] s={support}: Apriori found no itemsets.
Skipping CSV save.")
        return # Exit function if no itemsets found

    fis["length"] = fis["itemsets"].apply(len)
    # Save all frequent itemsets, including length 1, for potential
later use
    if not fis.empty and len(fis) > 0: # Explicitly check for non-
empty DataFrame with rows
        fis.to_csv(fis_path, index=False)
        print(f"Saved frequent itemsets → {fis_path}
(n={len(fis)})")
    else:
        print(f"[{label}] s={support}: No frequent itemsets to
save.")


    if fis[fis["length"] >= 2].empty:
        print(f"[{label}] s={support}: only 1-itemsets found
({len(fis)} total). No 2-3 itemsets → no rules. Skipping rules CSV
save.")
        return # Exit function if only 1-itemsets found


    # Generate rules from ALL frequent itemsets
    rules = association_rules(fis, metric="confidence",
min_threshold=1e-9)


    if rules.empty:
        print(f"[{label}] s={support}: association_rules produced 0
rules. Skipping rules CSV save.")
        return # Exit function if no rules produced

    rules_f = rules[(rules["confidence"] >= min_conf) &
(rules["lift"] >= min_lift)].copy()
    if not rules_f.empty and len(rules_f) > 0: # Explicitly check
for non-empty DataFrame with rows
        rules_f.to_csv(rules_path, index=False)
        print(f"Saved rules → {rules_path} (n={len(rules_f)})")
    else:
         print(f"[{label}] s={support}: No rules meet
min_conf/min_lift. Skipping rules CSV save.")
```

```
    # Scatter (support vs confidence)
    if do_plot and not rules_f.empty and len(rules_f) > 0: # Only
plot if there are rules to plot
        plot_df = rules_f if len(rules_f) <= max_plot_points else
rules_f.sample(n=max_plot_points, random_state=42)
        color = "blue" if label.lower().startswith("comment") else
("green" if label.lower().startswith("caption") else "orange")

        plt.figure(figsize=(6,5))
        plt.scatter(plot_df["support"], plot_df["confidence"],
c=color, edgecolors="none")
        plt.title(f"{label} | min_support={support} | support vs
confidence")
        plt.xlabel("support")
        plt.ylabel("confidence")
        plt.grid(True)
        plt.tight_layout()
        plot_path = os.path.join(OUT_DIR,
f"{label}_support_confidence_{suff}.png")
        plt.savefig(plot_path, dpi=200, bbox_inches="tight")
        plt.show()
        print(f"Saved plot → {plot_path}")

# --- Cell #15 ---
# Run Apriori pipeline for comments (supports 0.3, 0.2, 0.1, 0.15,
0.05)
apriori_pipeline(onehot_comments, "comments", support=0.30,
per_support_top_k=200)
apriori_pipeline(onehot_comments, "comments", support=0.20,
per_support_top_k=180)
apriori_pipeline(onehot_comments, "comments", support=0.15,
per_support_top_k=200)
apriori_pipeline(onehot_comments, "comments", support=0.10,
per_support_top_k=160)
apriori_pipeline(onehot_comments, "comments", support=0.05,
per_support_top_k=150)

# --- Cell #16 ---
# Run Apriori pipeline for captions (supports 0.3, 0.2, 0.1, 0.15,
0.05)
apriori_pipeline(onehot_captions, "captions", support=0.30,
per_support_top_k=200)
apriori_pipeline(onehot_captions, "captions", support=0.20,
per_support_top_k=180)
apriori_pipeline(onehot_captions, "captions", support=0.15,
per_support_top_k=200)
apriori_pipeline(onehot_captions, "captions", support=0.10,
per_support_top_k=160)
apriori_pipeline(onehot_captions, "captions", support=0.05,
per_support_top_k=150)

# --- Cell #17 ---
# Stemming
ps = PorterStemmer()
```

```
def stem_tokens_list(tokens_list):
    return [ps.stem(t) for t in tokens_list if isinstance(t, str)]

comments_stem = comments.copy()
captions_stem = captions.copy()
comments_stem['cleaned_tokens'] =
comments_stem['cleaned_tokens'].apply(stem_tokens_list)
captions_stem['cleaned_tokens'] =
captions_stem['cleaned_tokens'].apply(stem_tokens_list)

transactions_comments_stem, _ = make_transactions(comments_stem,
min_len=3, dedup=True)
transactions_captions_stem, _ = make_transactions(captions_stem,
min_len=3, dedup=True)

onehot_comments_stem = to_onehot_df(transactions_comments_stem)
onehot_captions_stem = to_onehot_df(transactions_captions_stem)

# Run Apriori for all supports
captions_supports = [0.30, 0.20, 0.15, 0.10, 0.05]
comments_supports = [0.10, 0.15, 0.05]

for s in captions_supports:
    print(f"[captions_stem] running Apriori at support={s}")
    apriori_pipeline(onehot_captions_stem, "captions_stem",
support=s, per_support_top_k=200)

for s in comments_supports:
    print(f"[comments_stem] running Apriori at support={s}")
    apriori_pipeline(onehot_comments_stem, "comments_stem",
support=s, per_support_top_k=200)

# --- Cell #18 ---
# Remove tokens under 4 characters, rebuild transactions/one-hot,
rerun Apriori (0.3)
def filter_min_chars(df, min_chars=4):
    df2 = df.copy()
    df2['cleaned_tokens'] = df2['cleaned_tokens'].apply(
        lambda toks: [t for t in toks if isinstance(t, str) and
len(t) >= min_chars]
    )
    return df2

comments_len4 = filter_min_chars(comments, min_chars=4)
captions_len4 = filter_min_chars(captions, min_chars=4)

# Transactions (len>=3)
transactions_comments_len4, _ = make_transactions(comments_len4,
min_len=3, dedup=True)
transactions_captions_len4, _ = make_transactions(captions_len4,
min_len=3, dedup=True)

onehot_comments_len4 = to_onehot_df(transactions_comments_len4)
onehot_captions_len4 = to_onehot_df(transactions_captions_len4)
```

```
# Run Apriori for all supports
captions_supports = [0.30, 0.20, 0.15, 0.10, 0.05]
comments_supports = [0.10, 0.15, 0.05]

for s in captions_supports:
    print(f"[captions_len4] Apriori at support={s}")
    apriori_pipeline(onehot_captions_len4, "captions_len4",
support=s, per_support_top_k=200)

for s in comments_supports:
    print(f"[comments_len4] Apriori at support={s}")
    apriori_pipeline(onehot_comments_len4, "comments_len4",
support=s, per_support_top_k=200)

# --- Cell #19 ---
# len>=4

# captions_len4
apriori_pipeline(onehot_captions_len4, "captions_len4",
support=0.20, per_support_top_k=180)
apriori_pipeline(onehot_captions_len4, "captions_len4",
support=0.10, per_support_top_k=160)
apriori_pipeline(onehot_captions_len4, "captions_len4",
support=0.15, per_support_top_k=180)
apriori_pipeline(onehot_captions_len4, "captions_len4",
support=0.05, per_support_top_k=140)

# --- Cell #20 ---
# len>=4

# comments_len4
apriori_pipeline(onehot_comments_len4, "comments_len4",
support=0.20, per_support_top_k=180)
apriori_pipeline(onehot_comments_len4, "comments_len4",
support=0.10, per_support_top_k=160)
apriori_pipeline(onehot_comments_len4, "comments_len4",
support=0.15, per_support_top_k=180)
apriori_pipeline(onehot_comments_len4, "comments_len4",
support=0.05, per_support_top_k=140)

# --- Cell #21 ---
# Merge comments + captions, create one-hot, rerun Apriori (0.3,
0.2, 0.1, 0,15, 0.05)
transactions_merged = transactions_comments + transactions_captions
onehot_merged = to_onehot_df(transactions_merged)

apriori_pipeline(onehot_merged, "merged", support=0.30,
per_support_top_k=220)
apriori_pipeline(onehot_merged, "merged", support=0.20,
per_support_top_k=200)
apriori_pipeline(onehot_merged, "merged", support=0.10,
per_support_top_k=180)
apriori_pipeline(onehot_merged, "merged", support=0.15,
per_support_top_k=220)
```

```
apriori_pipeline(onehot_merged, "merged", support=0.05,
per_support_top_k=180)
```

Visualization & Exporting Results

```
# Apriori runner
def _supp_suffix(s: float) -> str:
    # 0.3 -> 'supp03', 0.15 -> 'supp015', 0.05 -> 'supp005'
    return "supp" + f"{s:.2f}".split(".")[1].ljust(2, "0")

def apriori_pipeline(onehot: pd.DataFrame, label: str,
                     support: float,
                     min_conf: float = 0.6,
                     min_lift: float = 1.2,
                     per_support_top_k: int = 200,
                     max_plot_points: int = 250,
                     do_plot: bool = True):
    os.makedirs(OUT_DIR, exist_ok=True)
    assert 0 < support <= 1.0, "support must be in (0,1]"
    onehot = onehot.astype(bool)
    suff = _supp_suffix(support)

    # Pre-filter single items by support
    item_support = onehot.mean(axis=0).sort_values(ascending=False)
    keep_cols = item_support[item_support >= support].index.tolist()

    fis_path   = os.path.join(OUT_DIR,
f"{label}_frequent_itemsets_{suff}.csv")
    rules_path = os.path.join(OUT_DIR,
f"{label}_association_rules_{suff}.csv")

    if not keep_cols:
        print(f"[{label}] s={support}: no 1-items meet support.
Skipping CSV save.")
        return # Exit function if no 1-items meet support

    # Restrict to top-K items
    if per_support_top_k is not None and len(keep_cols) >
per_support_top_k:
        keep_cols =
item_support.loc[keep_cols].iloc[:per_support_top_k].index.tolist()
    onehot_s = onehot[keep_cols]

    # Apriori up to length 3
    fis = apriori(onehot_s, min_support=support, use_colnames=True,
low_memory=True, max_len=3)

    if fis.empty:
        print(f"[{label}] s={support}: Apriori found no itemsets.
Skipping CSV save.")
        return # Exit function if no itemsets found
    fis["length"] = fis["itemsets"].apply(len)
    # Save all frequent itemsets, including length 1, for potential
later use
    if not fis.empty and len(fis) > 0: # Explicitly check for non-
empty DataFrame with rows
        fis.to_csv(fis_path, index=False)
        print(f"Saved frequent itemsets → {fis_path}
(n={len(fis)})")
    else:
        print(f"[{label}] s={support}: No frequent itemsets to
save.")


    if fis[fis["length"] >= 2].empty:
        print(f"[{label}] s={support}: only 1-itemsets found
({len(fis)} total). No 2-3 itemsets → no rules. Skipping rules CSV
save.")
        return # Exit function if only 1-itemsets found


    # Generate rules from ALL frequent itemsets
    rules = association_rules(fis, metric="confidence",
min_threshold=1e-9)


    if rules.empty:
        print(f"[{label}] s={support}: association_rules produced 0
rules. Skipping rules CSV save.")
        return # Exit function if no rules produced

    rules_f = rules[(rules["confidence"] >= min_conf) &
(rules["lift"] >= min_lift)].copy()
    if not rules_f.empty and len(rules_f) > 0: # Explicitly check
for non-empty DataFrame with rows
        rules_f.to_csv(rules_path, index=False)
        print(f"Saved rules → {rules_path} (n={len(rules_f)})")
    else:
         print(f"[{label}] s={support}: No rules meet
min_conf/min_lift. Skipping rules CSV save.")


    # Scatter (support vs confidence)
    if do_plot and not rules_f.empty and len(rules_f) > 0: # Only
plot if there are rules to plot
        plot_df = rules_f if len(rules_f) <= max_plot_points else
rules_f.sample(n=max_plot_points, random_state=42)
        color = "blue" if label.lower().startswith("comment") else
("green" if label.lower().startswith("caption") else "orange")

        plt.figure(figsize=(6,5))
        plt.scatter(plot_df["support"], plot_df["confidence"],
c=color, edgecolors="none")
        plt.title(f"{label} | min_support={support} | support vs
confidence")
        plt.xlabel("support")
        plt.ylabel("confidence")
        plt.grid(True)
        plt.tight_layout()
        plot_path = os.path.join(OUT_DIR,
f"{label}_support_confidence_{suff}.png")
        plt.savefig(plot_path, dpi=200, bbox_inches="tight")
        plt.show()
        print(f"Saved plot → {plot_path}")

# --- Cell #23 ---
# Top 10 2-itemsets by support
def _color_for(label: str):
    """Helper to return color based on label."""
    if label.lower().startswith("comment"):
        return "skyblue"
    elif label.lower().startswith("caption"):
        return "lightgreen"
    elif label.lower().startswith("merged"):
        return "orange"
    else:
        return "gray"

def plot_topN_itemsets(label: str, supp_key="0.30", length=2,
metric="support", topN=10):
    supp = SUPP[supp_key]
    fis = load_itemsets(label, supp_key)
    if fis.empty:
        print(f"[{label}] No itemsets at {supp_key}")
        return
    top = fis[fis["length"]==length].sort_values(metric,
ascending=False).head(topN).copy()
    if top.empty:
        print(f"[{label}] No {length}-itemsets at {supp_key}")
        return
    top["label"] = top["itemsets_parsed"].apply(lambda t: " &
".join(t))
    plt.figure(figsize=(8,5))
    plt.barh(top["label"][::-1], top[metric][::-1],
color=_color_for(label))
    plt.xlabel(metric.capitalize())
    plt.title(f"{label} | Top {topN} {length}-itemsets by {metric}
({supp_key})")
    plt.tight_layout()
    out = os.path.join(OUT_DIR,
f"{label}_top{length}itemsets_{metric}_{supp}.png")
    plt.savefig(out, dpi=200, bbox_inches="tight"); plt.show()
    print("Saved:", out)

# Top 10 3-itemsets by confidence
def plot_top3_confidence(label: str, supp_key="0.30", topN=10):
    supp = SUPP[supp_key]
    rules = load_rules(label, supp) # Pass the support suffix here
    if rules.empty:
        print(f"[{label}] No rules at {supp_key}")
        return
    rules["rule_label"] = rules.apply(
        lambda r: " & ".join(r["ante_parsed"]) + " → " + " &
".join(r["cons_parsed"]), axis=1)
    top = rules.sort_values("confidence",
ascending=False).head(topN)
    plt.figure(figsize=(8,5))
    plt.barh(top["rule_label"][::-1], top["confidence"][::-1],
color=_color_for(label))
    plt.xlabel("Confidence")
    plt.title(f"{label} | Top {topN} Rules by Confidence
({supp_key})")
    plt.tight_layout()
    out = os.path.join(OUT_DIR,
f"{label}_top3itemsets_confidence_{supp}.png")
    plt.savefig(out, dpi=200, bbox_inches="tight"); plt.show()
    print("Saved:", out)

# Word Cloud from Most Frequent Items
def wordcloud_from_onehot(onehot_df, label: str, top_n=300):
    freqs =
onehot_df.astype(bool).mean(axis=0).sort_values(ascending=False).hea
d(top_n)
    wc = WordCloud(width=1000, height=600, background_color="white")
    img = wc.generate_from_frequencies(freqs.to_dict())
    plt.figure(figsize=(10,6))
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"{label} | Word Cloud of Most Frequent Tokens")
    out = os.path.join(OUT_DIR, f"{label}_wordcloud.png")
    plt.savefig(out, dpi=220, bbox_inches="tight"); plt.show()
    print("Saved:", out)

# Cluster Graph of Word Associations
def graph_from_rules(label: str, supp_key="0.30", top_edges=40):
    supp = SUPP[supp_key]
    rules = load_rules(label, supp) # Pass the support suffix here
    if rules.empty:
        print(f"[{label}] No rules at {supp_key}")
        return
    rr = rules.sort_values("lift",
ascending=False).head(top_edges).copy()
    G = nx.Graph()
    for _, r in rr.iterrows():
        for a in r["ante_parsed"]:
            for c in r["cons_parsed"]:
                w = float(r["lift"])
                G.add_edge(a, c, weight=w)

    pos = nx.spring_layout(G, seed=42, k=0.7)
    plt.figure(figsize=(10,8))
    widths = [2 * G[u][v]['weight'] for u, v in G.edges()]
    node_color = "lightgreen" if
label.lower().startswith("captions") else "#89CFF0"
    nx.draw_networkx_nodes(G, pos, node_color=node_color,
node_size=700)
```

```
    nx.draw_networkx_edges(G, pos, width=widths, alpha=0.7)
    nx.draw_networkx_labels(G, pos, font_size=9)
    plt.axis("off")
    plt.title(f"{label} | Association Graph (Top {top_edges} by
Lift, {supp_key})")
    out = os.path.join(OUT_DIR, f"{label}_assoc_graph_{supp}.png")
    plt.savefig(out, dpi=220, bbox_inches="tight"); plt.show()
    print("Saved:", out)

# Helper to load rules with parsed itemsets
def load_rules(label: str, supp_suffix: str) -> pd.DataFrame:
    """Load association rules CSV written by the pipeline, add
parsed tuples."""
    pattern = os.path.join(OUT_DIR,
f"{label}_association_rules_{supp_suffix}.csv")
    paths = glob.glob(pattern)
    if not paths or not os.path.exists(paths[0]) or
os.path.getsize(paths[0]) == 0:
        return pd.DataFrame() # Return empty DataFrame if file is
missing or empty

    try:
        df = pd.read_csv(paths[0])
        if df.empty:
            return pd.DataFrame() # Return empty DataFrame if file
is empty after reading
    except pd.errors.EmptyDataError:
        return pd.DataFrame() # Catch EmptyDataError and return
empty DataFrame
    except Exception as e:
        print(f"Error reading rules file {paths[0]}: {e}")
        return pd.DataFrame() # Catch any other exceptions and
return empty DataFrame

    if {"antecedents","consequents"}.issubset(df.columns):
        df["ante_parsed"] = df["antecedents"].apply(_parse_fs)
        df["cons_parsed"] = df["consequents"].apply(_parse_fs)
    elif {"antecedents_str","consequents_str"}.issubset(df.columns):
         df["ante_parsed"] = df["antecedents_str"].apply(_parse_fs)
         df["cons_parsed"] = df["consequents_str"].apply(_parse_fs)
    else:
        print(f"Warning: Unexpected columns in rules file
{paths[0]}")
        return pd.DataFrame() # Return empty DataFrame if columns
are unexpected

    return df

# --- Cell #24 ---
# Define which supports exist for each dataset
available_supports = {
    "captions": ["0.30", "0.20", "0.10", "0.15", "0.05"],
    "comments": ["0.30", "0.20", "0.10", "0.15", "0.05"]
}
```

```
# Loop through datasets and their valid support levels
for label, supports in available_supports.items():
    print(f"\nProcessing {label.upper()}")
    for supp_key in supports:
        # Top 10 2-itemsets by support
        plot_topN_itemsets(label, supp_key, length=2,
metric="support")

        # Top 10 3-itemsets by confidence and Association Graph
(only if rules exist)
        rules_path = os.path.join(OUT_DIR,
f"{label}_association_rules_{SUPP[supp_key]}.csv")
        if os.path.exists(rules_path) and
os.path.getsize(rules_path) > 0:
             # Pass the correct support suffix to load_rules
            rules_df = load_rules(label, SUPP[supp_key])
            if not rules_df.empty:
                plot_top3_confidence(label, supp_key)
                graph_from_rules(label, supp_key)
            else:
                 print(f"[{label}] Skipping rule-based plots at
{supp_key} (no rules found after loading).")
        else:
            print(f"[{label}] Skipping rule-based plots at
{supp_key} (no rules file found or is empty).")

# --- Cell #25 ---
# WORD CLOUDS (from one-hot data)
wordcloud_from_onehot(onehot_captions, "captions")
wordcloud_from_onehot(onehot_comments, "comments")

# --- Cell #26 ---
# Compare patterns – captions vs comments
def compare_patterns(onehot_a, onehot_b, label_a="captions",
label_b="comments", top_n=15):
    # Compute average item presence (support) for each dataset
    sa =
onehot_a.astype(bool).mean(axis=0).sort_values(ascending=False).head
(top_n)
    sb =
onehot_b.astype(bool).mean(axis=0).sort_values(ascending=False).head
(top_n)

    # Combine for quick tabular comparison
    cmp = pd.DataFrame({label_a: sa, label_b: sb}).fillna(0)
    display(cmp)

    # Plot support comparison lines
    plt.figure(figsize=(10,5))
    plt.plot(range(len(sa)), sa.values, marker="o", label=label_a,
color="green")
    plt.plot(range(len(sb)), sb.values, marker="o", label=label_b,
color="blue")
    plt.xticks(range(top_n), range(1, top_n+1))
    plt.ylabel("Support")
    plt.title("Top-item supports: captions vs comments")
    plt.legend()
    plt.tight_layout()
    out = os.path.join(OUT_DIR, "compare_top_items.png")
    plt.savefig(out, dpi=200, bbox_inches="tight")
    plt.show()
    print("Saved:", out)

# Run comparison
compare_patterns(onehot_captions, onehot_comments)

# --- Cell #27 ---
# Save final cleaned dataset with itemsets and rules
# Combines all available itemsets/rules files for each label into
two master CSVs.
def consolidate_outputs(label: str):
    fis_files =
sorted(glob.glob(f"{label}_frequent_itemsets_*.csv"))
    rules_files =
sorted(glob.glob(f"{label}_association_rules_*.csv"))

    all_fis = []
    for p in fis_files:
        try:
            df = pd.read_csv(p)
            df["src_file"] = os.path.basename(p)
            all_fis.append(df)
        except Exception:
            pass
    all_rules = []
    for p in rules_files:
        try:
            df = pd.read_csv(p)
            df["src_file"] = os.path.basename(p)
            all_rules.append(df)
        except Exception:
            pass

    fis_out = pd.concat(all_fis, ignore_index=True) if all_fis else
pd.DataFrame()
    rules_out = pd.concat(all_rules, ignore_index=True) if all_rules
else pd.DataFrame()

    fis_path = os.path.join(OUT_DIR,
f"{label}_FINAL_frequent_itemsets.csv")
    rules_path = os.path.join(OUT_DIR,
f"{label}_FINAL_association_rules.csv")
    fis_out.to_csv(fis_path, index=False)
    rules_out.to_csv(rules_path, index=False)
    print("Saved:", fis_path, "|", rules_path)

consolidate_outputs("captions")
consolidate_outputs("comments")
```

Results and Discussion

Dataset Overview and Observation

The comments dataset was more fragmented, consisting of short user responses with informal tone and link references. On the other hand, the captions dataset consisted of full-length instructional sentences, resulting in longer baskets and richer vocabulary. The histogram of basket lengths illustrated this clearly. While comment transactions clustered below 20 tokens, caption transactions extended beyond 900 tokens in some cases. This difference directly influenced the pattern discovery stage, as longer baskets allowed more frequent co-occurrences and higher chances of finding valid itemsets.

Token Frequency and Distribution

The comments show a dominant presence of external references which include hyperlink indicators like *"http"* and resource-sharing elements such as *"youtube"* and *"spotify"* and *"href"* and *"com"*. The comments show a collection of unconnected keywords that users have created. The captions contain numerous instructional terms as well as action-oriented words which include *"going", "use", "see", "get", "know",* and *"make"*. The word clouds showed that captions contain more unified educational

Manual Pattern Exploration

The comments network shows multiple separate clusters that contain technical terms like *“api”, “spotify”, “twilio”*, and *“developer”* which reflect practical discussions between viewers. The words “thank” and “course” which appeared on the edges of the text demonstrated users showed appreciation and active participation. The captions network showed a more compact organization which focused on the words *“use”, “get”, “let”,* and *“know”* because the instructor maintained a steady instructional style. The network structure shows fewer nodes but more connections which indicates that teaching terms appear repeatedly throughout the text creating stronger thematic coherence.

Aspect	**Comments Network**	**Captions Network**
Nodes	80	45
Edges	367	399
Pattern Type	Fragmented, practical	Structured, instructional

Table 5: Comparison of Co-occurrence Network Characteristics

Algorithmic Pattern Discovery

The Apriori algorithm operated on processed data to analyze both captions and comments through support thresholds that ranged from 0.30 to 0.05. The analysis found that only captions generated frequent itemsets at support levels of 0.3 or above because comments did not contain enough repeated tokens to produce any rules. The process generated thousands of new combinations from captions which maintained high confidence values between 0.75 and 1.00 as the threshold decreased. The patterns from comments were fewer but they maintained confidence scores that exceeded 0.9.

The results demonstrate that captions use instructional language with fixed phrases *like "api → request"* and *"use → client"* but comments show casual language through phrases such as *"thank → course"* and *"spotify → api".*

Dataset	**Support Range**	**Itemsets Found**	**Confidence Range**	**Example Rule**
Captions	0.30–0.05	↑ from 229 to 510,000+	0.70–1.00	api → request
Comments	0.30–0.05	↓ below 200	0.90–1.00	thank → course
Merged	0.10–0.05	None (no 2–3 itemsets)	–	–

Table 6: Summary of Apriori Results Across Datasets and Support Thresholds

Exploration Using Variations

(a) Stemming vs. Lemmatization

The transition from lemmatization to stemming resulted in captions producing over 11 million itemsets at 0.05 support because word forms were combined during the process. The process created more detailed information, but it became harder to understand because it combined different forms of the same word such as use and using and used. The comments that were posted showed strong confidence, but they appeared in low numbers and did not show much diversity.

(b) Token Length Filtering (len ≥ 4)

The exclusion of brief tokens from the dataset resulted in improved noise reduction which made the data more understandable. The captions_len4 section keeps its main connections between *"client → use"* and *"api → code"*. The analysis of Comments_len4 revealed a limited number of stable rules which appeared mainly between specialized terms such as *"spotify → twilio"*.

(c) Merged Dataset

The analysis of combined datasets revealed no important multi-item rules which proved that narrative captions and informal comments operate as separate language styles.

Reflection

(a) What surprising pair or rule did you find?

One surprising pattern was *“api → request”* appearing consistently in captions, capturing the course’s technical focus. Meanwhile, *“thank → course”* in comments reflected emotional engagement from viewers.

(b) What was challenging about discovering patterns?

The main challenge was data imbalance. Comments were sparse, so many thresholds produced no frequent itemsets. Lowering the support solved this but generated excessive combinations that required further filtering.

(c) What would you explore next?

Future work could combine sentiment and topic modeling to interpret how viewers feel about specific concepts. Tracking these patterns over time could also show how audience understanding evolves.

Association Insights and Visualization

The association rule mining revealed clear contrasts between captions and comments in their lexical and structural patterns. Captions showed instructional and contextual language, while comments reflected more technical URL-like text.

Caption Patterns

The data shows strong 2-item and 3-item associations across all support levels (0.30–0.05) when using captions. The high support level results revealed that the most common phrases were e **“going & let”**, **“let & right”**, and **“going & say”** which demonstrated the repetitive instructional nature of coding tutorials.

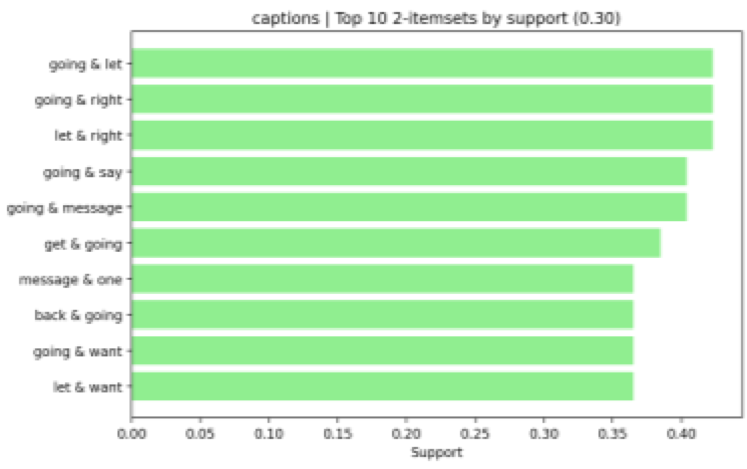

Figure 11 (a) 2 itemset (0.30)

The rules at lower support levels included technical terms like **“check → click”**, **“number → link”**, and **“curl → request”** which demonstrated real-world coding examples. The confidence interval maintained its high values between 0.85 and 1.00.

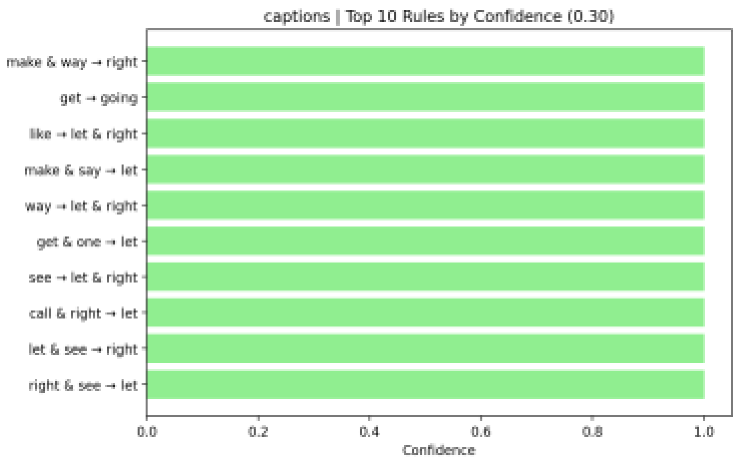

Figure 11 (b) 3 itemset (0.30)

Comment Patterns

Meaningful comment patterns appeared only at low supports (≤ 0.10). The analysis of top pairs including **href & http**, **com & youtube**, and **http & www** showed that numerous comments contained web fragments or embedded code.

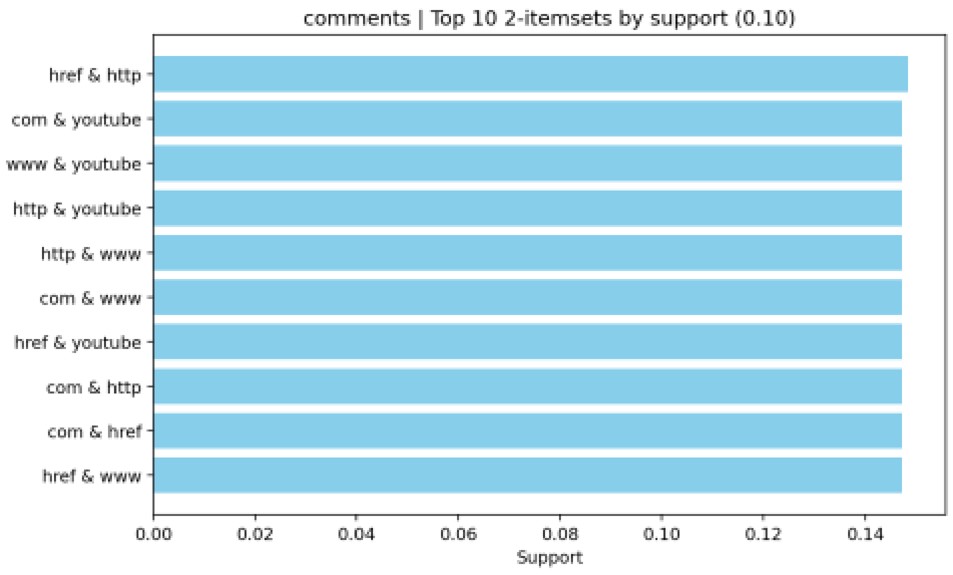

Figure 12 (a) 2 itemset (0.10)

The analysis at 0.05 support revealed one specific rule which appeared frequently in the data: **“amp & www → wxsd”**.

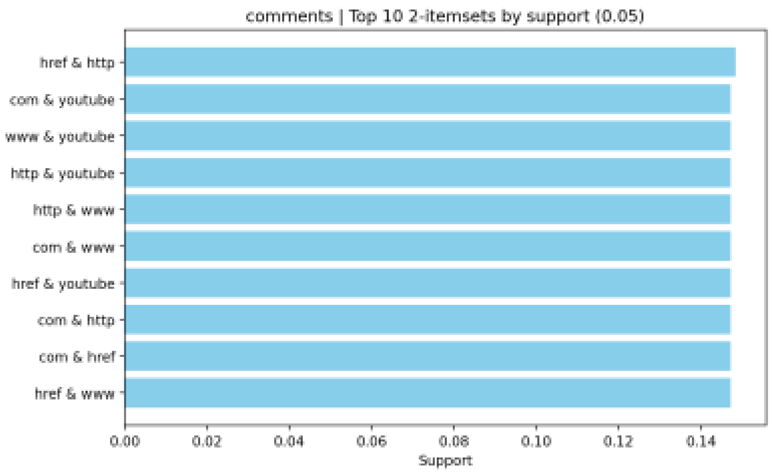

Figure 12 (b) 2 itemset (0.05)

The association network analysis showed that the terms youtube, http, and href existed in close-knit groups which matched hyperlink structures instead of language-based meanings.

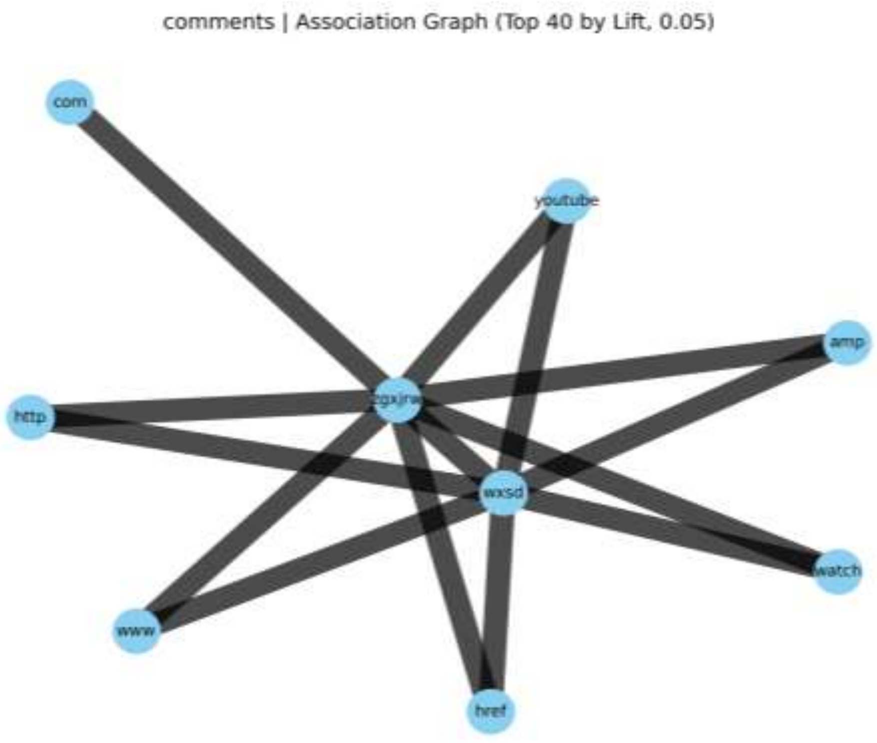

Figure 13: Comments 0.05 – Association Graph

Conclusion

This laboratory successfully extended the YouTube text-mining workflow from data cleaning to pattern discovery using both manual and algorithmic methods. The study showed that captions and comments use different types of language because captions employ organized instructional text which leads to reliable association rules and comments show broken content with mainly links and technical elements. The study found that Apriori analysis revealed multiple support levels which identified two main patterns in captions: ***api*** → ***request*** and ***client*** → ***use*** and one high-confidence pattern in comments: ***thank*** → ***course***. The research revealed how different support thresholds together with stemming techniques and short token removal affect the equilibrium between interpretability and pattern richness. The research findings showed that support reduction leads to greater variety but generates duplicate content. The study demonstrated that association rule mining can identify both thematic and contextual links from unstructured text data. Future work should incorporate sentiment or topic modeling to better interpret the intent and tone behind these linguistic patterns.