Final project guidelines

Note: Use these guidelines if and only if you are pursuing a **final project of your own design**. For those taking the final exam instead of the project, see the (separate) final exam notebook.

The task

Your task is to: identify an interesting problem connected to the humanities or humanistic social sciences that's addressable with the help of computational methods, formulate a hypothesis about it, devise an experiment or experiments to test your hypothesis, present the results of your investigations, and discuss your findings.

These tasks essentially replicate the process of writing an academic paper. You can think of your project as a paper in miniature.

You are free to present each of these tasks as you see fit. You should use narrative text (that is, your own writing in a markdown cell), citations of others' work, numerical results, tables of data, and static and/or interactive visualizations as appropriate. Total length is flexible and depends on the number of people involved in the work, as well as the specific balance you strike between the ambition of your question and the sophistication of your methods. But be aware that numbers never, ever speak for themselves. Quantitative results presented without substantial discussion will not earn high marks.

Your project should reflect, at minimum, ten **or more** hours of work by each participant, though you will be graded on the quality of your work, not the amount of time it took you to produce it. Most high-quality projects represent twenty or more hours of work by each member.

Pick an important and interesting problem!

No amount of technical sophistication will overcome a fundamentally uninteresting problem at the core of your work. You have seen many pieces of successful computational humanities research over the course of the semester. You might use these as a guide to the kinds of problems that interest scholars in a range of humanities disciplines. You may also want to spend some time in the library, reading recent books and articles in the professional literature. **Problem selection and motivation are integral parts of the project.** Do not neglect them.

Format

You should submit your project as a PDF document created using the included $\LaTeX$ template. Consult the template for information on formatting and what is expected in each section. You can use your favorite text editor or something like Overleaf to edit this document. You will also submit this Jupyter notebook, along with all data necessary to reproduce your analysis. If your dataset is too large to share easily, let us know in advance so that we can find a workaround.

All code used in the project should be present in the notebook (except for widely-available libraries that you import), but **be sure that we can read and understand your report in full without rerunning the code**.

Because you are submitting essentially a mini-paper in the PDF writeup, I don't have any particular formatting expections for written material in this notebook. However, you should include **all code used when completing the final project, with comments added for clarity**. It should be straightforward to map code from the notebook to sections/figures/results in your paper, and vice versa.

Grading

This project takes the place of the take-home final exam for the course. It is worth 35% of your overall grade. You will be graded on

the quality and ambition of each aspect of the project. No single component is more important than the others.

Practical details

- The project is due at **4:30 PM EST on Wednesday, December 17** via upload to CMS of a single zip file containing your fully executed Jupyter notebook report, a PDF copy of the notebook, and all associated data. **You may not use slip days for the final project or exam**.
- You may work alone or in a group of up to three total members.
  - If you work in a group, be sure to list the names of the group members.
  - For groups, create your group on CMS and submit one notebook for the entire group. **Each group should also submit a statement of responsibility** that describes in general terms who performed which parts of the project.
- You may post questions on Ed, but should do so privately (visible to course staff only).

Your info

- NetID(s): kgc42
- Name(s): Kyle Chu

In [1]:
```
import pandas as pd
import numpy as np
import kagglehub
import os
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import spacy

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import re
```

```
from scipy import stats
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
import seaborn as sns

from sklearn.linear_model import LogisticRegression
```

Step 1: Data Loading & Filtering

We'll load the Steam Reviews dataset, filter to Rainbow Six Siege English reviews, and remove short reviews. Class balancing will be performed after sentiment analysis, once we have filtered to negative reviews only.

In [2]:
```
path = kagglehub.dataset_download("najzeko/steam-reviews-202
print("Path to dataset files:", path)

GAME_APPID = 359550
```

```
Path to dataset files: /Users/kych2204/.cache/kagglehub/datas
ets/najzeko/steam-reviews-2021/versions/1
```

Load in the data

In [3]:
```
# load the CSV file

csv_path = os.path.join(path, "steam_reviews.csv")
print(f"Loading data from {csv_path}...")

chunk_size = 50000
chunks = []
total_read = 0
chunks_processed = 0

for chunk in pd.read_csv(csv_path, chunksize=chunk_size, low
    chunks_processed += 1

    chunk['app_id'] = pd.to_numeric(chunk['app_id'], errors=

    chunk_filtered = chunk[
        (chunk['app_id'] == GAME_APPID) &
        (chunk['language'].str.lower() == 'english')
    ].copy()
```

```
    if len(chunk_filtered) > 0:
        chunks.append(chunk_filtered)
        total_read += len(chunk_filtered)

df = pd.concat(chunks, ignore_index=True)
print(f"\nTotal Rainbow Six Siege English reviews loaded: {le
```

```
Loading data from /Users/kych2204/.cache/kagglehub/datasets/n
ajzeko/steam-reviews-2021/versions/1/steam_reviews.csv...

Total Rainbow Six Siege English reviews loaded: 371,154
```

Clean and filter the data

In [4]:
```
df['recommended'] = df['recommended'].astype(bool)

# remove null/empty reviews
initial_count = len(df)
df = df[df['review'].notna()].copy()
df = df[df['review'].str.strip() != ''].copy()
print(f"After removing null/empty reviews: {len(df):,} (remov

# only keep reviews from users who purchased the game
before_purchase_filter = len(df)
df = df[df['steam_purchase'] == True].copy()
print(f"After filtering to steam_purchase == True: {len(df):,

# remove very short reviews under 10 words
df['token_count'] = df['review'].str.split().str.len()
df = df[df['token_count'] >= 10].copy()
print(f"After removing reviews with < 10 tokens: {len(df):,}"

# check class distribution
print(f"\nClass distribution:")
print(df['recommended'].value_counts())

df.head()
```

```
After removing null/empty reviews: 370,156 (removed 998)
After filtering to steam_purchase == True: 312,169 (removed 5
7,987)
After removing reviews with < 10 tokens: 108,264

Class distribution:
recommended
True     90053
False    18211
Name: count, dtype: int64
```

Out[4]:

	Unnamed: 0	app_id	app_name	review_id	language	review
3	5860060	359550	Tom Clancy's Rainbow Six Siege	84794099	english	servers are shit actual shit but the game itse...
5	5860063	359550	Tom Clancy's Rainbow Six Siege	84793878	english	i do recomend this game bc it is rlly fun anf ...
9	5860080	359550	Tom Clancy's Rainbow Six Siege	84792481	english	This game is just awesome, the graphics and ev...
11	5860083	359550	Tom Clancy's Rainbow Six Siege	84792431	english	this game drives me crazy, but im still playin...
19	5860098	359550	Tom Clancy's Rainbow Six Siege	84791184	english	A good game overall. The only downside that I ...

5 rows × 24 columns

In [5]:
```
# remove unneeded columns
keep_cols = ['review', 'recommended', 'app_id', 'language',

df = df[keep_cols].copy()

print(f"\nDataset shape: {df.shape}")

df.head()
```

```
Dataset shape: (108264, 6)
```

Out[5]:

	review	recommended	app_id	language	token_count	times
3	servers are shit actual shit but the game itse...	True	359550	english	44	
5	i do recomend this game bc it is rlly fun anf ...	True	359550	english	16	
9	This game is just awesome, the graphics and ev...	True	359550	english	117	
11	this game drives me crazy, but im still playin...	False	359550	english	18	
19	A good game overall. The only downside that I ...	True	359550	english	67	

Step 2: Sentiment Analysis & Negative Subset

We'll use VADER sentiment analysis to identify negative reviews, then filter to only negative reviews for our analysis.

In [6]:

```
try:
    nltk.data.find('vader_lexicon')
```

```
except LookupError:
    nltk.download('vader_lexicon')

sia = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/kych2204/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [7]:
```
# Configuration for sentiment analysis
NEG_THRESHOLD = -0.2
```

In [8]:
```
# Compute sentiment scores for each review
print("Computing sentiment scores...")
df['sentiment_compound'] = df['review'].apply(lambda x: sia.p

print(f"Sentiment score statistics:")
print(f"  Mean: {df['sentiment_compound'].mean():.3f}")
print(f"  Median: {df['sentiment_compound'].median():.3f}")
print(f"  Min: {df['sentiment_compound'].min():.3f}")
print(f"  Max: {df['sentiment_compound'].max():.3f}")
```

```
Computing sentiment scores...
Sentiment score statistics:
  Mean: 0.394
  Median: 0.637
  Min: -1.000
  Max: 1.000
```

In [9]:
```
# Filter to negative reviews only
print(f"\nFiltering to negative reviews (sentiment_compound <
df_negative = df[df['sentiment_compound'] < NEG_THRESHOLD].co

print(f"Total negative reviews: {len(df_negative):,}")
print(f"Removed {len(df) - len(df_negative):,} non-negative

print(f"\nClass distribution in negative reviews:")
print(df_negative['recommended'].value_counts())
```

```
Filtering to negative reviews (sentiment_compound < -0.2)...
Total negative reviews: 20,910
Removed 87,354 non-negative reviews

Class distribution in negative reviews:
recommended
True     11461
False     9449
Name: count, dtype: int64
```

In [10]:
```
# balance datasets

negative_recommended = df_negative[df_negative['recommended']
negative_not_recommended = df_negative[df_negative['recommend

print(f"\nNegative + Recommended before balancing: {len(negat
print(f"Negative + Not Recommended before balancing: {len(neg

n_sample = min(len(negative_recommended), len(negative_not_re
print(f"\nSampling {n_sample:,} reviews from each class...")

negative_rec_sample = negative_recommended.sample(n=n_sample,
negative_not_rec_sample = negative_not_recommended.sample(n=n

df_negative_balanced = pd.concat([negative_rec_sample, negati
df_negative_balanced = df_negative_balanced.sample(frac=1, ra

print(f"\nFinal balanced negative dataset: {len(df_negative_b
print(f"Class distribution after balancing:")
print(df_negative_balanced['recommended'].value_counts())
```

```
Negative + Recommended before balancing: 11,461
Negative + Not Recommended before balancing: 9,449

Sampling 9,449 reviews from each class...

Final balanced negative dataset: 18,898 reviews
Class distribution after balancing:
recommended
False    9449
True     9449
Name: count, dtype: int64
```

In [11]:
```
# display 10 random reviews from balanced dataset of negative
print("Sample of 10 negative reviews (balanced dataset):")
print("=" * 80)
```

```
sample_reviews = df_negative_balanced.sample(10)

for idx, row in sample_reviews.iterrows():
    rec_status = "RECOMMENDED" if row['recommended'] else "NO
    sentiment = row['sentiment_compound']
    print(f"\n[{rec_status}] (Sentiment: {sentiment:.3f})")
    print(f"Review: {row['review']}")
    print("-" * 80)
```

```
Sample of 10 negative reviews (balanced dataset):
================================================================
==================

[RECOMMENDED] (Sentiment: -0.930)
Review: this is not an easy FPS game for newbie, if you have
experiences about FPS game, try it. If not, dont buy it becau
se you will be very angry and dont know why you die and how y
ou die. Anyway, it is a hardcore game.
----------------------------------------------------------------
------------------

[NOT RECOMMENDED] (Sentiment: -0.859)
Review: This game is the worst tryhard trash ive ever seen. I
hope every developer and every player of this game will get b
onecancer and their death will be fuckin painfully
----------------------------------------------------------------
------------------

[NOT RECOMMENDED] (Sentiment: -0.426)
Review: Its a great game but the only problem I have it that
people get matched up with people who are a higher level then
you and your teammates, I see people who are 4x higher then m
y level and most of the time my team loses, not because the
y're bad, its because the higher level people has more skills
then we low level do.


Over all I would rate this game a 6.5/10
----------------------------------------------------------------
------------------

[NOT RECOMMENDED] (Sentiment: -0.947)
Review: The game would be so much better if all the players w
ould have some actual integrity and sportsmanship. This is pr
obably the most toxic community I've ever been into. There ar
e players who will straight up tell you to uninstall because
they killed you once, without any actual reason to be toxic.
Then they will proceed to comment on everything you do and al
ways come with the same lame argument: "i'M nOt eVeN tRyInG".
What's hilarious is that these players think and pretend to b
e good at the game by playing the same basic operators which
require just 2 brain cells to use, operators which require no
skill, I'm talking about Jager, Bandit, Ash etc. Some of them
have straight up the worst Ranked Stats, I'm talking about 0.
5/0.6 win rate and k/d. Their only argument and the most pred
ictable one with which they try to hit back is: "hAhA yOu gOt
```

KiLlEd bY sOmEoNe wOrSe tHeN yOu". Is that supposed to offens
ive? Just because your stats are bad doesn't mean killing som
ebody makes him/her bad at the game. Anybody can kill somebod
y in a video game, it's not written in stone that good player
s can't get killed by bad players. Important note: this happe
ns in casual. Yes, all of this happens in casual. It's so sad
and pathetic seeing people do this just so they can feel abov
e a random person they just found online through a video game
from which nobody gains nothing (obviously). I can't even ima
gine how many insecurities these neckbeards have. Maybe I'm t
oo normal to understand them because I play games for fun, bu
t I guess I picked the wrong game.

Leaving the biggest downside of the game aside, I've also enc
ountered multiple times some bugs regarding the sound. In one
game I simply could not hear explosions, gunfire or footstep
s, just the interface of the game when pressing "Esc" and som
e effects.

The only fun part of this game is creating a private game and
playing just with friends, the rest of the game is usually st
ressful and making me more and more allergic to stupidity.

[NOT RECOMMENDED] (Sentiment: -0.587)
Review: The worst game i have ever played in the history of m
y gaming career. Would not recommend.

[RECOMMENDED] (Sentiment: -0.226)
Review: in the beginning it wasn't good enough. Then it was c
hanged. many operators were added and the game play has been
improved as well. BUT, the toxicity and racism of the communi
ty (North American Servers at least) hasn't been addressed al
most at all. Hopefully the environment will soon be unsafe fo
r such behaviors.

[RECOMMENDED] (Sentiment: -0.727)
Review: One of the best 1st person shooters in a long time in
my opinion. I mean who doesn't like to tear down walls and ki
ll the enemies on the other side!


```
[RECOMMENDED] (Sentiment: -0.689)
Review: if u want a fucking jage to fucking spawn kill u this
is a game for u
--------------------------------------------------------------
------------------

[RECOMMENDED] (Sentiment: -0.421)
Review: this is the game ive been looking to waste my life an
d money on
--------------------------------------------------------------
------------------

[NOT RECOMMENDED] (Sentiment: -0.258)
Review: A 5 year old game and I can't play more than 3 matche
s without game breaking bugs. I went completely deaf my last
game.
--------------------------------------------------------------
------------------
```

In [12]:
```
# turn recommended column into binary
df_negative_balanced['recommended'] = df_negative_balanced['

df = df_negative_balanced.copy()

print(f"\nFinal dataset for analysis: {len(df):,} negative re
print(f"Columns: {df.columns.tolist()}")
df.head()
```

```
Final dataset for analysis: 18,898 negative reviews
Columns: ['review', 'recommended', 'app_id', 'language', 'tok
en_count', 'timestamp_created', 'sentiment_compound']
```

Out[12]:

	review	recommended	app_id	language	token_count	timesta
0	Fucking shit game, a disgrace to Tom Clancy's ...	0	359550	english	14	
1	Honestly filled with annoying ass trolls and s...	0	359550	english	33	
2	I have so many hours into this broken game. Sa...	1	359550	english	17	
3	with 1k hours, i can now say, that i fucking h...	1	359550	english	21	
4	this game is full of bugs, its so fuckin shit ...	0	359550	english	38	

Step 3: Text Preprocessing & Feature Engineering

We'll preprocess the text, create lexicon-based features for bugs/performance, monetization/price, and gameplay/fun, and fit a topic model.

Text Preprocessing

In [13]:
```
def preprocess_text(text):
    '''Clean text for feature extraction'''
    text = text.lower()
    text = re.sub(r'http\S+|www\S+', '', text)
    return text

df['clean_text'] = df['review'].apply(preprocess_text)
print(f"Preprocessed {len(df):,} reviews")

nlp = spacy.load("en_core_web_sm")

def lemmatize_text(text):
    doc = nlp(text)
    lemmas = [token.lemma_ for token in doc if token.is_alpha
    return " ".join(lemmas)

df['clean_text'] = df['clean_text'].apply(lemmatize_text)
print(f"Lemmatized {len(df):,} reviews for topic modeling")

df.head()
```

```
Preprocessed 18,898 reviews
Lemmatized 18,898 reviews for topic modeling
```

Out[13]:

	review	recommended	app_id	language	token_count	timesta
0	Fucking shit game, a disgrace to Tom Clancy's ...	0	359550	english	14	
1	Honestly filled with annoying ass trolls and s...	0	359550	english	33	
2	I have so many hours into this broken game. Sa...	1	359550	english	17	
3	with 1k hours, i can now say, that i fucking h...	1	359550	english	21	
4	this game is full of bugs, its so fuckin shit ...	0	359550	english	38	

Lexicon-Based Features

In [14]:
```
# Bug / Performance complaints
QUALITY_LEXICON = [
    "crash",
    "lag", "laggy",
    "fps",
    "bug",
```

```
    "glitch",
    "freeze",
    "stutter",
    "performance",
    "optimization",
    "optimize",
    "broken",
    "unplayable",
    "frame",
    "server",
    "disconnect",
    "ping",
    "latency",
    "drop",
]

# Monetization / Price complaints
MONETIZATION_LEXICON = [
    "microtransaction",
    "mtx",
    "lootbox",
    "dlc",
    "pay2win",
    "pay-to-win",
    "overpriced",
    "refund",
    "sale",
    "expensive",
    "price",
    "cost",
    "money",
    "greedy",
    "greed",
    "monetization",
    "premium",
    "skin",
    "cosmetic",
]

# Gameplay / Fun / Content
GAMEPLAY_LEXICON = [
    "fun",
    "gameplay",
    "combat",
    "balance",
    "content",
    "story",
```

```
    "graphic",
    "soundtrack",
    "music",
    "replay",
    "grind",
    "coop",
    "friend",
    "enjoyable",
    "enjoy",
    "addictive",
    "satisfying",
    "satisfaction",
    "mechanic",
    "skill",
    "competitive",
    "rank",
    "matchmaking",
    "strategy",
    "tactical",
    "teamwork",
    "operator",
    "map",
    "weapon",
]
```

In [15]:
```
def count_lexicon_words(text, lexicon):
    tokens = text.split()
    count = sum(1 for token in tokens if token in lexicon)
    return count

df['quality_count'] = (df['clean_text'].apply(lambda x: coun
df['money_count'] = (df['clean_text'].apply(lambda x: count_
df['gameplay_count'] = (df['clean_text'].apply(lambda x: cou

df['quality_count'] = df['quality_count'].fillna(0)
df['money_count'] = df['money_count'].fillna(0)
df['gameplay_count'] = df['gameplay_count'].fillna(0)

df['exclamation_count'] = df['review'].apply(lambda x: x.cou

PROFANITY_LIST = ['shit', 'fuck', 'damn', 'crap', 'hell', 'a
df['profanity_count'] = df['clean_text'].apply(lambda x: sum

df.head()
```

Out[15]:

	review	recommended	app_id	language	token_count	timesta
0	Fucking shit game, a disgrace to Tom Clancy's ...	0	359550	english	14	
1	Honestly filled with annoying ass trolls and s...	0	359550	english	33	
2	I have so many hours into this broken game. Sa...	1	359550	english	17	
3	with 1k hours, i can now say, that i fucking h...	1	359550	english	21	
4	this game is full of bugs, its so fuckin shit ...	0	359550	english	38	

Topic Modeling

In [16]:
```
# Vectorize lemmatized text for topic modeling
vectorizer = CountVectorizer(
    min_df=5,
    max_df=0.8,
    max_features=5000,
    lowercase=False,
```

```
    stop_words='english',
    token_pattern=r'\b[a-z]+\b'
)

X_counts = vectorizer.fit_transform(df['clean_text'])
print(f"Vocabulary size: {len(vectorizer.get_feature_names_ou
print(f"Document-term matrix shape: {X_counts.shape}")
```

```
Vocabulary size: 5,000
Document-term matrix shape: (18898, 5000)
```

In [17]:
```
N_TOPICS = 7
```

In [18]:
```
print(f"\nFitting LDA model with {N_TOPICS} topics...")
lda = LatentDirichletAllocation(
    n_components=N_TOPICS,
    random_state=42,
    max_iter=20,
    learning_method='batch'
)

topic_model = lda.fit(X_counts)
print("Topic model fitted")

topic_proportions = lda.transform(X_counts)
print(f"Topic proportions shape: {topic_proportions.shape}")
```

```
Fitting LDA model with 7 topics...
Topic model fitted
Topic proportions shape: (18898, 7)
```

In [19]:
```
# Add topic proportion features to dataframe
for i in range(N_TOPICS):
    df[f'topic_{i}'] = topic_proportions[:, i]

print(f"Added {N_TOPICS} topic proportion features")

df.head()
```

```
Added 7 topic proportion features
```

Out[19]:

	review	recommended	app_id	language	token_count	timesta
0	Fucking shit game, a disgrace to Tom Clancy's ...	0	359550	english	14	
1	Honestly filled with annoying ass trolls and s...	0	359550	english	33	
2	I have so many hours into this broken game. Sa...	1	359550	english	17	
3	with 1k hours, i can now say, that i fucking h...	1	359550	english	21	
4	this game is full of bugs, its so fuckin shit ...	0	359550	english	38	

In [20]:

```
feature_names = vectorizer.get_feature_names_out()

print(f"\nTop 10 words for each of {N_TOPICS} topics:")
print("=" * 80)
for topic_idx in range(N_TOPICS):
    top_words_idx = topic_model.components_[topic_idx].argso
    top_words = [feature_names[i] for i in top_words_idx]
    top_weights = [topic_model.components_[topic_idx][i] for
```

```
    print(f"\nTopic {topic_idx}:")
    for word, weight in zip(top_words, top_weights):
        print(f"  {word:15s} {weight:.4f}")
```

```
Top 10 words for each of 7 topics:
============================================================
==================

Topic 0:
  game            3337.8344
  buy             2023.3807
  operator        1419.8778
  edition         897.1421
  money           773.8313
  starter         732.1420
  pay             707.1234
  like            673.0530
  just            551.2209
  rainbow         513.9176

Topic 1:
  shoot           2353.2717
  kill            1962.0678
  wall            1072.7551
  enemy           924.1449
  die             868.8772
  head            729.5928
  team            555.7396
  hostage         511.7692
  bullet          475.0468
  headshot        451.3103

Topic 2:
  ban             3162.3184
  game            1461.7601
  say             1252.4946
  ubisoft         1174.7551
  people          876.4793
  word            743.1414
  just            639.8539
  chat            582.4904
  toxic           527.6792
  make            496.6211

Topic 3:
  game            9238.1728
  play            3595.9938
  bad             1730.3633
  good            1519.4262
  just            1233.5854
  like            1184.6457
```

```
  fun             1143.1626
  hate            1124.1983
  hour            1082.8317
  time            921.4525

Topic 4:
  shit            4490.1411
  game            4025.1874
  fuck            1775.5565
  server          1444.2049
  broken          1435.9621
  fucking         1195.8517
  u               1055.7627
  ubisoft         871.5515
  rank            764.0519
  fix             742.2058

Topic 5:
  game            10145.2935
  bad             1935.3863
  gun             1865.0833
  play            1690.8638
  just            1589.6303
  bug             1573.4294
  fix             1517.2368
  make            1388.8875
  issue           1333.4542
  new             1294.9800

Topic 6:
  game            6217.3254
  team            3128.1793
  play            2590.4302
  player          2276.8398
  kill            1972.7674
  match           1584.0719
  time            1411.1408
  people          1197.6837
  like            1176.7799
  just            1157.0121
```

In [21]:
```
print(f"\nFinal dataset shape: {df.shape}")
feature_cols = ['quality_count', 'money_count', 'gamemplay_c
topic_cols = [f'topic_{i}' for i in range(N_TOPICS)]
all_features = feature_cols + topic_cols
```

```
print(f"Total number of features: {len(all_features)}")
print(f"Total features:{all_features}")
```

```
Final dataset shape: (18898, 20)
Total number of features: 13
Total features:['quality_count', 'money_count', 'gamemplay_co
unt', 'token_count', 'exclamation_count', 'profanity_count',
'topic_0', 'topic_1', 'topic_2', 'topic_3', 'topic_4', 'topic
_5', 'topic_6']
```

Step 4: Descriptive Analysis

We'll compare lexicon features and topic proportions between negative reviews that still recommend the game and those that do not. This gives an initial, lecture-style view of whether gameplay, bugs/performance, and monetization talk differ across the two groups before fitting a formal regression model.

In [22]:
```
print("Group sizes (negative reviews only):")
print(df['recommended'].value_counts().rename(index={0: 'Not

lex_cols = ['quality_count', 'money_count', 'gameplay_count']
summary_lex = df.groupby('recommended')[lex_cols].agg(['mean
summary_lex
```

```
Group sizes (negative reviews only):
recommended
Not recommended    9449
Recommended        9449
Name: count, dtype: int64
```

Out[22]:

	quality_count			money_co		
	mean	std	median	mean	std	med
recommended						
0	1.223079	2.914038	0.0	0.493389	1.418120	
1	0.710627	2.128585	0.0	0.210845	1.026913	

In [23]:
```
# t-test and p-value significance results
rows = []
for col in lex_cols:
    g_rec = df.loc[df['recommended'] == 1, col]
```

```
    g_not = df.loc[df['recommended'] == 0, col]
    t_stat, p_val = stats.ttest_ind(g_rec, g_not, equal_var=F
    rows.append({
        'feature': col,
        'mean_rec': g_rec.mean(),
        'mean_not': g_not.mean(),
        't_stat': t_stat,
        'p_value': p_val,
    })

lex_test_results = pd.DataFrame(rows)
print("\nT-test results for lexicon features:")
lex_test_results
```

```
T-test results for lexicon features:
```

Out[23]:

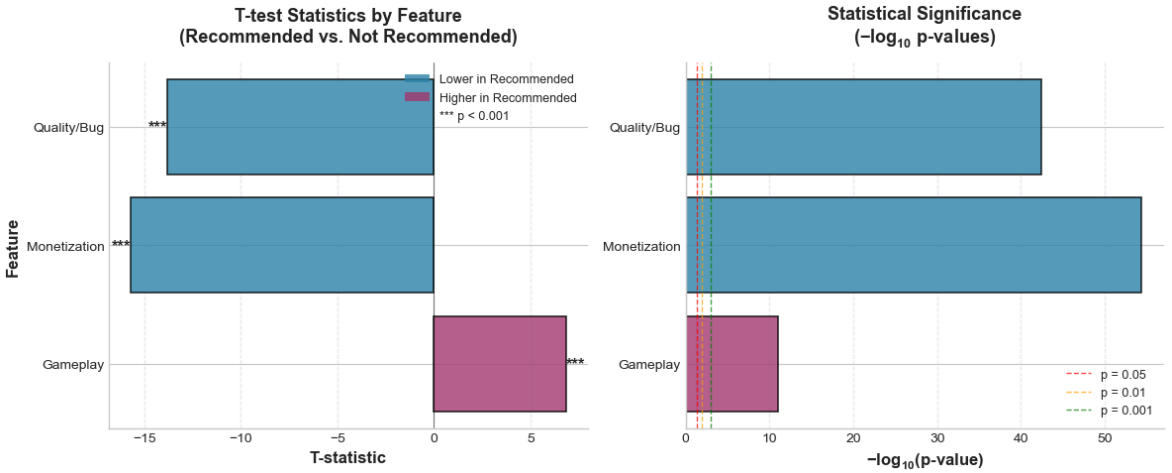
	feature	mean_rec	mean_not	t_stat	p_value
0	quality_count	0.710627	1.223079	-13.803817	4.091839e-43
1	money_count	0.210845	0.493389	-15.686327	4.513015e-55
2	gameplay_count	1.688273	1.421307	6.835429	8.440462e-12

In [24]:

```
# graph for t-test and p-value significance results
try:
    plt.style.use('seaborn-v0_8-whitegrid')
except:
    try:
        plt.style.use('seaborn-whitegrid')
    except:
        plt.style.use('seaborn')
sns.set_palette("husl")

plt.rcParams['mathtext.default'] = 'regular'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

lex_test_results['feature_label'] = lex_test_results['feature
lex_test_results['feature_label'] = lex_test_results['feature
    'Quality': 'Quality/Bug',
    'Money': 'Monetization',
    'Gameplay': 'Gameplay'
})
```

```
lex_test_results_sorted = lex_test_results.copy()
lex_test_results_sorted['is_positive'] = lex_test_results_so
lex_test_results_sorted['abs_t_stat'] = lex_test_results_sor
lex_test_results_sorted = lex_test_results_sorted.sort_value
lex_test_results_sorted = lex_test_results_sorted.drop(['is_

colors = ['#2E86AB' if x < 0 else '#A23B72' for x in lex_tes
bars = ax1.barh(lex_test_results_sorted['feature_label'], le

for i, (idx, row) in enumerate(lex_test_results_sorted.iterr
    p_val = row['p_value']
    t_stat = row['t_stat']

    if p_val < 0.001:
        sig_text = '***'
    elif p_val < 0.01:
        sig_text = '**'
    elif p_val < 0.05:
        sig_text = '*'
    else:
        sig_text = ''

    x_pos = t_stat + (0.5 if t_stat > 0 else -0.5)
    ax1.text(x_pos, i, sig_text, ha='center' if t_stat > 0 e
             va='center', fontsize=12, fontweight='bold')

ax1.axvline(x=0, color='black', linestyle='-', linewidth=0.8

ax1.set_xlabel('T-statistic', fontsize=12, fontweight='bold'
ax1.set_ylabel('Feature', fontsize=12, fontweight='bold')
ax1.set_title('T-test Statistics by Feature\n(Recommended vs
ax1.grid(axis='x', alpha=0.3, linestyle='--')
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)

legend_elements = [
    Patch(facecolor='#2E86AB', alpha=0.8, label='Lower in Re
    Patch(facecolor='#A23B72', alpha=0.8, label='Higher in R
    plt.Line2D([0], [0], marker='', linestyle='', label='***
]
ax1.legend(handles=legend_elements, loc='upper right', fonts

lex_test_results_sorted['neg_log10_p'] = -np.log10(lex_test_
colors2 = ['#2E86AB' if x < 0 else '#A23B72' for x in lex_te

bars2 = ax2.barh(lex_test_results_sorted['feature_label'], l
                 color=colors2, alpha=0.8, edgecolor='black'
```

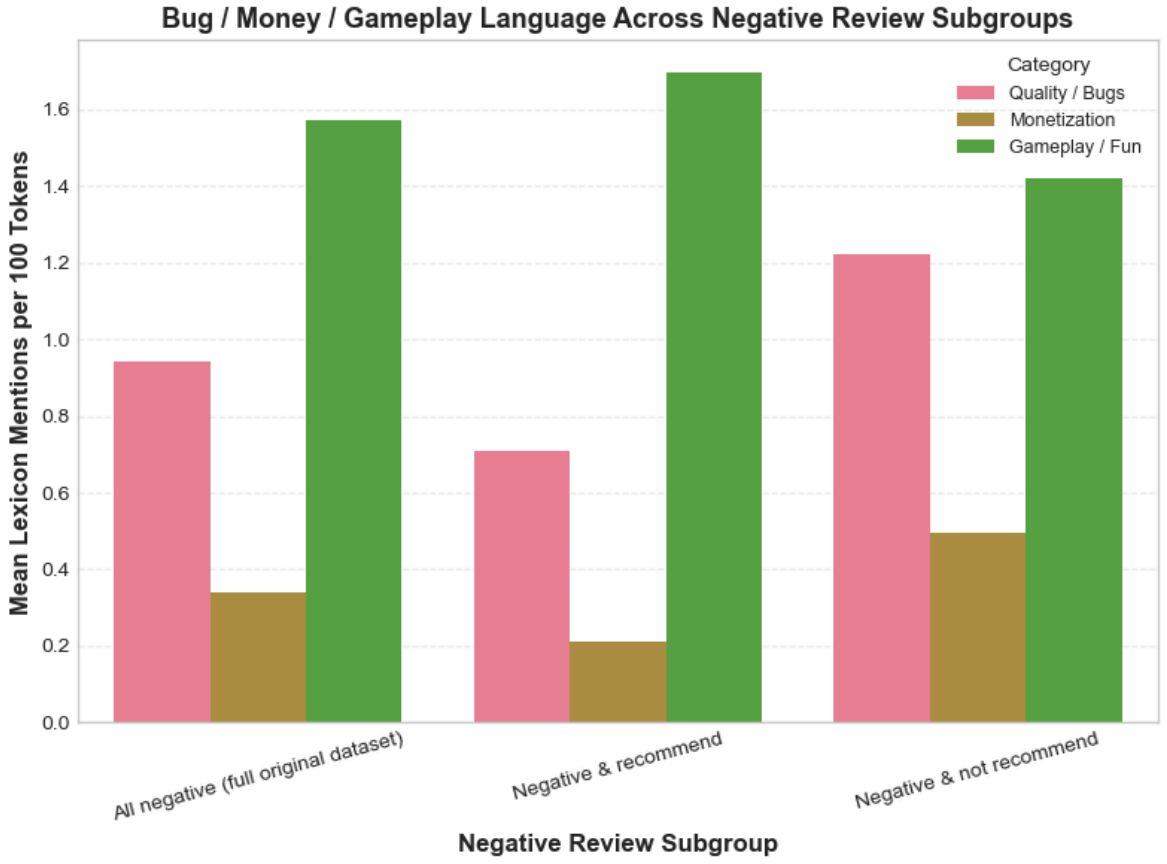
```
ax2.axvline(x=-np.log10(0.05), color='red', linestyle='--',
ax2.axvline(x=-np.log10(0.01), color='orange', linestyle='--
ax2.axvline(x=-np.log10(0.001), color='green', linestyle='--

ax2.set_xlabel(r'$-log_{10}$(p-value)', fontsize=12, fontweig
ax2.set_ylabel('')
ax2.set_title('Statistical Significance\n' + r'($-log_{10}$ p
ax2.grid(axis='x', alpha=0.3, linestyle='--')
ax2.spines['top'].set_visible(False)
ax2.spines['right'].set_visible(False)
ax2.legend(loc='lower right', fontsize=9, framealpha=0.9)

plt.tight_layout()
plt.savefig('plots/t_test_results.png', dpi=300, bbox_inches=
plt.show()
```



In [25]:
```
df_all_neg = df_negative.copy()

df_all_neg['clean_text'] = df_all_neg['review'].apply(preproc
df_all_neg['clean_text'] = df_all_neg['clean_text'].apply(len

def count_lexicon_words(text, lexicon):
    tokens = text.split()
    return sum(1 for token in tokens if token in lexicon)

df_all_neg['quality_count'] = (
    df_all_neg['clean_text'].apply(lambda x: count_lexicon_wo
    / df_all_neg['token_count'] * 100
)
df_all_neg['money_count'] = (
    df_all_neg['clean_text'].apply(lambda x: count_lexicon_wo
    / df_all_neg['token_count'] * 100
)
```

```
df_all_neg['gameplay_count'] = (
    df_all_neg['clean_text'].apply(lambda x: count_lexicon_w
    / df_all_neg['token_count'] * 100
)

df_all_neg[['quality_count', 'money_count', 'gameplay_count'
    ['quality_count', 'money_count', 'gameplay_count']
].fillna(0)

# Define groups
groups = {
    "All negative (full original dataset)": df_all_neg,
    "Negative & recommend": df_all_neg[df_all_neg['recommende
    "Negative & not recommend": df_all_neg[df_all_neg['recomm
}

rows = []
for gname, gdf in groups.items():
    rows.append({
        "group": gname,
        "Quality / Bugs": gdf['quality_count'].mean(),
        "Monetization": gdf['money_count'].mean(),
        "Gameplay / Fun": gdf['gameplay_count'].mean(),
    })

split_df = pd.DataFrame(rows)

split_long = split_df.melt(
    id_vars='group',
    value_vars=['Quality / Bugs', 'Monetization', 'Gameplay /
    var_name='Category',
    value_name='Percent_per_100_tokens'
)

plt.figure(figsize=(8, 6))
sns.barplot(
    data=split_long,
    x='group',
    y='Percent_per_100_tokens',
    hue='Category'
)

plt.xlabel('Negative Review Subgroup', fontsize=12, fontweigh
plt.ylabel('Mean Lexicon Mentions per 100 Tokens', fontsize=1
plt.title('Bug / Money / Gameplay Language Across Negative Re
plt.xticks(rotation=15)
plt.legend(title='Category', fontsize=9)
```

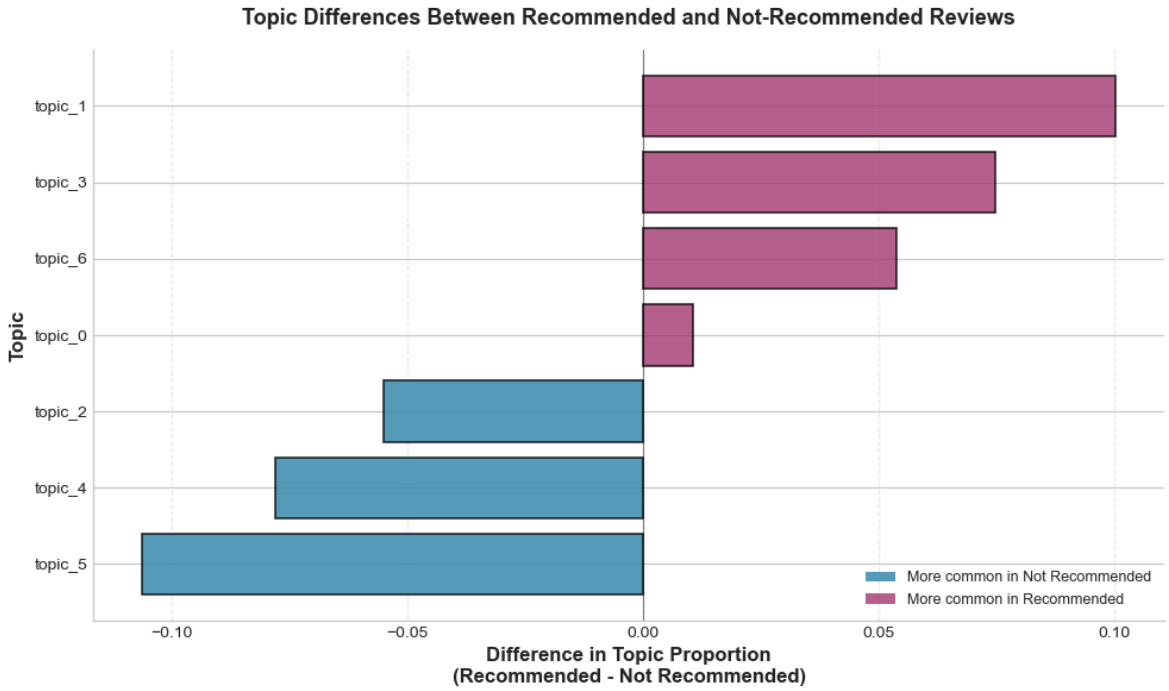
```
plt.grid(axis='y', alpha=0.3, linestyle='--')
plt.tight_layout()
plt.savefig('plots/lexicon_subgroups.png', dpi=300, bbox_incl
plt.show()
```



In [26]:
```
# identify which topics are most different between recommende

topic_cols = [c for c in df.columns if c.startswith('topic_')

topic_means = df.groupby('recommended')[topic_cols].mean().T
topic_means.columns = ['mean_not_recommended', 'mean_recommer
topic_means['diff_rec_minus_not'] = topic_means['mean_recomme
topic_means_sorted = topic_means.reindex(topic_means['diff_re
display(topic_means_sorted.head(10))

top_topics_for_model = list(topic_means_sorted.head(5).index

plt.style.use('seaborn-v0_8-whitegrid')

plt.rcParams['mathtext.default'] = 'regular'

topic_viz = topic_means_sorted.head(10).copy()
topic_viz = topic_viz.sort_values('diff_rec_minus_not', ascer
```

```
fig, ax = plt.subplots(figsize=(10, 6))

colors = ['#2E86AB' if x < 0 else '#A23B72' for x in topic_vi

bars = ax.barh(topic_viz.index, topic_viz['diff_rec_minus_not
               color=colors, alpha=0.8, edgecolor='black',

ax.axvline(x=0, color='black', linestyle='-', linewidth=0.8,

ax.set_xlabel('Difference in Topic Proportion\n(Recommended -
ax.set_ylabel('Topic', fontsize=12, fontweight='bold')
ax.set_title('Topic Differences Between Recommended and Not-R
ax.grid(axis='x', alpha=0.3, linestyle='--')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

legend_elements = [
    Patch(facecolor='#2E86AB', alpha=0.8, label='More common
    Patch(facecolor='#A23B72', alpha=0.8, label='More common
]

ax.legend(handles=legend_elements, loc='lower right', fontsiz

plt.tight_layout()
plt.savefig('plots/topic_differences.png', dpi=300, bbox_incl
plt.show()
```

	mean_not_recommended	mean_recommended	diff_rec_minus
topic_5	0.233699	0.127372	-0.10
topic_1	0.057900	0.158244	0.10
topic_4	0.170016	0.091894	-0.0
topic_3	0.179093	0.253846	0.07
topic_2	0.127686	0.072589	-0.05
topic_6	0.131555	0.185309	0.05
topic_0	0.100052	0.110747	0.01



Step 5: Logistic Regression for Hypothesis Testing

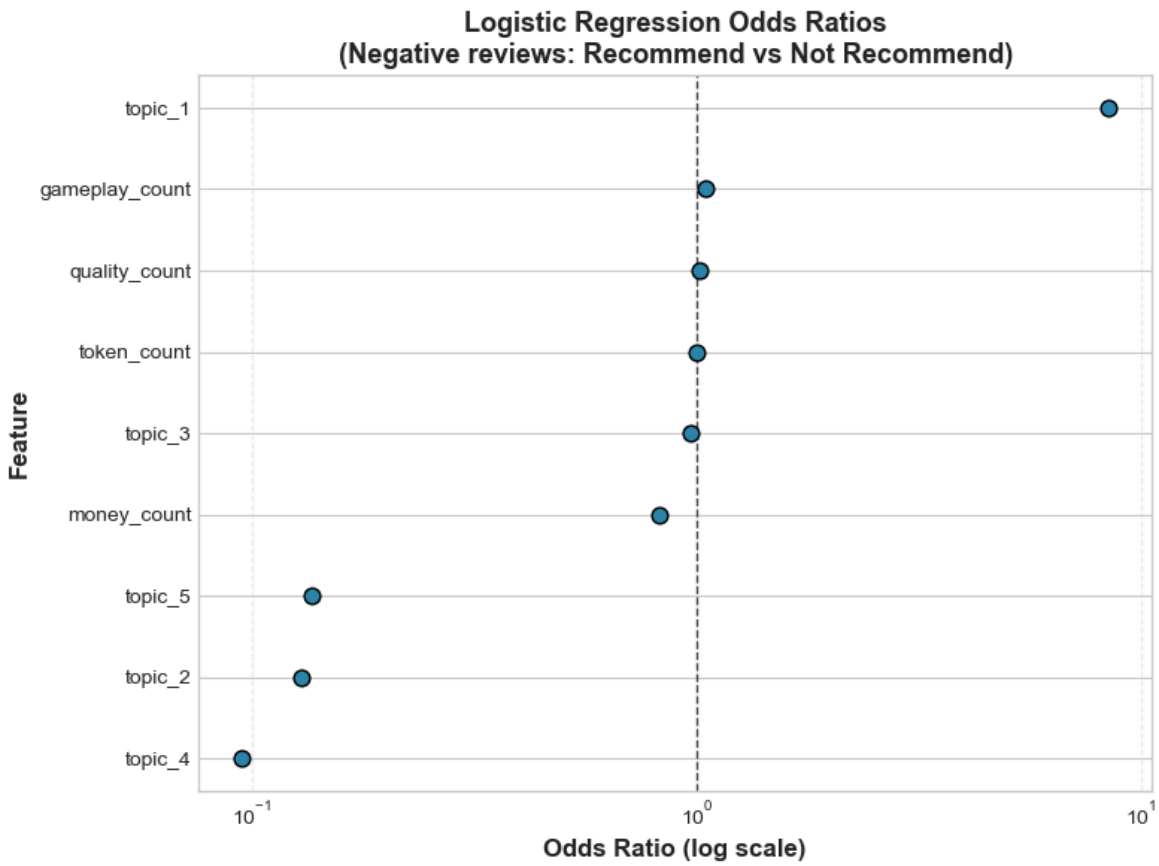
In [27]:
```
# logistic regression to find coefficients of each feature

lex_cols = ['quality_count', 'money_count', 'gameplay_count']
base_features = lex_cols + ['token_count']

topic_cols = [c for c in df.columns if c.startswith('topic_']
selected_topics = [t for t in top_topics_for_model if t in to

feature_cols = base_features + selected_topics
print("Features used in logistic regression:")
print(feature_cols)

model_df = df.dropna(subset=feature_cols + ['recommended']).c

X = model_df[feature_cols]
y = model_df['recommended']

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X, y)

coef_series = pd.Series(log_reg.coef_[0], index=feature_cols)
odds_ratios = np.exp(coef_series)
odds_ratios_df = pd.DataFrame({
```

```
    'coef': coef_series,
    'odds_ratio': odds_ratios
})

print("\nOdds ratios for lexicon predictors:")
print(odds_ratios_df.loc[lex_cols, :])

print("\nOdds ratios for selected topic predictors:")
print(odds_ratios_df.loc[selected_topics, :])
```

```
Features used in logistic regression:
['quality_count', 'money_count', 'gameplay_count', 'token_cou
nt', 'topic_5', 'topic_1', 'topic_4', 'topic_3', 'topic_2']

Odds ratios for lexicon predictors:
                     coef  odds_ratio
quality_count   0.011841    1.011911
money_count    -0.194068    0.823602
gameplay_count  0.048951    1.050169

Odds ratios for selected topic predictors:
              coef  odds_ratio
topic_5 -1.995350    0.135966
topic_1  2.130700    8.420762
topic_4 -2.351613    0.095216
topic_3 -0.028645    0.971762
topic_2 -2.049519    0.128797
```

In [28]:

```
or_df = odds_ratios_df.copy()
or_df = or_df.reset_index().rename(columns={'index': 'feature
or_df = or_df.sort_values('odds_ratio')

plt.figure(figsize=(8, 6))

plt.axvline(x=1.0, color='black', linestyle='--', linewidth=1

plt.scatter(or_df['odds_ratio'], or_df['feature'], s=60, colo

plt.xscale('log')
plt.xlabel('Odds Ratio (log scale)', fontsize=12, fontweight=
plt.ylabel('Feature', fontsize=12, fontweight='bold')
plt.title('Logistic Regression Odds Ratios\n(Negative reviews

plt.grid(axis='x', alpha=0.3, linestyle='--')
plt.tight_layout()
plt.savefig('plots/odds_ratios.png', dpi=300, bbox_inches='ti
plt.show()
```


Logistic Regression Odds Ratios
(Negative reviews: Recommend vs Not Recommend)
topic_1
gameplay_count
quality_count
token_count
topic_3
money_count
topic_5
topic_2
topic_4
Feature
10^{-1}
10^{0}
10^{1}
Odds Ratio (log scale)