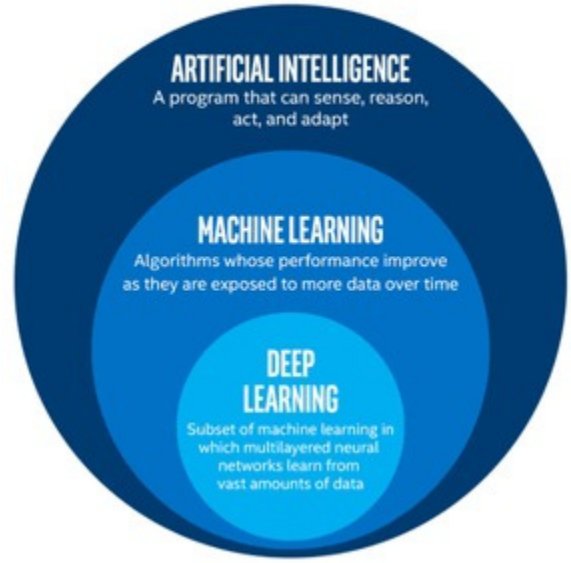
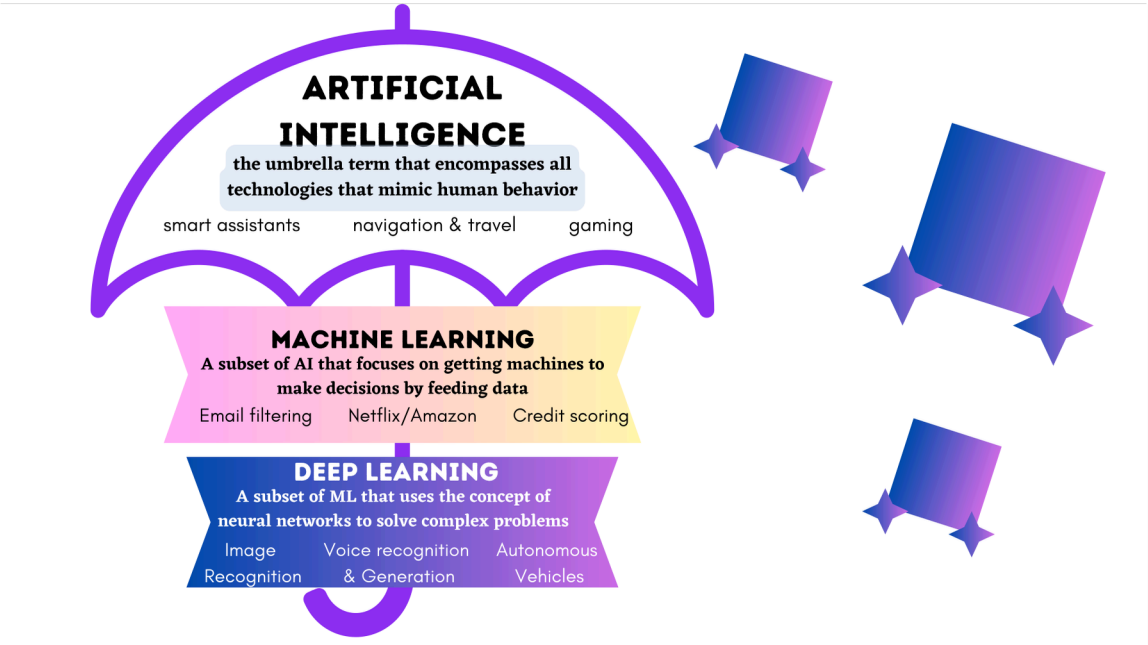
Machine Learning Recap Notes

1. Introduction

What is Machine Learning?





Machine Learning (ML) is a subset of Artificial Intelligence (AI) that enables computers to learn and make decisions from data without being explicitly programmed for every task.

So instead of writing specific instructions, we provide algorithms with examples and let them **find patterns**.

Think of ML like teaching a child to recognize animals. Instead of describing every detail of what makes a cat a cat, you show them hundreds of pictures of cats and dogs. Eventually, they learn to distinguish between them on their own.

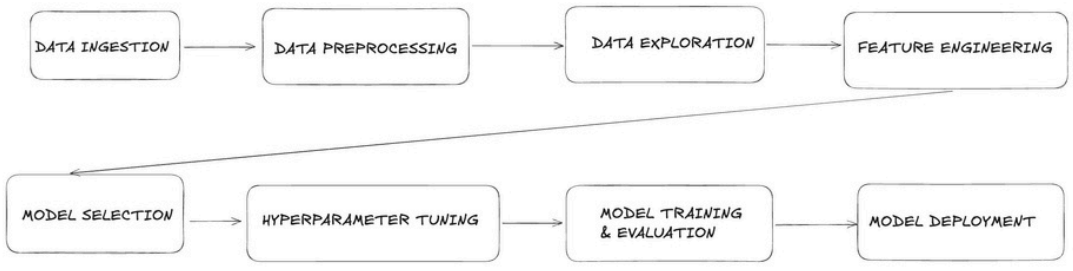
Why is Machine Learning Important?

- **Automation**: Automate complex decision-making processes
- **Pattern Recognition**: Find hidden patterns in large datasets
- **Prediction**: Make predictions about future events
- **Personalization**: Customize experiences (like Netflix recommendations)
- **Efficiency**: Process vast amounts of data quickly

2. Glossary

- **Features (X)**: Input variables (like height, weight, age)
- **Target/Label (y)**: What we want to predict (like price, category)
- **Data**: The information we feed to our algorithms
- **Dataset**: Collection of data
- **DataFrame**: 2D table-like data structure in pandas
- **Training Set**: Dataset used to teach the algorithm
- **Test Set**: Dataset used to evaluate performance
- **Algorithm**: The mathematical method used to find patterns
- **Model**: The result after training an algorithm on data

3. Machine Learning Pipeline



Step	Short Definition
1. Data Ingestion	Load data from files, APIs, or databases
2. Data Preprocessing	Clean and fix messy data
3. Data Exploration	Look at data to understand patterns
4. Feature Engineering	Create better input variables
5. Model Selection	Pick the best algorithm
6. Hyperparameter Tuning	Adjust settings for better performance
7. Model Training and Evaluation	Train model and test accuracy
8. Model Deployment	Put model into real use

1. **Data Ingestion**
   → The process of **gathering data** from various sources:

   - CSV files
   - SQL databases

APIs•
Real•time streams 2.
Data Preprocessing

→ **Cleaning the raw data** to make it usable, including:

- Handling missing values
- Removing duplicates
- Converting data types
- Normalizing values
- Fixing errors or inconsistencies

3. **Data Exploration**
   → Also known as **Exploratory Data Analysis (EDA)**. We analyze data to understand patterns, correlations, and potential issues, using:

   - Charts
   - Summaries
   - Visualizations

4. **Feature Engineering**
   → Creating or modifying **input variables (features)** to help the model learn better. This step transforms raw data into more meaningful representations. For example:

   - Converting `"date of birth"` into `"age"` to make it a usable number
   - Creating `"BMI"` from `"height"` and `"weight"`
   - Encoding categories (e.g., turning `"Male"` / `"Female"` into 0 and 1)

5. **Model Selection**
   → Choosing the **appropriate algorithm** based on the task (e.g., regression,

classification, clustering). The choice depends on:

- Data type
- Project goal
- Need for interpretability

6. **Hyperparameter Tuning**
   → Adjusting the model's settings (like tree depth, number of clusters, or learning rate) to improve performance.
   Good hyperparameters can greatly boost model accuracy and reduce overfitting.

   Common techniques include:

   - Grid Search: Tries all possible combinations of hyperparameter values
   - Random Search: Tries random combinations for faster results on large spaces

   These methods help find the best settings based on validation performance (e.g., accuracy or RMSE)

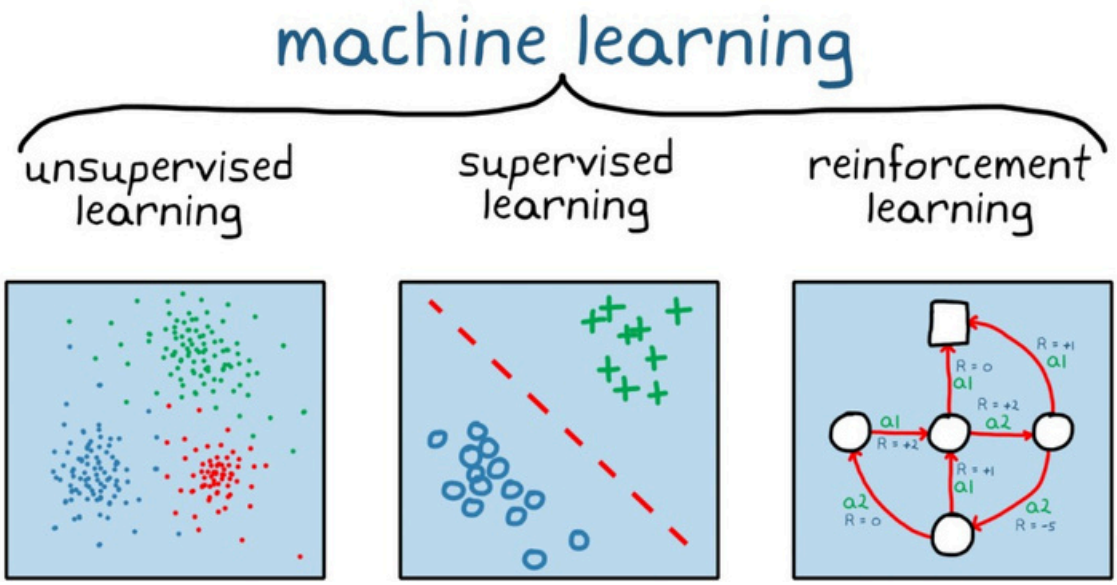
7. **Model Training and Evaluation**
   → **Training** the model on a training dataset and then **evaluating** it on a test dataset. We measure how well it performs using metrics like:

   - Accuracy
   - Precision & Recall
   - RMSE (Root Mean Square Error = how far your predictions are from the real values), etc.

8. **Model Deployment**
   → Putting the trained model into **real-world usage**, such as:

   - Integrating into a web or mobile app
   - Embedding into backend systems
   - Making real-time predictions on new, unseen data

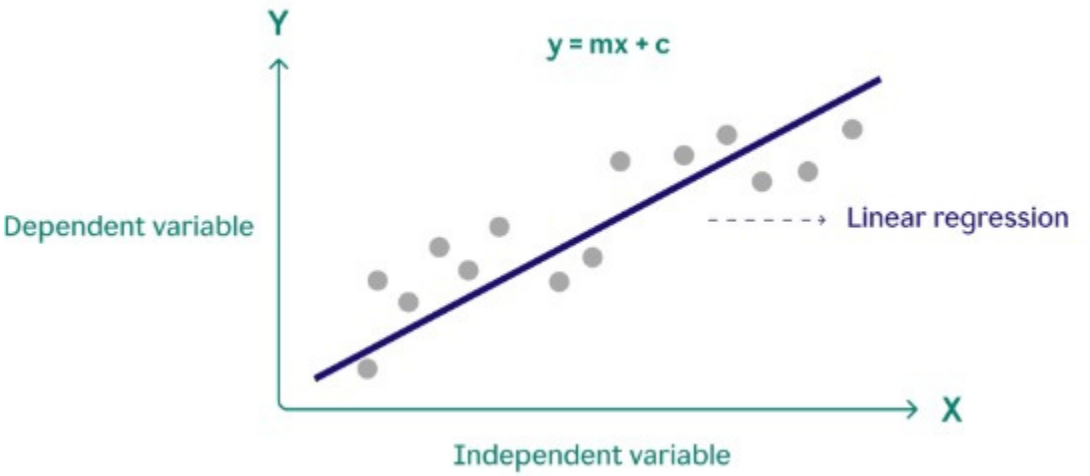
4. Types of Machine Learning



	Unsupervised Learning	Supervised Learning	Reinforcement Learning
Definition	Finding patterns in data without labeled examples	Learning with labeled examples (we know the correct answers)	Learning through trial and error with rewards/penalties
Examples	- Customer segmentation - Market basket analysis - Anomaly detection	- Email spam detection (spam/not spam) - House price prediction - Image recognition	- Game playing (chess, Go) - Robot navigation - Trading strategies
Common Algorithms	- K-Means Clustering - Hierarchical Clustering - Principal Component Analysis (PCA)	- Linear Regression - Decision Trees - Random Forest - Support Vector Machines	- Q-learning - Deep Q-Networks (DQN) - Policy Gradient Methods

5. Common Algorithms

1. Linear Regression

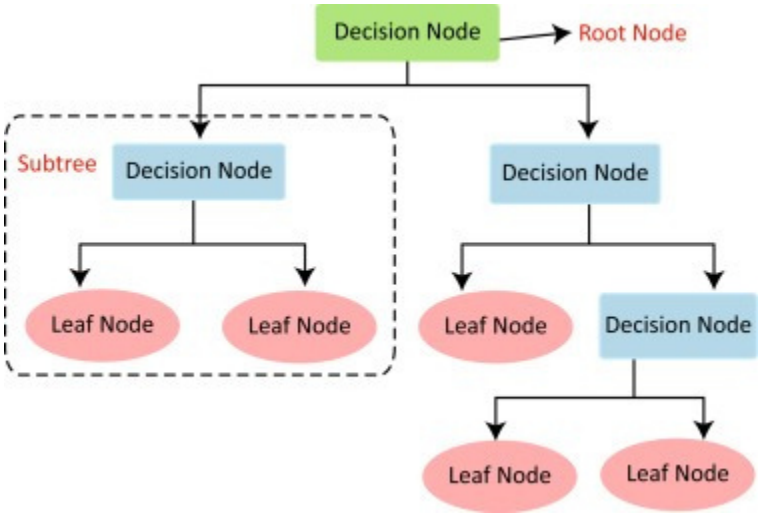


Definition: A model that predicts a continuous value based on the relationship between input features and the target.

Use Case: Predicting prices, trends (e.g., house price prediction).

Strengths	Weaknesses
Simple and easy to interpret	Struggles with non-linear patterns
Works well when the relationship is linear	Sensitive to outliers

2. Decision Trees

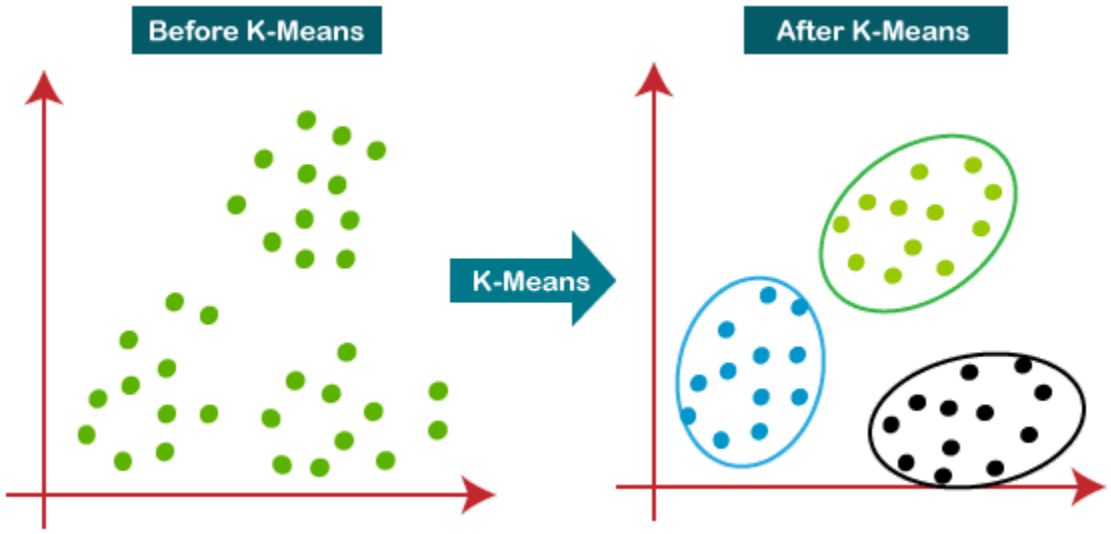


Definition: A tree-based model that splits the data based on feature values to make decisions.

Use Case: Classification problems like determining whether an email is spam or not.

Strengths	Weaknesses
Easy to understand and visualize	Prone to overfitting
Can handle both numerical and categorical data	Small changes in data can lead to a completely different tree
No need for feature scaling	

3. K-Means Clustering



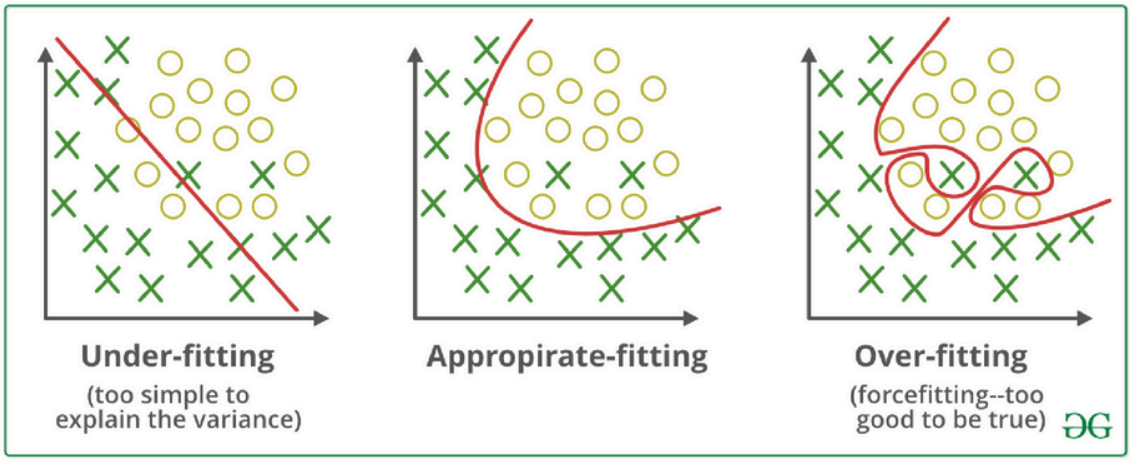
Definition: An unsupervised algorithm that groups data into K clusters based on similarity.

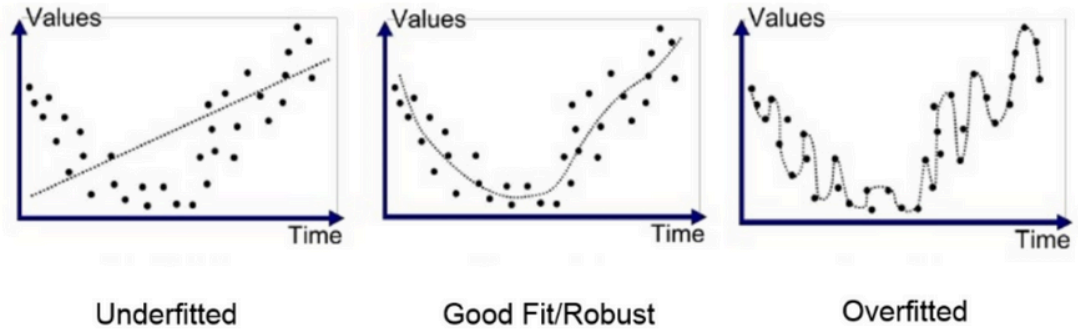
Use Case: Customer segmentation, grouping similar users or products.

Strengths	Weaknesses
Fast and efficient for large datasets	Requires choosing K (number of clusters) manually
Easy to understand and implement	Doesn’t work well with non-spherical clusters or different cluster sizes
	Sensitive to outliers

6. Additionals

1. Overfitting vs Underfitting





	Underfitting (Too Simple)	Overfitting (Too Complex)	Good Fit (Just Right)
What it is	Model is too basic to capture real patterns	Model memorizes training data instead of learning general patterns	Model captures the true underlying patterns
Example	Using a straight line to predict house prices when the relationship is curved	A student who memorizes answers but can't solve new problems	A student who understands the concepts and can apply them in new situations
Signs	Poor performance on both training and test data	Great performance on training data, terrible on new data	Good performance on both training and test data
Symptoms	High bias, low variance	Low bias, high variance	Balanced bias and variance
Solution	Use more complex models or add more features	Use simpler models, get more data, or apply regularization	Maintain a good balance between model complexity and data quality

How to Detect:

```
# Training accuracy: 95%
# Test accuracy: 96%          ← Good fit!

# Training accuracy: 60%
```

```
# Test accuracy: 58%          ← Underfitting

# Training accuracy: 99%
# Test accuracy: 65%          ← Overfitting
```

2. Bias-Variance Tradeoff

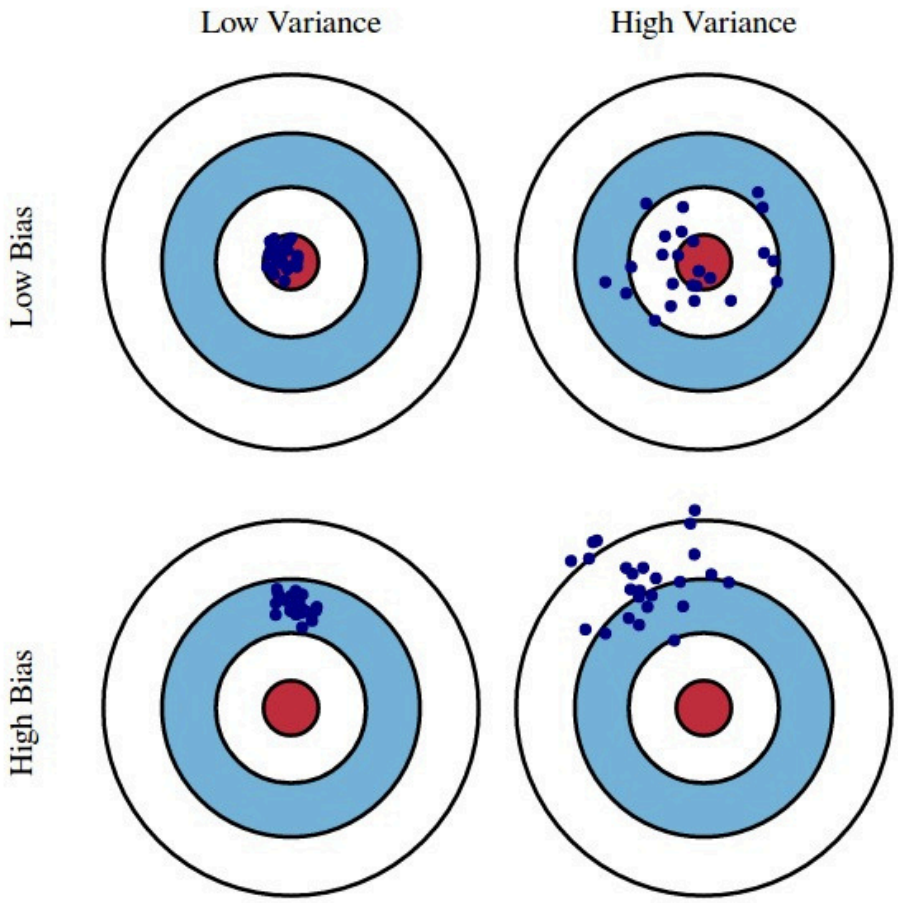


Fig. 1 Graphical illustration of bias and variance.

- **Bias**: Error due to overly simple assumptions (underfitting)
- **Variance**: Error due to too much complexity (overfitting)
- **Goal**: Find a balance where both are low

High Bias → Model is too simple → Underfitting
High Variance → Model is too complex → Overfitting
Just Right → Low bias and low variance → Good generalization

3. Tips to Increase Model Accuracy

- Clean your data properly
- Feature engineering (create meaningful input variables)
- Try different models and compare
- Use cross-validation
- Tune hyperparameters (e.g., learning rate, depth, number of trees)
- Add more high-quality data
- Use ensemble methods (e.g., Random Forest, XGBoost)
- Regularize your models (e.g., L1/L2 regularization)

7. Useful References

1. Online Resources

- **FreeCodeCamp**

  Website: freecodecamp.org

  YouTube Channel: FreeCodeCamp

  Popular Videos:

  - Intro to Machine Learning
  - Machine Learning Crash Course

- **DataCamp**

  Website: datacamp.com

2. Dataset

- **Kaggle**: kaggle.com/datasets (huge collection of real datasets)
- **UCI ML Repository**: archive.ics.uci.edu/ml (classic ML datasets)
- **Google Dataset Search**: datasetsearch.research.google.com
- **Government Data**: data.gov (US), data.gov.sg (Singapore)
- **Awesome Public Datasets**: github.com/awesomedata/awesome-public-datasets

🎯 Let's Code: YouTube Trending Video Analysis

In this notebook, we'll walk through a complete mini data science project using **real-world YouTube trending data**.

What We'll Cover:

1. **Data Ingestion** – Load trending video data from CSV and SQL database
2. **Data Preprocessing** – Remove invalid, zero, or error-filled entries
3. **Data Exploration** – Visualize distributions and trends in views, dates, and features
4. **Feature Engineering** – Create new variables like ratios, title length, and publish timing
5. **Model Selection** – Compare classification models (Random Forest, Gradient Boosting, AdaBoost)
6. **Hyperparameter Tuning** – Use GridSearchCV to find the best model settings
7. **Model Training & Evaluation** – Train models and evaluate performance using accuracy
8. **Model Deployment** – Launch a Gradio dashboard to showcase model results and visual insights

We'll also use:

- **SQLite** for data handling
- **Pandas/Matplotlib/Seaborn** for analysis
- **Scikit-learn** for machine learning
- **Gradio** to create a simple interactive interface

Pipeline 1 - Data Ingestion

Step 1: Install Dependencies

Install specific versions of essential libraries for data handling, visualization, modeling, and web UI (Gradio).

In [ ]:
```
# Cell 1: Install specific library versions
# ----------------------------------------------------------

!pip install pandas==1.5.3 \
                    numpy==1.24.2 \
```

```
sqlalchemy==2.0.8 \
scikit-learn==1.2.2 \
matplotlib==3.7.1 \
seaborn==0.12.2 \
gradio==4.44.1
```

Step 2: Import Libraries & Set Plot Style

Import core libraries for data manipulation (pandas, numpy), SQL handling (sqlite3, sqlalchemy), machine learning (sklearn), plotting (matplotlib, seaborn), and dashboarding (gradio). Also sets a clean and readable default plotting style using Matplotlib.

In [ ]:

```
# Cell 2: Imports & styling
# ------------------------------------------------------------

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sqlite3
from sqlalchemy import create_engine

from sklearn.model_selection import train_test_split, GridSearchCV from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline from sklearn.linear_model import LogisticRegression from sklearn.naive_bayes import GaussianNB, BernoulliNB from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis \ as QDA from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier from sklearn.metrics import classification_report, roc_auc_score, roc_curve from sklearn.metrics import accuracy_score



import gradio as gr
import seaborn as sns

# Use default matplotlib style for simplicity
plt.style.use('default')
```

Step 3: Load Dataset & Initial Preview

Load the raw YouTube dataset (USvideos.csv) and display its dimensions and a sample of the first few rows to get an initial sense of the data.

In [ ]:

```
# Cell 3: Data Ingestion & preview
# ------------------------------------------------------------

raw_df = pd.read_csv('USvideos.csv')
```

```
print("Raw data shape:", raw_df.shape)
raw_df.head(1).T
```

Raw data shape: (40949, 16)

Out[ ]:

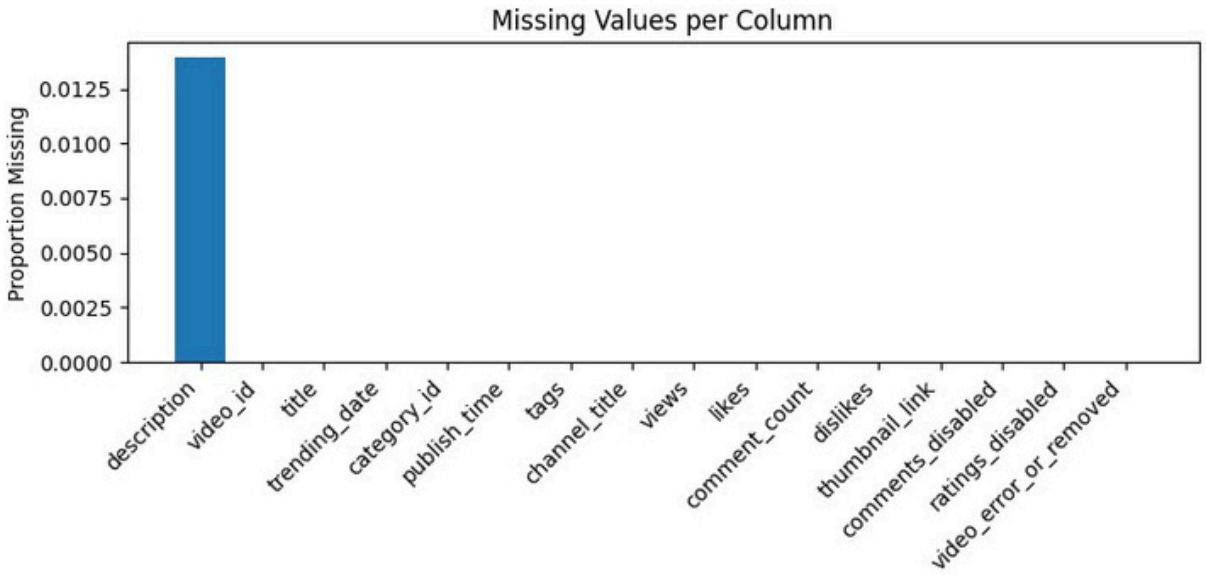
	0
video_id	2kyS6SvSYSE
trending_date	17.14.11
title	WE WANT TO TALK ABOUT OUR MARRIAGE
channel_title	CaseyNeistat
category_id	22
publish_time	2017-11-13T17:13:01.000Z
tags	SHANtell martin
views	748374
likes	57527
dislikes	2966
comment_count	15954
thumbnail_link	https://i.ytimg.com/vi/2kyS6SvSYSE/default.jpg
comments_disabled	False
ratings_disabled	False
video_error_or_removed	False
description	SHANTELL'S CHANNEL - https://www.youtube.com/s...

Step 4: Visualize Missing Data Proportions

Calculate and visualize the proportion of missing values per column to understand data quality and decide if any preprocessing or filtering is needed.

In [ ]:

```
# Cell 4: Missing-value proportions (simple bar chart)
# ------------------------------------------------------------

missing = raw_df.isnull().mean().sort_values(ascending=False)
plt.figure(figsize=(8,4))
plt.bar(missing.index, missing.values)
plt.xticks(rotation=45, ha='right')
plt.ylabel("Proportion Missing")
plt.title("Missing Values per Column")
plt.tight_layout()
plt.show()
```



Step 5: Save Raw Data to SQLite Database

Persist the raw DataFrame into a local SQLite database (youtube_trending.db) for SQL-based querying and transformation. Then confirm the data was written by counting the rows.

In [ ]:
```
# Cell 5: Write raw data to SQLite & verify
# ------------------------------------------------------------

engine = create_engine('sqlite:///youtube_trending.db')
raw_df.to_sql('raw_data', con=engine, if_exists='replace', index=False)
conn = sqlite3.connect('youtube_trending.db')

# use SQL script to find total rows in raw data
cnt = pd.read_sql_query(
"""
SELECT COUNT(*) AS cnt
FROM raw_data
""",
conn
).iloc[0, 0]
print("Rows in raw_data table:", cnt)
```

```
Rows in raw_data table: 40949
```

Pipeline 2 - Data Preprocessing

Step 6: Clean Data with SQL Filtering

Create a cleaned version of the dataset by filtering out videos that:

- Have 0 views

- Were removed or had errors
- Had comments disabled

Then compare row counts before and after cleaning to verify the impact.

In [ ]:
```
# Cell 6: Data cleaning via SQL
# ------------------------------------------------------------

c = conn.cursor()
c.execute("DROP TABLE IF EXISTS cleaned_data")

# 1) Create cleaned_data table with data from raw_data that have views > 0,
# no errors, and comments enabled.
c.execute("""
      CREATE TABLE cleaned_data AS
      SELECT *
         FROM raw_data
        WHERE views > 0
          AND video_error_or_removed = 0
          AND comments_disabled = 0
""")
conn.commit()

# 2) Compare total rows of data before and after cleaning
before = pd.read_sql_query(
"""
SELECT COUNT(*) AS cnt
FROM raw_data
""",
conn
).iloc[0, 0]
after = pd.read_sql_query(
"""
SELECT COUNT(*) AS cnt
FROM cleaned_data
""",
conn
).iloc[0, 0]
print(f"Before cleaning: {before} rows\nAfter cleaning:                {after} rows")
```

```
Before cleaning: 40949 rows
After cleaning: 40293 rows
```

In [ ]:
```
# 3) Quick overview of cleaned_data (YouTube version)
cleaned_df = pd.read_sql_query("""
      SELECT views, likes, dislikes, comment_count
         FROM cleaned_data
        LIMIT 100
""", conn)
print("Sample of cleaned_data:")
display(cleaned_df)

# 4) Numeric summary of key columns
stats = pd.read_sql_query("""
      SELECT
```

```
        AVG(views)     AS avg_views,
        AVG(likes)     AS avg_likes,
        AVG(dislikes)  AS avg_dislikes,
        AVG(comment_count) AS avg_comments,

        MIN(views)     AS min_views,
        MAX(views)     AS max_views
    FROM cleaned_data
""", conn)
pd.set_option("display.float_format", "{:.2f}".format)
print("Numeric summary of cleaned_data:")
display(stats.T)
```

Sample of cleaned_data:

	views	likes	dislikes	comment_count
0	748374	57527	2966	15954
1	2418783	97185	6146	12703
2	3191434	146033	5339	8181
3	343168	10172	666	2146
4	2095731	132235	1989	17518
...	...	...	...	...
95	836544	40195	373	976
96	284666	16396	81	949
97	3371669	202676	3394	20086
98	195685	14338	171	1070
99	1098897	43875	1326	4702

100 rows × 4 columns

Numeric summary of cleaned_data:

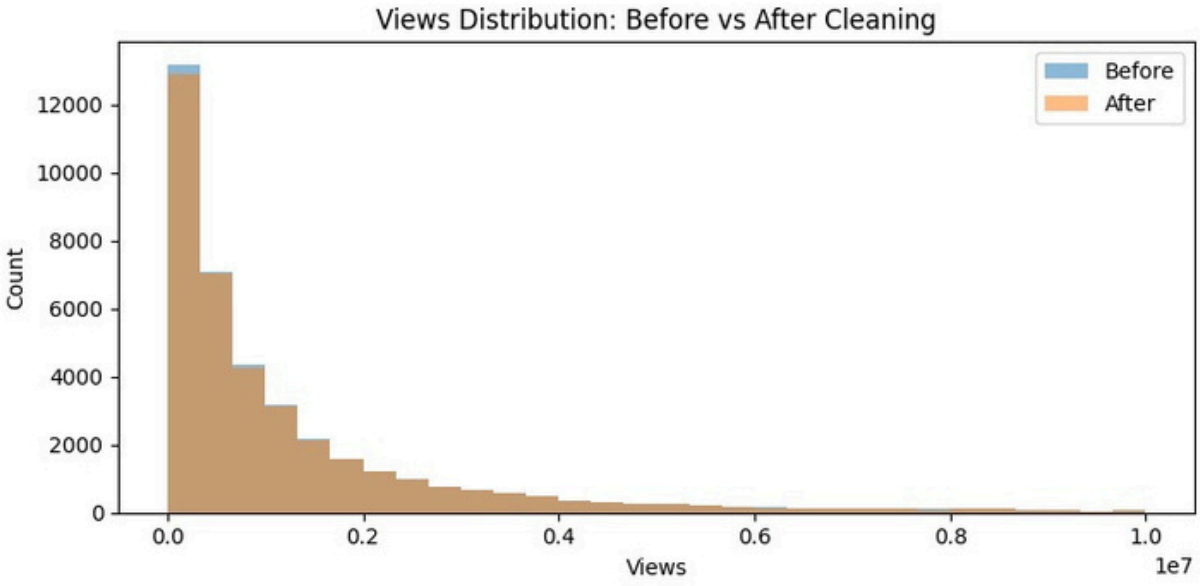
	0
avg_views	2358704.80
avg_likes	75111.08
avg_dislikes	3728.82
avg_comments	8582.86
min_views	549.00
max_views	225211923.00

Step 7: Compare Views Distribution Before & After Cleaning

Plot histograms of video view counts before and after data cleaning. This helps assess how the filtering affected the overall distribution and whether extreme or invalid values were removed.

In [8]:

```
# Cell 7: Views distribution before vs after cleaning (simple histograms)
# ------------------------------------------------------------

# 1) Load the 'views' column from the raw and cleaned datasets
before_v = pd.read_sql_query("SELECT views FROM raw_data", conn)['views']
after_v = pd.read_sql_query("SELECT views FROM cleaned_data", conn)['views']

# 2) Plot histograms to compare the distribution of views before and after
# cleaning
plt.figure(figsize=(8,4))
plt.hist(before_v, bins=30, alpha=0.5, label='Before', range=(0, 1e7))
plt.hist(after_v, bins=30, alpha=0.5, label='After', range=(0, 1e7))
plt.xlabel("Views") plt.ylabel("Count") plt.title("Views
Distribution: Before vs After Cleaning") plt.legend() plt.tight_layout()
plt.show()
```



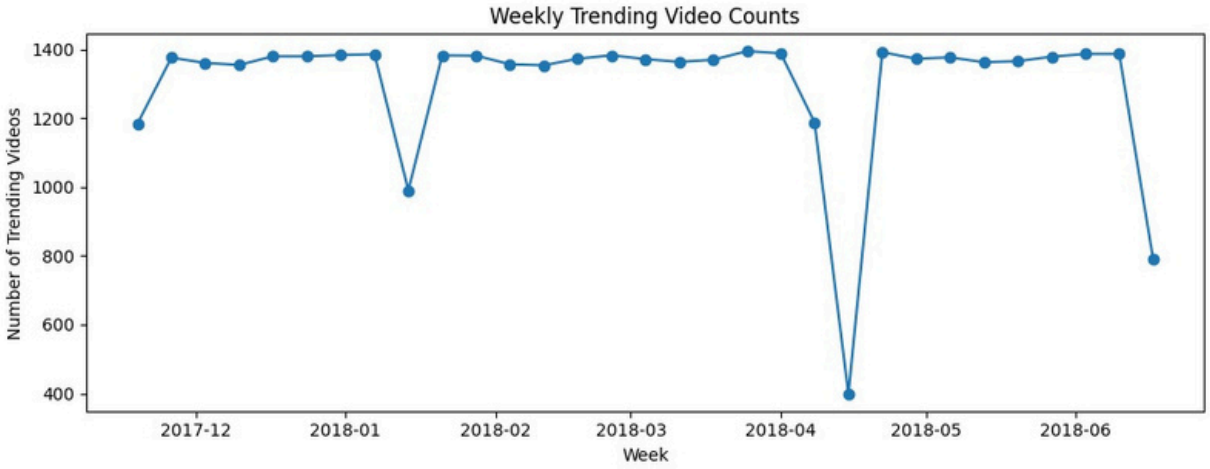
Pipeline 3 - Data Exploration

Step 8: Weekly Trending Video Counts

Use SQL and pandas to parse and convert trending dates. Aggregate video counts per week and plot the number of trending videos over time to uncover seasonality or platform behavior trends.

In [9]:

```
# Cell 8: Weekly trending video counts (fixed date parsing, simple line plot
# ------------------------------------------------------------

# 1) Query daily trending video counts with corrected date format
monthly = pd.read_sql_query("""
      SELECT
         date(
            '20' || substr(trending_date,1,2) || '-' ||
            substr(trending_date,7,2)                  || '-' ||
            substr(trending_date,4,2)
         ) AS TrendDate,
         COUNT(*) AS Count
      FROM    cleaned_data
      GROUP BY TrendDate
      ORDER BY TrendDate
""", conn)

# 2) Ensure TrendDate is a proper datetime type
monthly['TrendDate'] = pd.to_datetime(monthly['TrendDate'],
                                      format='%Y-%m-%d')

# 3) Resample counts by week, summing daily counts
weekly_counts = monthly.set_index('TrendDate')['Count'] \
                       .resample('W') \
                       .sum()

# 4) Plot weekly trending video counts with a simple line plot
plt.figure(figsize=(10,4))
plt.plot(weekly_counts.index, weekly_counts.values, marker='o')
plt.xlabel("Week")
plt.ylabel("Number of Trending Videos")
plt.title("Weekly Trending Video Counts")
plt.tight_layout()
plt.show()

# 5) Display the monthly daily counts dataframe
display(monthly)
```



	TrendDate	Count
0	2017-11-14	198
1	2017-11-15	198
2	2017-11-16	197
3	2017-11-17	197
4	2017-11-18	197
...	...	...
200	2018-06-10	198
201	2018-06-11	198
202	2018-06-12	198
203	2018-06-13	197
204	2018-06-14	198

205 rows × 2 columns

Pipeline 4 - Feature Engineering

Step 9: Engagement & Popularity

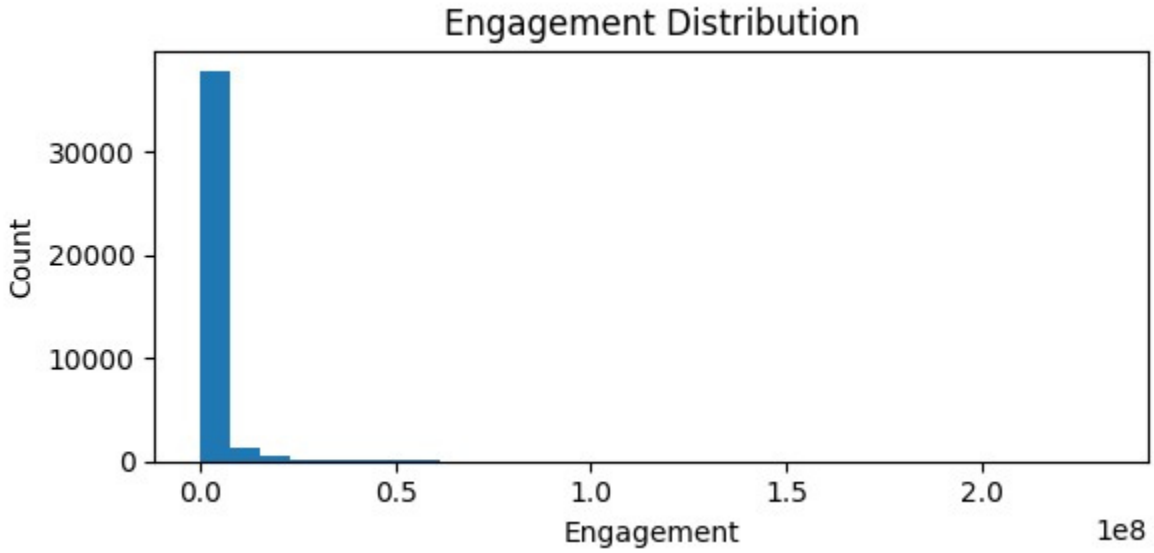
Add basic new features:

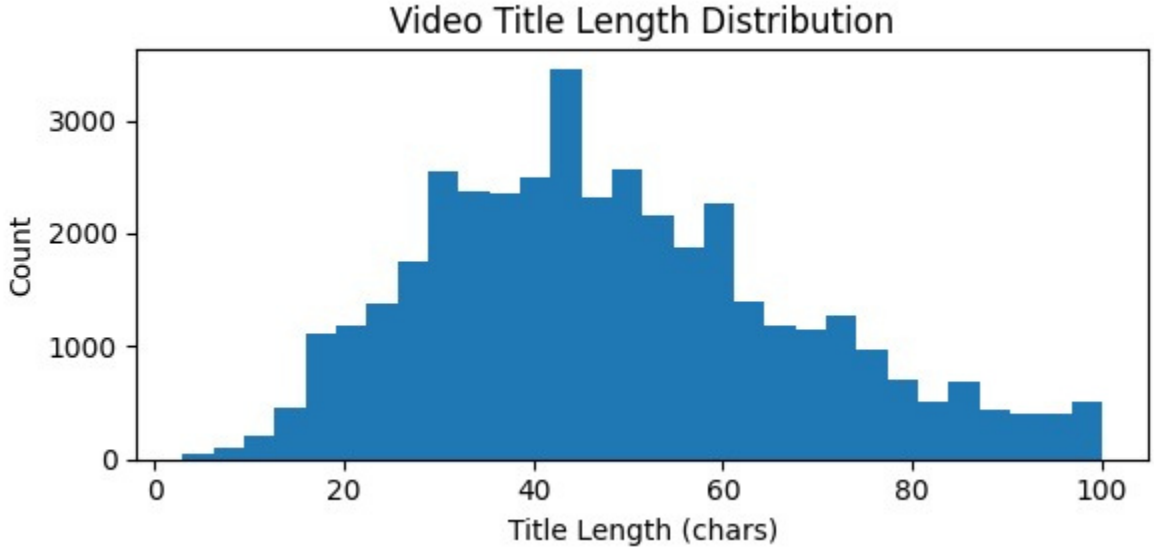
- **Engagement**: Combined count of views, likes, and comments
- **TitleLength**: Character count of the video title
- **is_popular**: A binary label marking videos with above-median views as "popular "

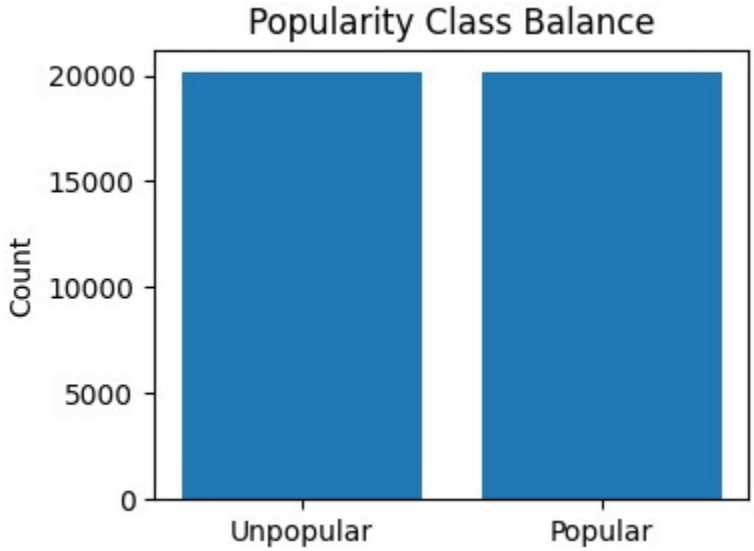
Also plot distributions of these new features and visualize the class balance of popularity.

In [10]:
```
# Cell 9: Feature Engineering – Basic numeric & title length # ----------
-----------------------------------------

# 1) Compute median views from cleaned_data
median_views = pd.read_sql_query(
      "SELECT views FROM cleaned_data",
      conn
)["views"].median()

# 2) Calculate and create new features: total engagement (sum of views +
# likes + comments), title length, and popularity flag
c.execute("DROP TABLE IF EXISTS feat_basic")
c.execute(f"""
```

```
    CREATE TABLE feat_basic AS
    SELECT
    *,
    (views + likes + comment_count) AS Engagement,
        LENGTH(title)                           AS TitleLength,
        CASE
            WHEN views > {median_views:.0f} THEN 1
            ELSE 0
        END AS is_popular
    FROM cleaned_data
""")
conn.commit()

# 3) Load new features from database
fb = pd.read_sql_query(
    "SELECT Engagement, TitleLength, is_popular FROM feat_basic",
    conn)

# 4) Plot Engagement distribution
plt.figure(figsize=(6,3))
plt.hist(fb['Engagement'], bins=30)
plt.title("Engagement Distribution")
plt.xlabel("Engagement")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 5) Plot Title Length distribution
plt.figure(figsize=(6,3))
plt.hist(fb['TitleLength'], bins=30)
plt.title("Video Title Length Distribution")
plt.xlabel("Title Length (chars)")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 6) Plot Popular vs Unpopular count
plt.figure(figsize=(4,3))
counts = fb['is_popular'].value_counts().sort_index()
plt.bar(['Unpopular','Popular'], counts.values)
plt.title("Popularity Class Balance")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 7) Display the feature table
display(fb)
```


Engagement Distribution
Count
30000
20000
10000
0
0.0
0.5
1.0
1.5
2.0
Engagement
1e8


Video Title Length Distribution
Count
3000
2000
1000
0
0
20
40
60
80
100
Title Length (chars)


Popularity Class Balance
Count
20000
15000
10000
5000
0
Unpopular
Popular

	Engagement	TitleLength	is_popular
0	821855	34	1
1	2528671	62	1
2	3345648	53	1
3	355486	32	0
4	2245484	24	1
...	...	...	...
40288	1726426	28	1
40289	1128742	26	1
40290	1118511	84	1
40291	5866858	35	1
40292	10807993	64	1

40293 rows × 3 columns

Step 10: Ratio-Based Metrics

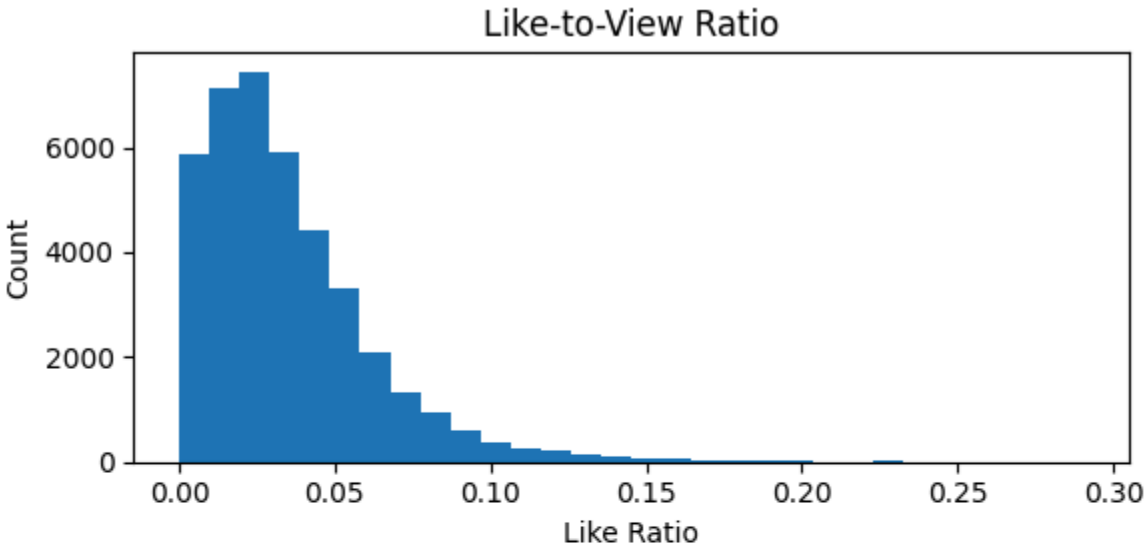
Add two derived metrics:

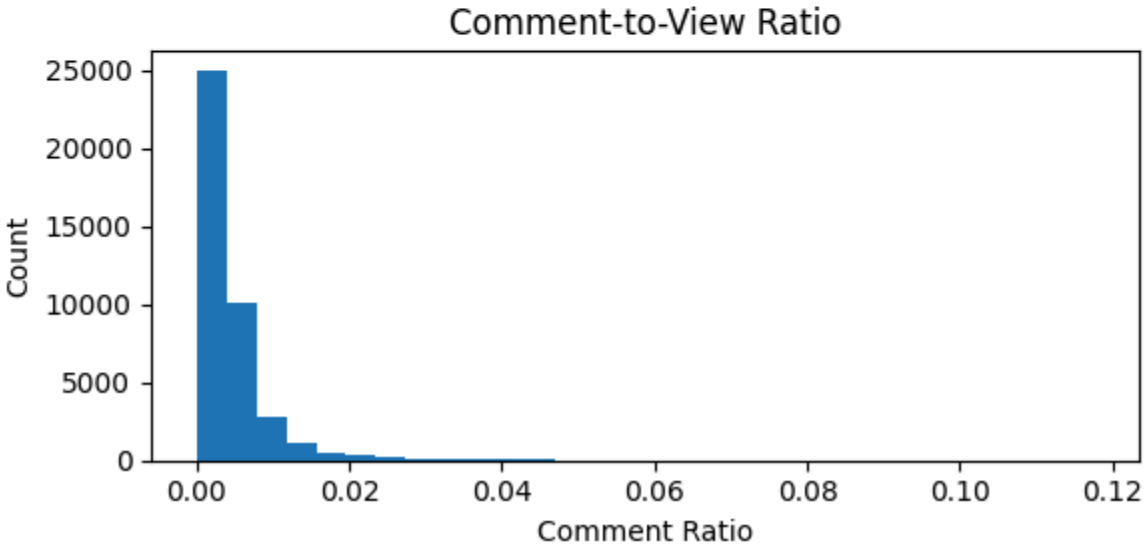
- **like_ratio**: Likes per view
- **comment_ratio**: Comments per view

These ratios normalize engagement by view count. Histograms show their distributions, giving insight into relative engagement.

In [11]:
```
# Cell 10: Feature Engineering – Derived ratios # ----------
----------------------------------

# 1) Create ratio features: like-to-view and comment-to-view ratios from
# basic features
c.execute("DROP TABLE IF EXISTS feat_ratios")
c.execute("""
      CREATE TABLE feat_ratios AS
      SELECT *,
                CAST(likes AS FLOAT)/views                  AS like_ratio,
                CAST(comment_count AS FLOAT)/views AS comment_ratio
        FROM feat_basic
""")
conn.commit()

# 2) Load the ratio features from the database
fr = pd.read_sql_query(
      "SELECT like_ratio, comment_ratio FROM feat_ratios",
      conn)
```

```
# 3) Plot Like-to-View Ratio distribution as histogram
plt.figure(figsize=(6,3))
plt.hist(fr['like_ratio'], bins=30)
plt.title("Like-to-View Ratio")
plt.xlabel("Like Ratio")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 4) Plot Comment-to-View Ratio distribution as histogram
plt.figure(figsize=(6,3))
plt.hist(fr['comment_ratio'], bins=30)
plt.title("Comment-to-View Ratio")
plt.xlabel("Comment Ratio")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 5) Display the ratio features table
display(fr)
```



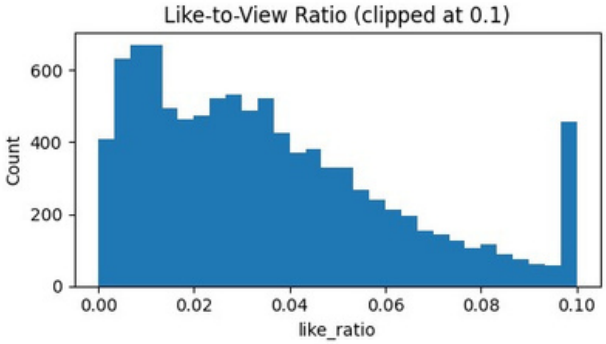


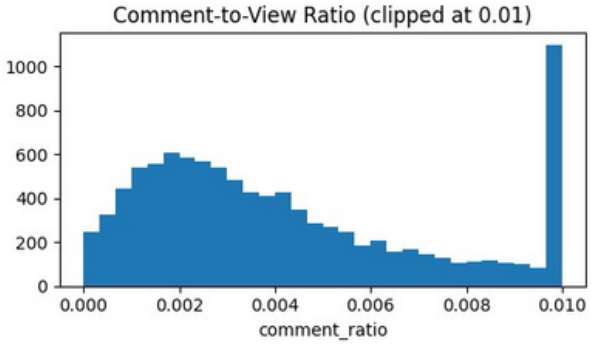
	like_ratio	comment_ratio
0	0.08	0.02
1	0.04	0.01
2	0.05	0.00
3	0.03	0.01
4	0.06	0.01
...	...	...
40288	0.02	0.00
40289	0.06	0.00
40290	0.05	0.00
40291	0.03	0.00
40292	0.03	0.01

40293 rows × 2 columns

In [12]:

```
# New Cell: Histogram of the new ratio features
# Load sample (10000 rows) of ratio features for analysis and visualization
ratios_df = pd.read_sql_query("""
      SELECT like_ratio, comment_ratio
         FROM feat_ratios
      LIMIT 10000
""", conn)

# Plot histogram of like-to-view ratio (values clipped at 0.1 for better
# visualization)
plt.figure(figsize=(10,3))
plt.subplot(1,2,1)
plt.hist(ratios_df['like_ratio'].clip(upper=0.1), bins=30)
plt.title("Like-to-View Ratio (clipped at 0.1)")
plt.xlabel("like_ratio")
plt.ylabel("Count")

# Plot histogram of comment-to-view ratio (values clipped at 0.01 for
# better visualization)
plt.subplot(1,2,2)
plt.hist(ratios_df['comment_ratio'].clip(upper=0.01), bins=30)
plt.title("Comment-to-View Ratio (clipped at 0.01)")
plt.xlabel("comment_ratio")
plt.tight_layout()
plt.show()

# Show the SQL table so far
display(ratios_df)
```





	like_ratio	comment_ratio
0	0.08	0.02
1	0.04	0.01
2	0.05	0.00
3	0.03	0.01
4	0.06	0.01
...	...	...
9995	0.01	0.00
9996	0.03	0.00
9997	0.01	0.00
9998	0.01	0.00
9999	0.01	0.00

10000 rows × 2 columns

Step 11: Temporal Attributes

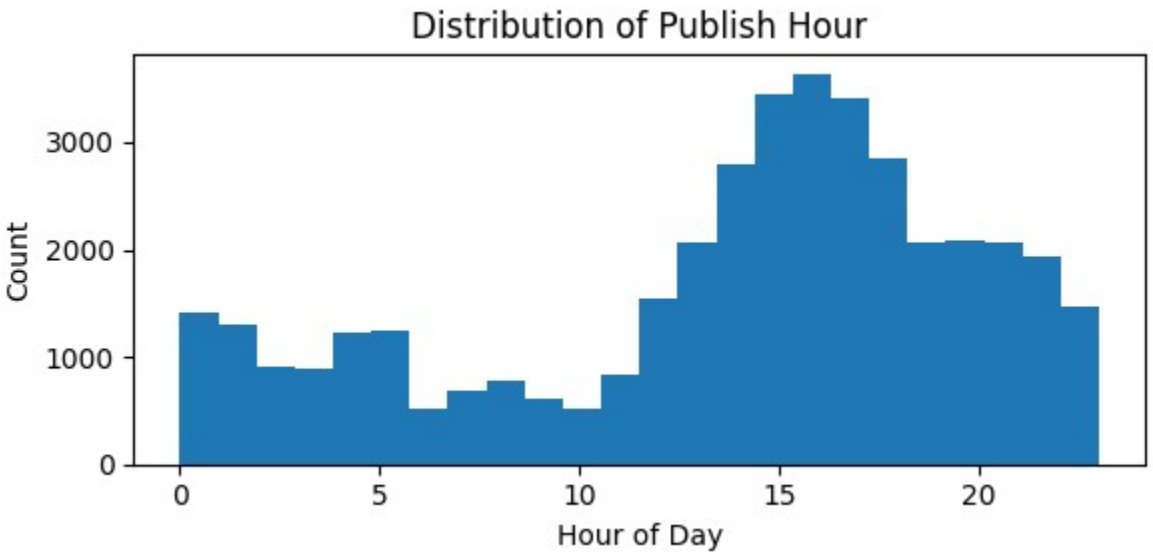
Extract:

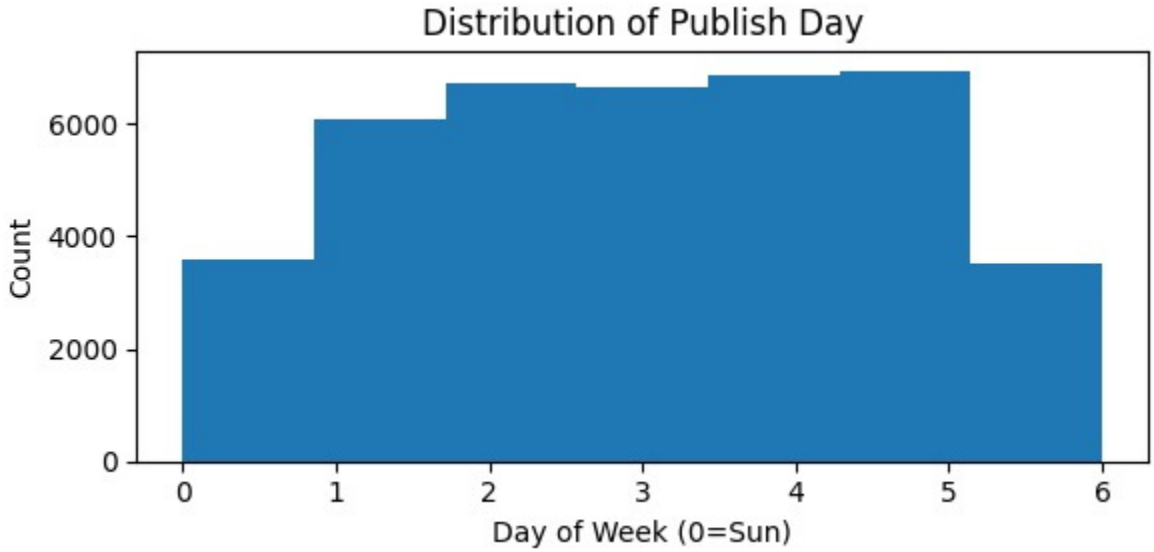
- **publish_hour**: Hour of video publishing
- **publish_dayofweek**: Day of the week (0 = Sunday)

Plot these features to understand when videos are commonly uploaded.

In [13]:
```
# Cell 11: Feature Engineering – Temporal features # ---------
-------------------------------------

# 1) Extract publish hour and publish day of week from publish_time,
# creating a new table with these temporal features
c.execute("DROP TABLE IF EXISTS feat_temp")
c.execute("""
      CREATE TABLE feat_temp AS
      SELECT
            f.*,
```

```
            CAST(strftime('%H', publish_time) AS INTEGER) AS publish_hour,
            CAST(strftime('%w', publish_time) AS INTEGER) AS publish_dayofweek
        FROM feat_ratios f
""")
conn.commit()

# 2) Load the new temporal features from the database
ft = pd.read_sql_query(
        "SELECT publish_hour, publish_dayofweek FROM feat_temp",
        conn)

# 3) Plot Publish Hour distribution as histogram
plt.figure(figsize=(6,3))
plt.hist(ft['publish_hour'], bins=24, range=(0,23))
plt.title("Distribution of Publish Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 4) Plot Publish Day of Week distribution as histogram
plt.figure(figsize=(6,3))
plt.hist(ft['publish_dayofweek'], bins=7, range=(0,6))
plt.title("Distribution of Publish Day")
plt.xlabel("Day of Week (0=Sun)")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 5) Display the extracted temporal features table
display(ft)
```





	publish_hour	publish_dayofweek
0	17	1
1	7	1
2	19	0
3	11	1
4	18	0
...	...	...
40288	13	5
40289	1	5
40290	17	5
40291	17	4
40292	17	4

40293 rows × 2 columns

Step 12: Channel-Level Aggregates

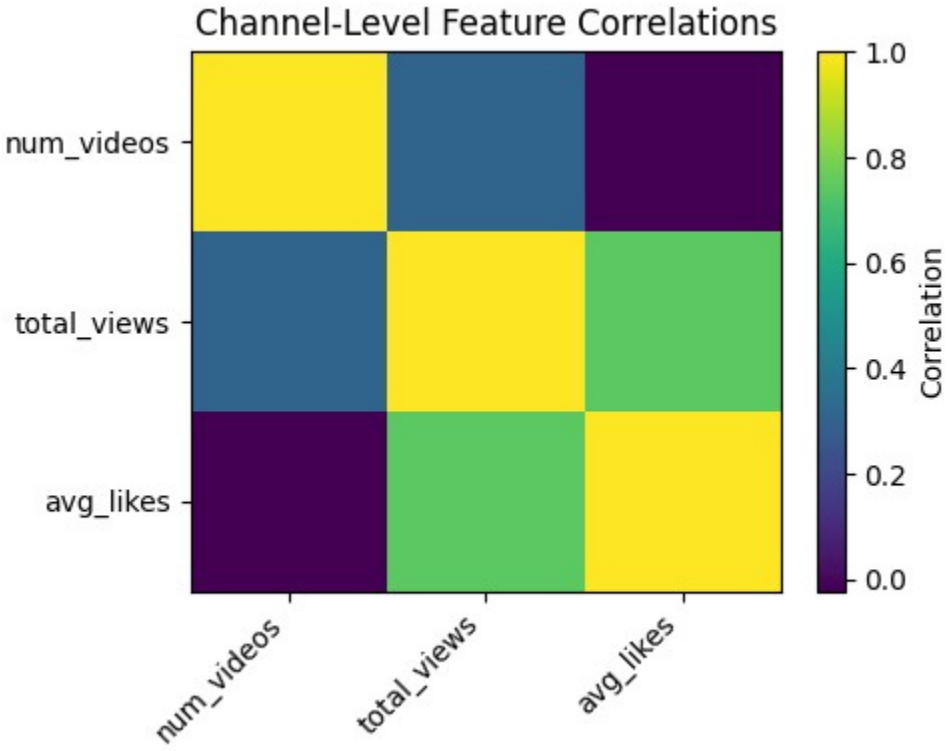
Aggregate features at the channel level:

- Number of videos
- Total views
- Average likes

Then merge these channel-level metrics back into the main dataset. Finally, show correlations between all numeric features using a heatmap to check for redundancy or predictive power.

In [14]:

```
# Cell 12: Feature Engineering – Channel aggregates & correlation # ------------
------------------------------------------------

# 1) Aggregate channel-level stats: count of videos, total views, and
# average likes per channel
c.execute("DROP TABLE IF EXISTS channel_agg")
c.execute("""
      CREATE TABLE channel_agg AS
      SELECT channel_title,
                COUNT(*)       AS num_videos,
                SUM(views) AS total_views,
                AVG(likes) AS avg_likes
         FROM feat_temp
 GROUP BY channel_title
""")
conn.commit()

# 2) Combine channel-level aggregates with detailed features by joining on
# channel_title
c.execute("DROP TABLE IF EXISTS features_final")
c.execute("""
      CREATE TABLE features_final AS
      SELECT t.*, c.num_videos, c.total_views, c.avg_likes
         FROM feat_temp t
 LEFT JOIN channel_agg c USING(channel_title)
""")
conn.commit()

# 3) Load selected features from final table and compute their correlation
# matrix
ff = pd.read_sql_query(
      "SELECT num_videos, total_views, avg_likes FROM features_final",
      conn)
corr = ff.corr()

# 4) Plot Correlation Matrix with Heatmap
plt.figure(figsize=(5,4))
plt.imshow(corr, cmap='viridis', aspect='auto')
plt.colorbar(label='Correlation')
plt.xticks(range(len(corr)), corr.columns, rotation=45, ha='right')
plt.yticks(range(len(corr)), corr.columns)
plt.title("Channel-Level Feature Correlations")
plt.tight_layout()
plt.show()

# 5) Show the SQL table so far
display(ff)
```



	num_videos	total_views	avg_likes
0	95	232745266	108499.39
1	24	97556377	116435.38
2	74	240999117	175268.62
3	147	142231380	20405.38
4	89	590616191	330282.83
...	...	...	...
40288	65	57641422	24093.48
40289	50	53189544	72969.14
40290	36	28093537	35714.75
40291	3	16880390	192520.67
40292	41	315404711	281794.98

40293 rows × 3 columns

Pipeline 5 - Model Selection

Step 13: Prepare Data for Engagement Prediction & Visual EDA

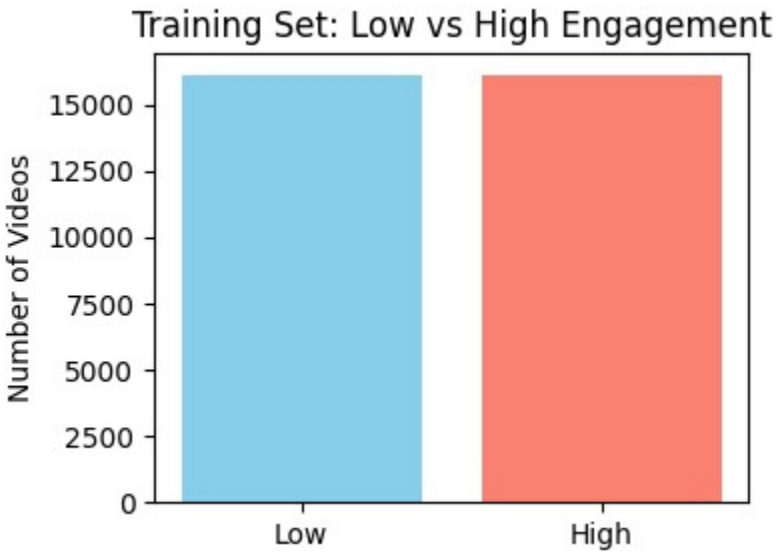
Define a new binary prediction task: classify videos into high vs low engagement (based on median `Engagement`).

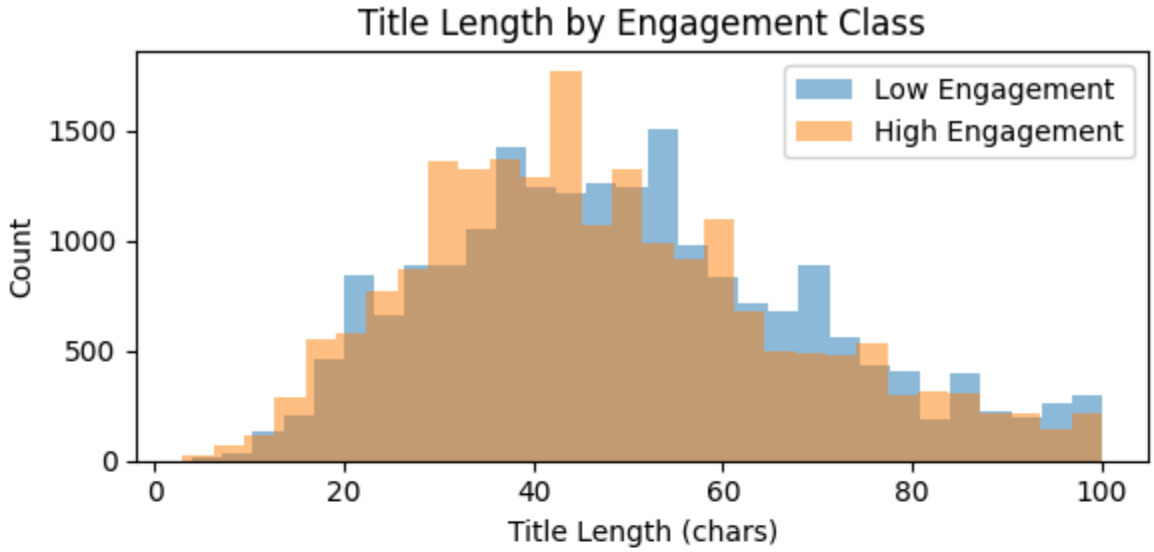
In [15]:

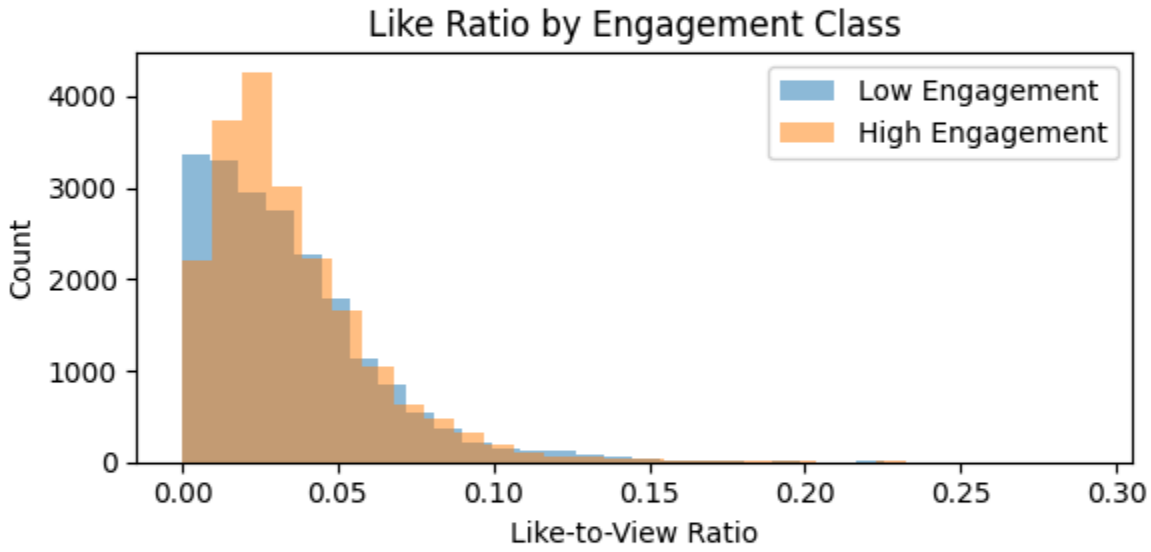
```
# Cell 13: Prepare Data for Engagement-Level Prediction & Extended EDA # ------------
---------------------------------------------------------- # 1) Load our engineered feature table
df_feat = pd.read_sql_query("SELECT * FROM features_final", conn)


# 2) Compute median Engagement (views + likes + comments)
median_eng = df_feat['Engagement'].median()
print(f"Median Engagement: {median_eng:.0f}")

# 3) Define binary target: high engagement if above median
df_feat['high_engagement'] = (df_feat['Engagement']>median_eng).astype(int)

# 4) Select input features (omit raw counts to avoid leakage)
features = [
      'TitleLength','like_ratio','comment_ratio',
      'publish_hour','publish_dayofweek',
      'num_videos','total_views','avg_likes'
]   X   =   df_feat[features]   y   =
df_feat['high_engagement']


# 5) Train/test split (stratify to keep class balance)
X_train, X_test, y_train, y_test = train_test_split(
      X, y, test_size=0.2, random_state=42, stratify=y
) print("Train shape:", X_train.shape, "Test shape:", X_test.shape)


# --- Extended Visualizations ---

# A) Class balance in training set
plt.figure(figsize=(4,3))
counts = y_train.value_counts().sort_index()
plt.bar(['Low','High'], counts.values, color=['skyblue','salmon'])
plt.title("Training Set: Low vs High Engagement")
plt.ylabel("Number of Videos")
plt.tight_layout()
plt.show()

# B) Title length distribution by engagement class
plt.figure(figsize=(6,3))
plt.hist(df_feat.loc[df_feat.high_engagement==0, 'TitleLength'],
            bins=30, alpha=0.5, label='Low Engagement')
plt.hist(df_feat.loc[df_feat.high_engagement==1, 'TitleLength'],
            bins=30, alpha=0.5, label='High Engagement')
plt.xlabel("Title Length (chars)")
plt.ylabel("Count")
plt.title("Title Length by Engagement Class")
plt.legend()
plt.tight_layout()
plt.show()

# C) Like-to-view ratio by engagement class
```

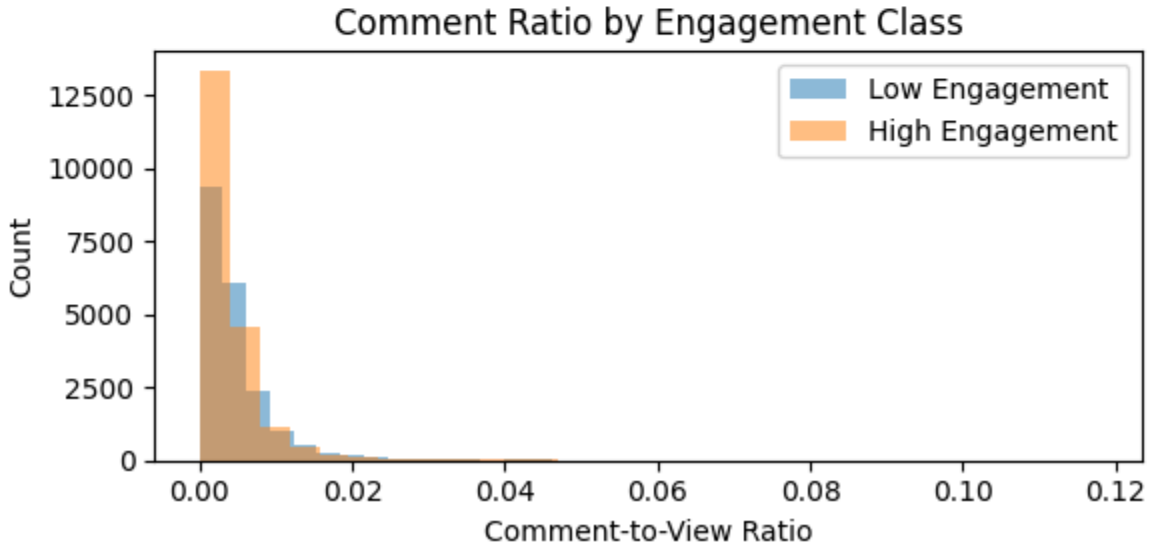
```
plt.figure(figsize=(6,3))
plt.hist(df_feat.loc[df_feat.high_engagement==0, 'like_ratio'],
             bins=30, alpha=0.5, label='Low Engagement')
plt.hist(df_feat.loc[df_feat.high_engagement==1, 'like_ratio'],
             bins=30, alpha=0.5, label='High Engagement')
plt.xlabel("Like-to-View Ratio")
plt.ylabel("Count")
plt.title("Like Ratio by Engagement Class")
plt.legend()
plt.tight_layout()
plt.show()

# D) Comment-to-view ratio by engagement class
plt.figure(figsize=(6,3))
plt.hist(df_feat.loc[df_feat.high_engagement==0, 'comment_ratio'],
             bins=30, alpha=0.5, label='Low Engagement')
plt.hist(df_feat.loc[df_feat.high_engagement==1, 'comment_ratio'],
             bins=30, alpha=0.5, label='High Engagement')
plt.xlabel("Comment-to-View Ratio")
plt.ylabel("Count")
plt.title("Comment Ratio by Engagement Class")
plt.legend()
plt.tight_layout()
plt.show()
```

Median Engagement: 706995
Train shape: (32234, 8) Test shape: (8059, 8)




Title Length by Engagement Class
Low Engagement
High Engagement
Count
0
500
1000
1500
0
20
40
60
80
100
Title Length (chars)


Like Ratio by Engagement Class
Low Engagement
High Engagement
Count
0
1000
2000
3000
4000
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Like-to-View Ratio


Comment Ratio by Engagement Class
Low Engagement
High Engagement
Count
0
2500
5000
7500
10000
12500
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Comment-to-View Ratio

Pipeline 6 - Hyperparameter Tuning

Step 14: Tuning Models Quickly Using GridSearchCV

Tune ensemble models with GridSearchCV using smaller parameter grids and fewer folds to speed up training.

In [16]:

```
# Cell 14: Faster Hyperparameter Tuning
# ------------------------------------------------
# Slimmed-down hyperparameter tuning with GridSearchCV (3-fold CV)

from sklearn.ensemble import RandomForestClassifier, \
GradientBoostingClassifier, AdaBoostClassifier
from sklearn.model_selection import GridSearchCV

# 1) Define our ensemble models
models = {
    'RandomForest':         RandomForestClassifier(random_state=42),
    'GradientBoosting':GradientBoostingClassifier(random_state=42),
    'AdaBoost':             AdaBoostClassifier(random_state=42)
}

# 2) Slimmed-down grids for quick tuning
param_grid = {
    'RandomForest': {
        'n_estimators': [100],
        'max_depth':        [None, 10]
    },
    'GradientBoosting': {
        'n_estimators':         [100],
        'learning_rate':        [0.1]
    },
    'AdaBoost': {
        'n_estimators':          [50],
            'learning_rate': [1.0]
    }
}

tuned_models = {}

# 3) Run GridSearchCV
for name, model in models.items():
    print(f"Tuning {name} (fast)...")
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid[name],
        cv=3,
        scoring='accuracy',
```

```
            n_jobs=-1
        )
        gs.fit(X_train, y_train)
        tuned_models[name] = gs.best_estimator_
        print(f"→ {name} best params: {gs.best_params_}")
```

```
Tuning RandomForest (fast)...
→ RandomForest best params: {'max_depth': None, 'n_estimators': 100}
Tuning GradientBoosting (fast)...
→ GradientBoosting best params: {'learning_rate': 0.1, 'n_estimators': 100}
Tuning AdaBoost (fast)...
→ AdaBoost best params: {'learning_rate': 1.0, 'n_estimators': 50}
```

Pipeline 7 - Model Training and Evaluation

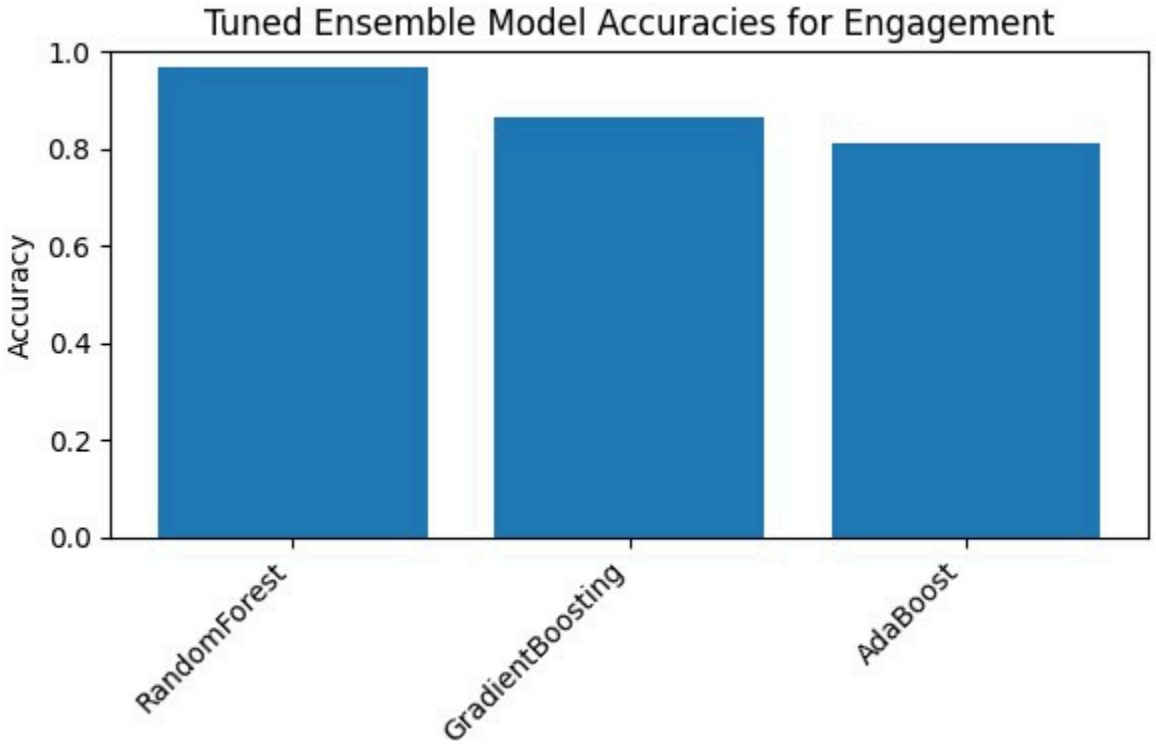
Step 15: Model Training & Evaluation

Compare the tuned ensemble models on the test set by printing their accuracies and visualizing them in a bar chart to identify the best candidate for deployment.

In [17]:

```
# Cell 15: Model Training & Evaluation
# ----------------------------------------------------

from sklearn.metrics import accuracy_score

# 1) Evaluate tuned models on test set
tuned_acc = {}
for name, model in tuned_models.items():
    preds = model.predict(X_test)
    tuned_acc[name] = accuracy_score(y_test, preds)
    print(f"→ {name} test accuracy: {tuned_acc[name]:.4f}")

# 2) Prepare data for plotting names =
list(tuned_acc.keys()) accuracies = [tuned_acc[n]
for n in names] x = np.arange(len(names))


# 3) Plot Accuracy of each Tuned Model with Bar Chart
plt.figure(figsize=(6,4))
plt.bar(x, accuracies)
plt.xticks(x, names, rotation=45, ha='right')
plt.ylim(0,1)
plt.ylabel("Accuracy")
plt.title("Tuned Ensemble Model Accuracies for Engagement")
plt.tight_layout()
plt.show()
```

```
→ RandomForest test accuracy: 0.9665
→ GradientBoosting test accuracy: 0.8635
→ AdaBoost test accuracy: 0.8120
```



Pipeline 8 - Model Deployment

Step 16: Gradio Dashboard – Insights & Model Outputs

Create an interactive dashboard using Gradio. It shows:

1. Title length distribution by engagement class
2. Tuned model accuracies
3. Top 10 channels by predicted engagement (using Random Forest)
4. Feature importance chart from Random Forest

   The dashboard makes it easy to explore the modeling results and insights visually.

In [18]:

```
# Cell 16: Gradio Dashboard — Title Length EDA & Extended Top Channels # ----------
------------------------------------------------------------

def show_dashboard():
    # 1) Title length distribution by engagement class
    fig1, ax1 = plt.subplots(figsize=(6,3))
    ax1.hist(df_feat.loc[df_feat.high_engagement==0, 'TitleLength'],
             bins=30, alpha=0.5, label='Low Engagement')
    ax1.hist(df_feat.loc[df_feat.high_engagement==1, 'TitleLength'],
             bins=30, alpha=0.5, label='High Engagement')
    ax1.set_xlabel("Title Length (chars)")
    ax1.set_ylabel("Count")
```

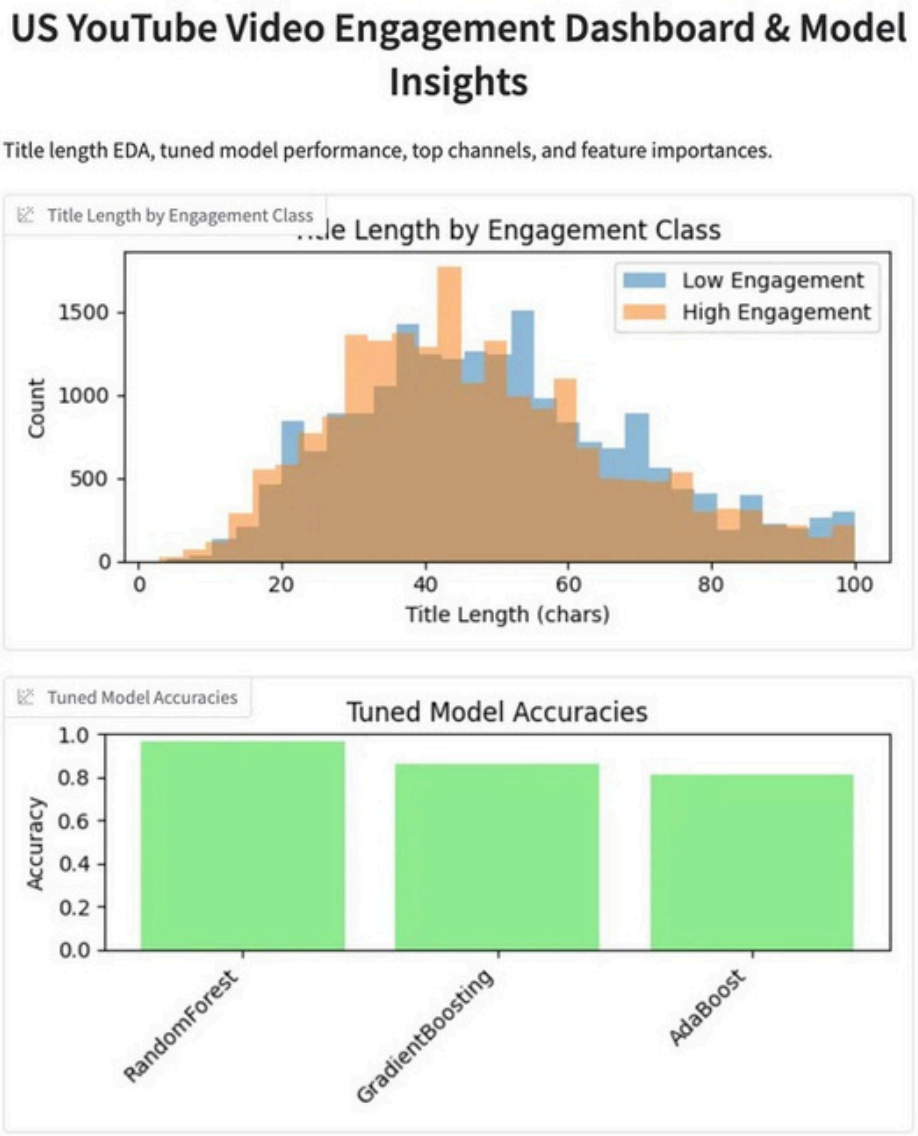
```
    ax1.set_title("Title Length by Engagement Class")
    ax1.legend()
    fig1.tight_layout()

    #2)Tuned model accuracies
    fig2,    ax2 = plt.subplots(figsize=(6,3))
    names_list = list(tuned_acc.keys())
    accs  = [tuned_acc[n] for n in names_list]
    x = np.arange(len(names_list))
    ax2.bar(x, accs, color='lightgreen')
    ax2.set_xticks(x)
    ax2.set_xticklabels(names_list, rotation=45, ha='right')
    ax2.set_ylim(0,1)
    ax2.set_ylabel("Accuracy")
    ax2.set_title("Tuned Model Accuracies")
    fig2.tight_layout()

    #3)Top-10 channels by average predicted engagement using the best
    #RandomForest model best_model = tuned_models['RandomForest'] probs =
    best_model.predict_proba(X_test)[:,1]  video_meta  =  df_feat.loc[X_test.index,
    ['channel_title']].copy()  video_meta['pred_high_eng'] = probs channel_probs =
    video_meta.groupby(


        'channel_title'
    )['pred_high_eng'].mean()
    top10_channels = (
        channel_probs
        .sort_values(ascending=False)
        .head(10)
        .reset_index()
        .rename(columns={'pred_high_eng': 'avg_pred_high_eng'})
    )

    #4) Feature importances of the RandomForest
    importances = best_model.feature_importances_
    imp_df = pd.Series(importances, index=features).sort_values()
    fig4, ax4 = plt.subplots(figsize=(6,3))
    imp_df.plot.barh(ax=ax4)
    ax4.set_title("RandomForest Feature Importances")
    fig4.tight_layout()

    #Return: title-length hist, accuracies bar, top10 DataFrame,
    # importances
    return fig1, fig2, top10_channels, fig4

gr.Interface(
    fn=show_dashboard,
    inputs=[],
    outputs=[
        gr.Plot(label="Title Length by Engagement Class"),
        gr.Plot(label="Tuned Model Accuracies"),
        gr.Dataframe(label="Top 10 Channels by Predicted Engagement"),
        gr.Plot(label="Feature Importances")
    ],
    title="US YouTube Video Engagement Dashboard & Model Insights",
```

```
        description = (
            "Title length EDA, tuned model performance, "
            "top channels, and feature importances."
        )
).launch()
```

It looks like you are running Gradio on a hosted Jupyter notebook, which req uires `share=True`. Automatically setting `share=True` (you can turn this of f by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://2f36a8a75a9556fa45.gradio.live

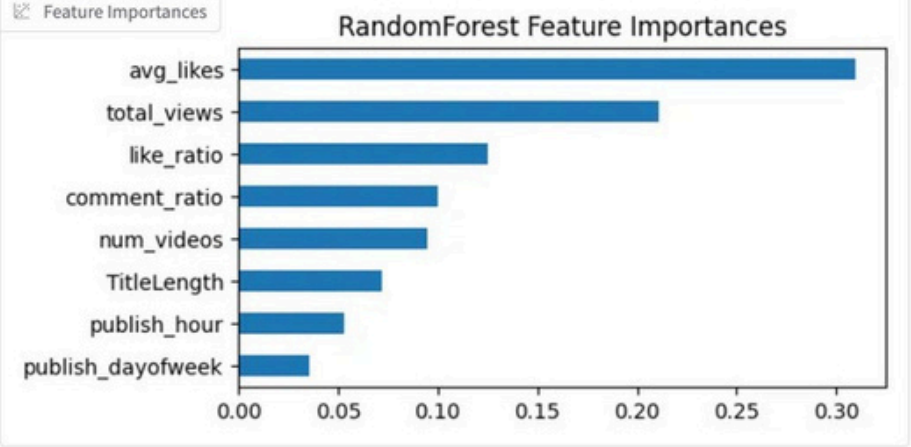
This share link expires in 1 week. For free permanent hosting and GPU upgrad es, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)



Top 10 Channels by Predicted Engagement

channel_title	avg_pred_high_eng
Adam Smith	1
Access	1
Adult Swim	1
gameslice	1
jacksepticeye	1
Matt Bentkowski	1
MalumaVEVO	1
LuisFonsiVEVO	1
Kia Motors America	1
boogie2988	1

Feature Importances



Clear Generate Flag