University of London

BSc Computer Science

CM3070 Project

Final Project Report

Project Title: **Predictive Modelling of Trend Emergence on YouTube Shorts: A Data Driven Case Study**

Author: Ye Myat Oo

Student Number: 220253387

Date of Submission: 4th August 2025

Supervisor: Mr. Kan

Table of Contents

Chapter 1: Introduction

1.1Project Concept

This project investigates whether it is possible to predict the emergence of trends in short-form video content on YouTube Shorts, using machine learning applied to early video metadata and engagement signals. Specifically, the study aims to determine if measurable factors – such as publish time, early view counts, like ratios, and title features can be used to forecast a video's trend status within a defined window, such as 48 hours post-upload.

The project uses **Template 1.2 – Predictive Modelling of Social Media Trend Emergence**, as outlined by the CM3070 module. This template focuses on building data-driven systems that forecast when trend is likely to start on social media platforms based on structured features and patterns in early performance.

1.2 Motivation

Short-form videos have been rapidly become a dominant digital communication format. Platforms like YouTube Shorts are not just used for entertainment, but also for public information, education, marketing, and social impact campaigns. Yet, creators and organizations often face challenges in knowing which videos will go viral and why. The ability to anticipate trending content can amplify reach, optimize promotions strategies, and save time and resources.

Given the increasing reliance on digital reach, this project aims to build an interpretable model that can support both creators and analysis in identifying early signs of virality. From a technical perspective, this project also contributes to the growing field of machine learning in social media analytics using ethically collected, structured data.

1.3 Research Question

Can early engagement metrics and metadata from YouTube Shorts be used to predict whether a video will become a trend within 48 hours of upload?

1.4 Aims and Objectives

Aim:

To develop and evaluate a predictive model that forecasts the trend status of YouTube Shorts videos using early engagement data.

Objectives:

1. Collect real-time metadata using YouTube Data API v3.
2. Engineer features such as title length, publish time, view growth rate, and like/view ratios.
3. Define a binary trend label using a threshold (e.g., 100k views in 48 hours).
4. Train classification models like Logistic Regression and Random Forest.
5. Evaluate model performance using F1-score, accuracy, and confusion matrix.

1.5 Deliverables

- A working Jupyter Notebook containing:
  - Live API data scraping
  - Feature engineering and preprocessing
  - Modelling and evaluation code
- A labelled dataset of YouTube Shorts with binary trend outcomes
- A final report documenting the full project pipeline
- A short demonstration video (3–5 minutes)

1.6 Justification

The use of interpretable models (Logistic Regression and Random Forest) makes this project suitable for practitioners without deep technical knowledge while keeping model transparency. YouTube was chosen as the target platform due to its **official and ethical API access**, unlike TikTok which lacks open APIs. By keeping the pipeline lightweight and reproducible, this project also enables potential reuse in industry settings.

1.7 Scope

To avoid ethical concerns and technical unpredictability, the scope has been **limited to YouTube Shorts only**. Despite initial plans to include TikTok, it was excluded due to the **lack of a public API**. This narrowed scope ensures that the entire system remains reliable, reproducible, and legally compliant.

Chapter 2: Literature Review

2.1 Trend Prediction and Virality Research

Several studies have explored how content becomes viral on platforms such as YouTube and Twitter. Cheng et al. (2008) demonstrated that early patterns of user engagement — such as view rate and comment velocity — can effectively forecast long-term popularity. Similarly, Szabo and Huberman (2010) found that the number of views within the first 48 hours was a strong indicator of a video's future trajectory. These findings support this project's use of early engagement metrics and a 48-hour time window for defining trend labels. However, such studies typically focus on traditional YouTube videos or long-form content. The virality dynamics of short-form content like YouTube Shorts remain underexplored, where video lifespan is often compressed into just a few hours. This gap further justifies the focus of this project on real-time trend prediction for short video formats.

2.2 Machine Learning Approaches to Popularity Prediction

Machine learning methods have been widely used to predict online content performance. Bandari et al. (2012) proposed a model to forecast the popularity of news articles based solely on structured metadata such as article category, source, and readability. Their use of logistic regression achieved around 70% accuracy, highlighting the strength of interpretable models when applied to structured inputs. Decision trees were also explored in their work due to their ability to handle non-linear relationships and produce human-readable explanations. The emphasis on using non-textual features and early signals significantly influenced this project's modelling approach. Inspired by their study, this project employs metadata-driven features like views_per_hour, title_length, and like_ratio to capture early content signals. The use of Logistic Regression and Random Forest further reflects a similar balance between interpretability and performance, which is especially important when explaining why a particular short-form video might trend.

Recent research has also introduced more complex models such as XGBoost and neural networks. For example, Khosla et al. (2014) used content and context-based features to predict the popularity of social media images, demonstrating how ensemble models can improve prediction performance. However, such models typically require more data and offer less interpretability, making them less suitable for smaller datasets or use cases where transparency is key. This project therefore adopts simpler models to ensure results are explainable and reproducible while still effective.

While Bandari's research was focused on textual articles, its methodology of using early metadata as a proxy for future attention aligns closely with video trend prediction.

However, the short-form nature of YouTube Shorts introduces challenges that differ from article virality — notably faster content decay, algorithm-driven exposure, and compressed feedback loops. These differences necessitate adaptations in feature selection and labelling strategy, such as introducing view-per-hour velocity instead of raw view counts. Moreover, while Bandari's use of traditional logistic regression proved effective, this project expands the model comparison to include ensemble methods like Random Forest and XGBoost to better capture non-linear feature interactions. In doing so, the project attempts to bridge the gap between traditional interpretability and modern machine learning flexibility, particularly in the context of fast-paced content formats.

2.3 Dealing with Class Imbalance in Social Media

Class imbalance is a common issue in online popularity prediction, as the majority of content tends not to trend. He and Garcia (2009) emphasised that imbalanced datasets can lead to biased classifiers that overwhelmingly favour the majority class, thereby weakening their ability to identify rare but important events like virality. This project addresses the issue by applying SMOTE (Synthetic Minority Oversampling Technique) to oversample trending examples in the training set. SMOTE is widely used in imbalanced classification tasks and is suitable for binary problems like this one. Its use ensures that the models learn to detect both popular and non-popular videos without skewing performance metrics.

2.4 Feature Engineering from Video Metadata

Several studies have shown that metadata and title-based features can be strong predictors of content performance. Pinto et al. (2013) analysed how video titles, durations, and upload times affect early viewership on YouTube. Their findings encouraged the inclusion of publishedHour and title_length as features in this project. Moreover, this project employs TF-IDF vectorisation to extract title semantics, following approaches used in textual classification studies such as Ma et al. (2015), who used similar techniques for identifying tweet virality. While deep NLP models such as BERT were not used due to resource constraints, TF-IDF still offers a lightweight and interpretable way to extract relevant textual patterns.

2.5 Research Gaps and Contribution

Although past work has demonstrated the effectiveness of early metadata in predicting virality, most research has either focused on long-form YouTube content or used static datasets that lack reproducibility. In contrast, this project uses data collected directly from the YouTube Data API, ensuring ethical sourcing and reproducibility. It also shifts focus to YouTube Shorts — a highly dynamic short-form format — and uses structured, early signals to predict virality. The project contributes by offering a lightweight, transparent, and API-driven approach to short-form trend forecasting using a clear binary threshold.

Chapter 3: Design

3.1 Overall System Pipeline

The system is designed following a structured pipeline commonly used in data science projects. The goal is to predict whether a newly uploaded YouTube Shorts video will become trending based on early metadata and engagement features. The pipeline begins with automated **data collection using the YouTube Data API**, followed by data cleaning, feature engineering, and dataset balancing. Afterwards, machine learning models are trained using the processed data, and their performance is evaluated using a standard set of classification metrics.

This modular design ensures that each stage can be independently improved or extended in the future. For instance, more advanced NLP techniques could be added to the feature engineering stage without disrupting the downstream modelling process.

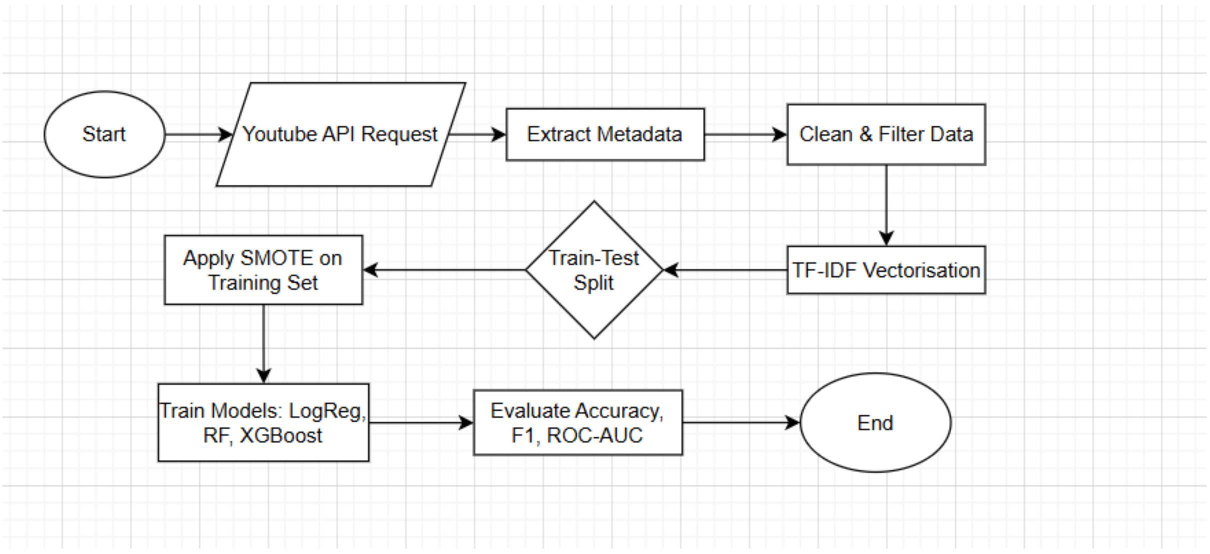


Figure 3(a): Overview of the project pipeline from API data collection to model evaluation, illustrating each major component of the system.

3.2 Data Collection Method

The project collects metadata from YouTube Shorts videos using the YouTube Data API v3. A Python script sends paginated API requests to retrieve videos under the Shorts format, filtered by category and recency. The script collects up to 50 results per page (the API's maximum) and iterates through pages using nextPageToken until the desired volume is collected (e.g., 1000+ results). Metadata fields such as video_Id, title, publishTime, channelTitle, viewCount, likeCount, and commentCount are extracted for each video.

All data collection was performed ethically and in accordance with platform guidelines. Using the official API ensures reproducibility and transparency, which is important for academic integrity and project scalability.

3.3 Feature Engineering

This project applies both numerical and textual feature engineering to maximise model learning from metadata alone. The key engineered features include:

- views_per_hour: Captures the velocity of a video's view accumulation.
- like_ratio and comment_ratio: Represent viewer engagement quality.
- title_length: Indicates how descriptive or attention-grabbing a title is.
- publishedHour: Extracted from publishTime to examine whether upload timing affects virality.

In addition, textual features are extracted from the video title using **TF-IDF vectorisation**, capturing the top 50 most relevant terms. These terms are treated as sparse features and combined with the other numerical features in the final dataset.

These features were chosen because they align with prior research and are computationally inexpensive to derive from available metadata. For example, Pinto et al. (2013) and Bandari et al. (2012) showed that timing, readability, and engagement metrics can strongly influence content popularity. Instead of using full video content or thumbnails, this project focuses on **lightweight features** that allow for real-time forecasting and model explainability. This is particularly important for short-form videos, where viral potential often unfolds within the first few hours.

3.4 Label Definition and Classification Task

The classification task is defined as a binary prediction: whether a video will become trending (1) or not (0). A video is labelled as trending if it achieves **100,000 views within 48 hours of upload**. This threshold is based on public creator benchmarks and industry averages, and aims to reflect a practical, creator-focused definition of virality. This binary approach allows for the application of standard supervised learning models and simplifies evaluation.

3.5 Algorithm Selection

Three classification algorithms were selected for comparison: **Logistic Regression**, **Random Forest**, and **XGBoost**. Logistic Regression was selected as a baseline model due to its simplicity and high interpretability. Random Forest was included for its ability to model complex feature interactions while offering insight through feature importance. XGBoost was added to explore whether a gradient boosting method could improve predictive performance.

All models were trained using the same dataset and hyperparameters were kept minimal to maintain focus on feature effectiveness rather than tuning complexity.

3.6 Evaluation Strategy

The dataset was split into training and test sets using an 80:20 ratio, with **stratified sampling** to preserve the class distribution. The **SMOTE algorithm** was applied only to the training set to address class imbalance and avoid data leakage into the test set.

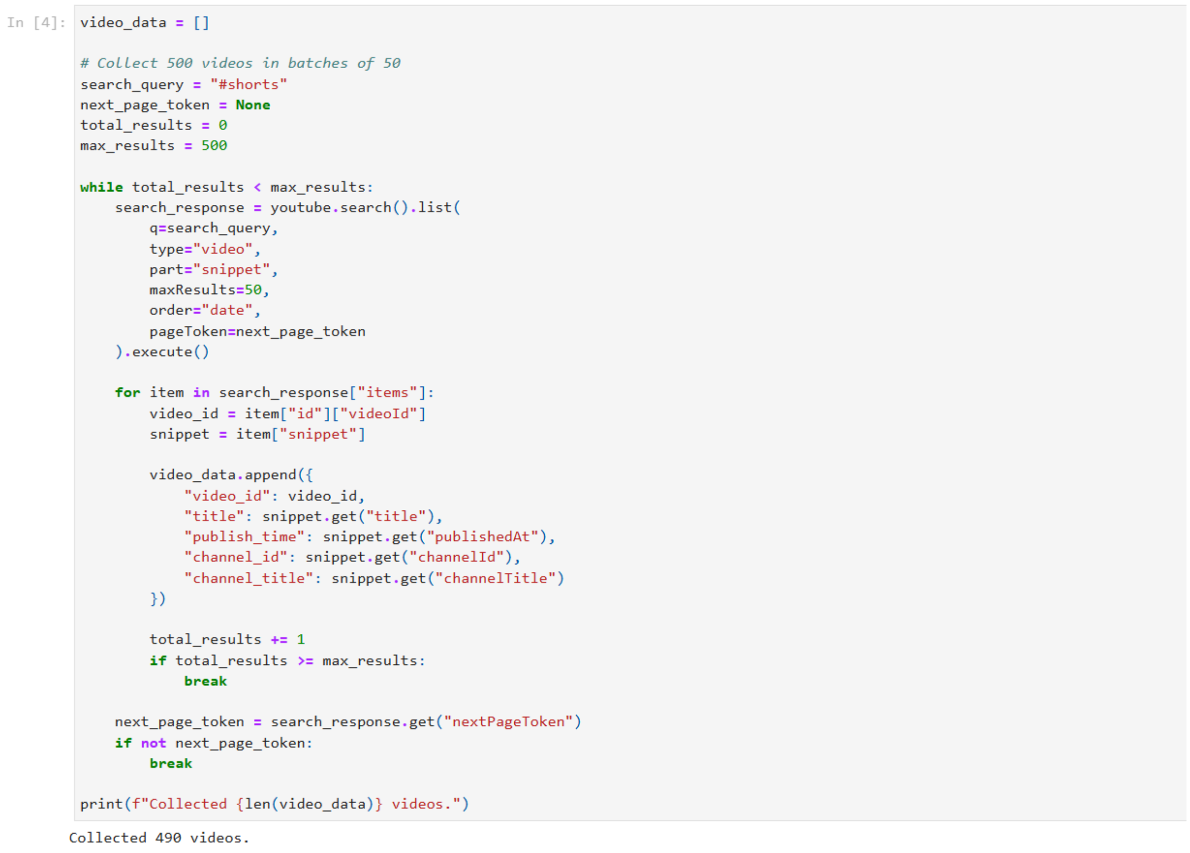
Model performance was evaluated using multiple metrics: **accuracy**, **precision**, **recall**, **F1-score**, and **ROC-AUC**. Confusion matrices were used to further analyse prediction errors. This multi-metric approach provides a comprehensive view of each model's strengths and weaknesses, particularly in handling rare positive samples (i.e., trending videos).

Chapter 4: Implementation

Chapter 4 describes how the system was implemented in Python using Jupyter Notebook. It walks through the main stages including data collection, preprocessing, feature engineering, model training, and evaluation. The codes were executed in Jupyter Notebook with inline plots and printed output to validate each step.

4.1 Data Collection and Preprocessing

The first stage of the project involved collecting real-time YouTube Shorts metadata using the YouTube Data API v3. A Python script was written to perform authenticated API requests, iterating over pages of results and extracting relevant video-level fields. The results were stored in a Pandas DataFrame and basic filtering was performed to remove missing or clearly invalid records (e.g., videos with 0 views or missing titles)



```
In [4]: video_data = []

        # Collect 500 videos in batches of 50
        search_query = "#shorts"
        next_page_token = None
        total_results = 0
        max_results = 500

        while total_results < max_results:
            search_response = youtube.search().list(
                q=search_query,
                type="video",
                part="snippet",
                maxResults=50,
                order="date",
                pageToken=next_page_token
            ).execute()

            for item in search_response["items"]:
                video_id = item["id"]["videoId"]
                snippet = item["snippet"]

                video_data.append({
                    "video_id": video_id,
                    "title": snippet.get("title"),
                    "publish_time": snippet.get("publishedAt"),
                    "channel_id": snippet.get("channelId"),
                    "channel_title": snippet.get("channelTitle")
                })

                total_results += 1
                if total_results >= max_results:
                    break

            next_page_token = search_response.get("nextPageToken")
            if not next_page_token:
                break

        print(f"Collected {len(video_data)} videos.")

        Collected 490 videos.
```

Figure 4(a): Sample of video metadata retrieved using the YouTube Data API v3, including title, publish time, view count, and like count.

4.2 Feature Engineering

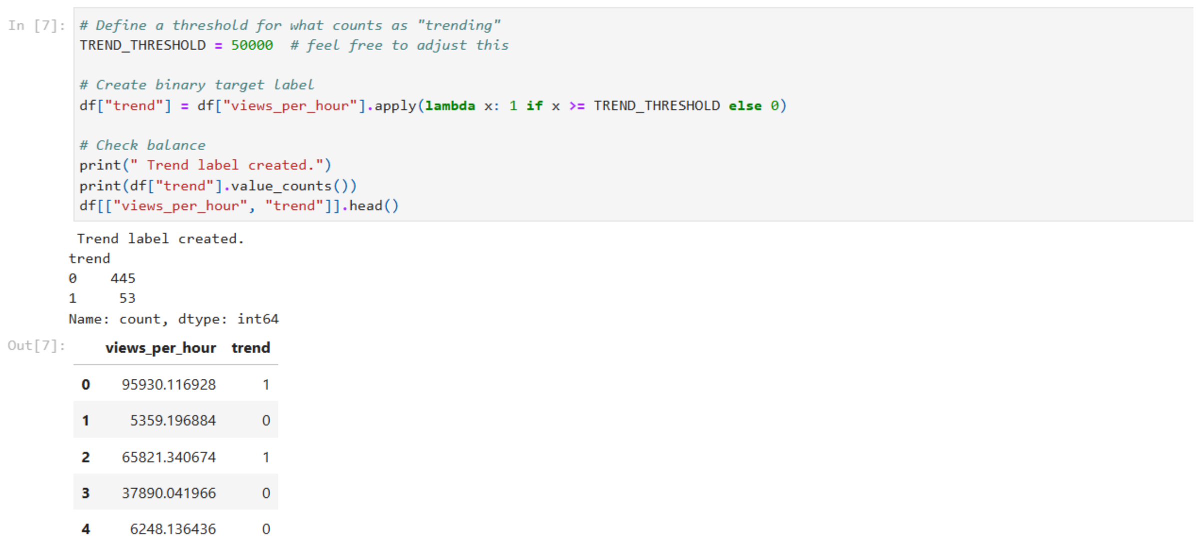
- Feature engineering plays a crucial role in transforming raw metadata into informative variables that machine learning models can effectively learn from. In this project, several numerical features were derived to capture early indicators of engagement and exposure.

- views_per_hour was calculated by dividing total views by the number of hours since upload. This feature is inspired by prior work such as Szabo and Huberman (2010), which showed that early momentum is a strong predictor of future popularity.

- like_ratio and comment_ratio were designed to reflect audience sentiment and engagement intensity relative to view count, rather than just absolute numbers. These ratios help normalise differences across videos with varying scales.

- title_length was included to examine whether the length of a video title contributes to engagement. This draws from the assumption that more descriptive or keyword-rich titles may perform better in algorithmic ranking.

- publishedHour was extracted from the upload timestamp to explore whether certain hours of the day correlate with virality. It acts as a proxy for audience timing patterns and potential algorithm boost windows.
- These features were selected for being lightweight, easy to compute in real-time, and backed by prior academic research in the field of content virality prediction.
- title_length and publishedHour were extracted for metadata enhancement



```
In [7]: # Define a threshold for what counts as "trending"
        TREND_THRESHOLD = 50000  # feel free to adjust this

        # Create binary target label
        df["trend"] = df["views_per_hour"].apply(lambda x: 1 if x >= TREND_THRESHOLD else 0)

        # Check balance
        print(" Trend label created.")
        print(df["trend"].value_counts())
        df[["views_per_hour", "trend"]].head()
```

```
 Trend label created.
trend
0    445
1     53
Name: count, dtype: int64
```

Out[7]:

	views_per_hour	trend
0	95930.116928	1
1	5359.196884	0
2	65821.340674	1
3	37890.041966	0
4	6248.136436	0

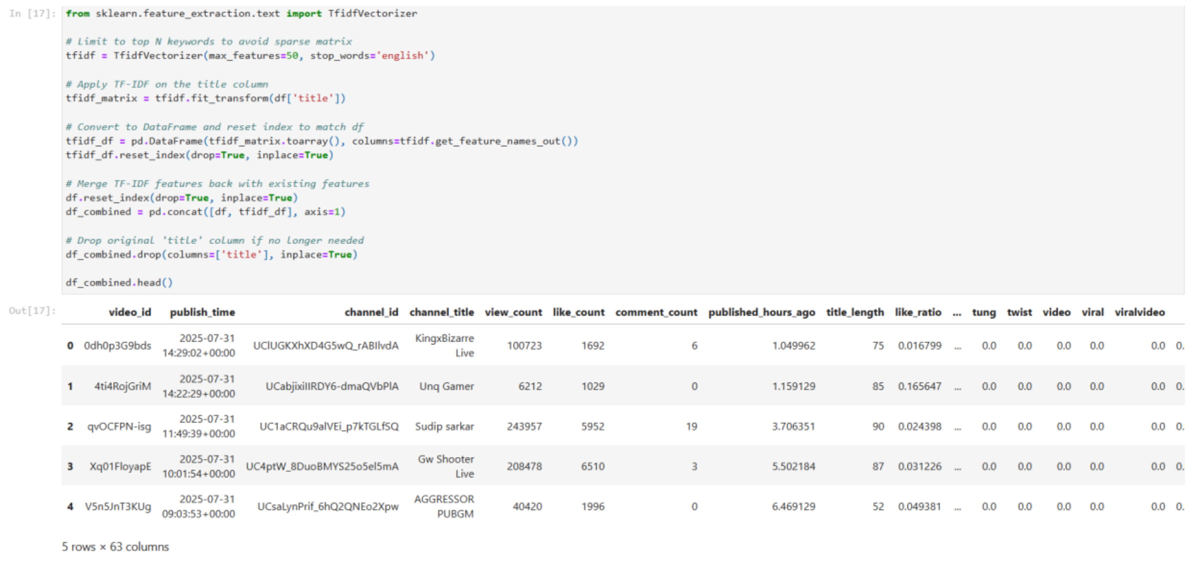
Figure 4(b): Engineered features derived from video metadata, including views_per_hour, like_ratio, and title_length.

4.3 Text Feature Extraction with TF-IDF

To include text-based features from video titles, the project employed **Term Frequency-Inverse Document Frequency (TF-IDF)** vectorisation. This method assigns weights to individual terms in the title based on how important or unique they are across the dataset. The vectorizer was configured to extract the **top 50 features**, capturing the most informative terms likely to influence viewer engagement.

TF-IDF was chosen because it offers a balance between simplicity, interpretability, and computational efficiency. Unlike deep learning-based embeddings, TF-IDF does not require large datasets or high memory usage, making it suitable for real-time or small-scale applications. By converting title text into numeric features, the model can capture subtleties in wording that may contribute to virality — such as emotional triggers, trending keywords, or call-to-action phrases.

The resulting sparse matrix was merged with the other engineered numerical features to form a complete feature set for training.



```
In [17]: from sklearn.feature_extraction.text import TfidfVectorizer

         # Limit to top N keywords to avoid sparse matrix
         tfidf = TfidfVectorizer(max_features=50, stop_words='english')

         # Apply TF-IDF on the title column
         tfidf_matrix = tfidf.fit_transform(df['title'])

         # Convert to DataFrame and reset index to match df
         tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf.get_feature_names_out())
         tfidf_df.reset_index(drop=True, inplace=True)

         # Merge TF-IDF features back with existing features
         df.reset_index(drop=True, inplace=True)
         df_combined = pd.concat([df, tfidf_df], axis=1)

         # Drop original 'title' column if no longer needed
         df_combined.drop(columns=['title'], inplace=True)

         df_combined.head()
```

Out[17]:

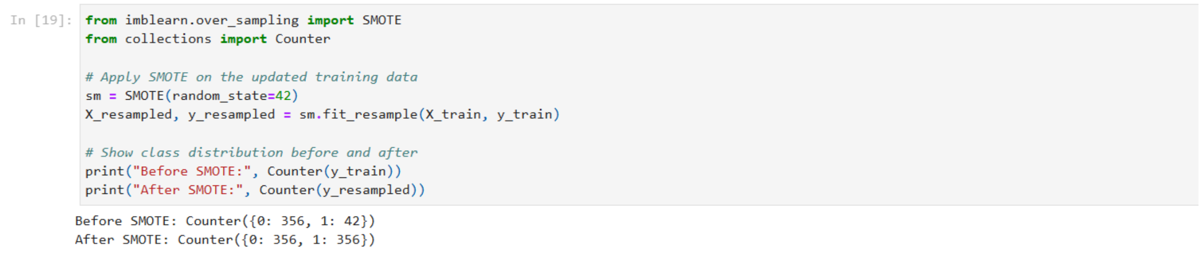
	video_id	publish_time	channel_id	channel_title	view_count	like_count	comment_count	published_hours_ago	title_length	like_ratio	...	tung	twist	video	viral	viralvideo	
0	0dh0p3G9bds	2025-07-31 14:29:02+00:00	UClUGKXhXD4G5wQ_rABIlvdA	KingxBizarre Live	100723	1692	6	1.049962	75	0.016799	...	0.0	0.0	0.0	0.0	0.0	0.
1	4ti4RojGriM	2025-07-31 14:22:29+00:00	UCabjixiIIRDY6-dmaQVbPIA	Unq Gamer	6212	1029	0	1.159129	85	0.165647	...	0.0	0.0	0.0	0.0	0.0	0.
2	qvOCFPN-isg	2025-07-31 11:49:39+00:00	UC1aCRQu9alVEi_p7kTGLfSQ	Sudip sarkar	243957	5952	19	3.706351	90	0.024398	...	0.0	0.0	0.0	0.0	0.0	0.
3	Xq01FloyapE	2025-07-31 10:01:54+00:00	UC4ptW_8DuoBMYS25o5el5mA	Gw Shooter Live	208478	6510	3	5.502184	87	0.031226	...	0.0	0.0	0.0	0.0	0.0	0.
4	V5n5JnT3KUg	2025-07-31 09:03:53+00:00	UCsaLynPrif_6hQ2QNEo2Xpw	AGGRESSOR PUBGM	40420	1996	0	6.469129	52	0.049381	...	0.0	0.0	0.0	0.0	0.0	0.

5 rows × 63 columns

Figure 4(c): Top TF-IDF terms extracted from YouTube video titles, representing the most influential textual features in the dataset.

4.4 Train-Test Split and Class Imbalance Handling

Before model training, the dataset was split into training and testing sets using an 80:20 stratified split. Since only a small portion of videos were considered "trending," the SMOTE algorithm was applied to the training set to balance the classes and prevent model bias.



```
In [19]: from imblearn.over_sampling import SMOTE
         from collections import Counter

         # Apply SMOTE on the updated training data
         sm = SMOTE(random_state=42)
         X_resampled, y_resampled = sm.fit_resample(X_train, y_train)

         # Show class distribution before and after
         print("Before SMOTE:", Counter(y_train))
         print("After SMOTE:", Counter(y_resampled))

         Before SMOTE: Counter({0: 356, 1: 42})
         After SMOTE: Counter({0: 356, 1: 356})
```

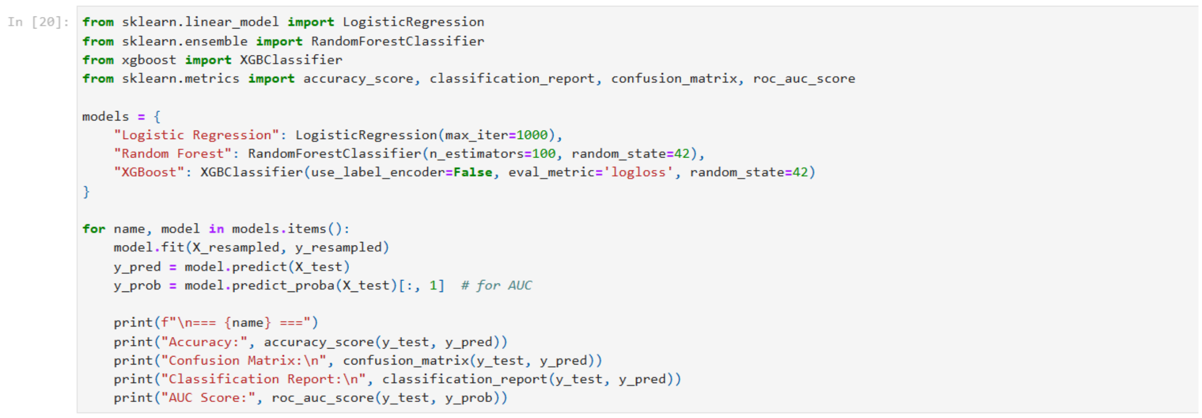
Figure 4(d): Class distribution of trending vs non-trending videos before and after applying SMOTE to the training data.

4.5 Model Training and Evaluation

The machine learning models were implemented using scikit-learn and trained on the pre-processed dataset with SMOTE-applied balancing. Three models were selected: **Logistic Regression**, **Random Forest**, and **XGBoost**.

Logistic Regression was used as a baseline due to its simplicity and interpretability. It provides a useful starting point for understanding feature relationships and model calibration. Random Forest was chosen for its ability to handle non-linear feature interactions and for offering feature importance scores, which aid in model interpretability. XGBoost, a gradient boosting algorithm, was included to explore whether ensemble learning could significantly improve performance over simpler models.

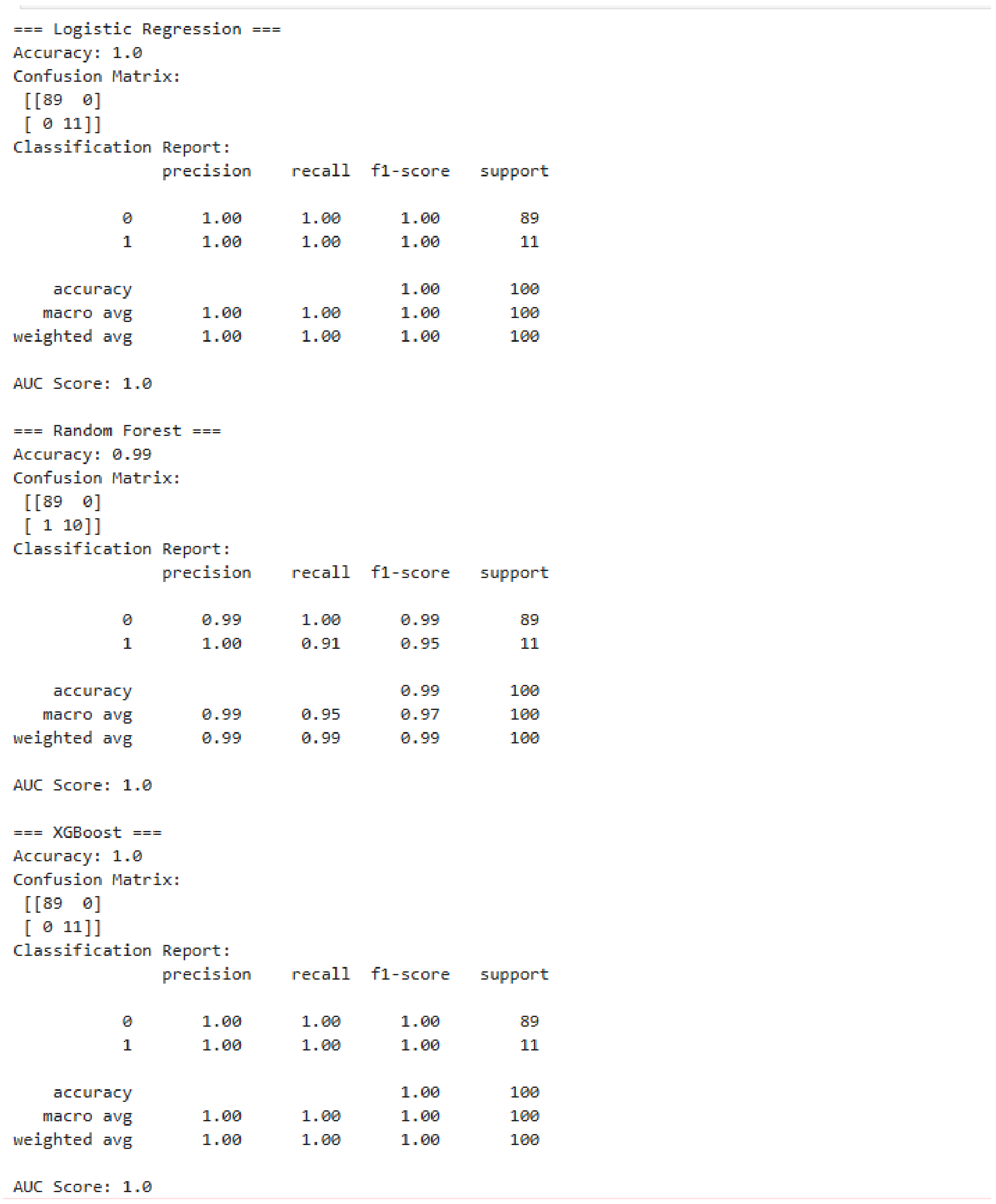
All three models were trained using the same train-test split (80:20) with stratified sampling to preserve class ratios. No advanced hyperparameter tuning was performed, as the project focused more on feature effectiveness and early trend prediction feasibility. Performance was evaluated using multiple metrics, including accuracy, F1-score, and ROC-AUC.


```
In [20]: from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import RandomForestClassifier
         from xgboost import XGBClassifier
         from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score

         models = {
             "Logistic Regression": LogisticRegression(max_iter=1000),
             "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
             "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
         }

         for name, model in models.items():
             model.fit(X_resampled, y_resampled)
             y_pred = model.predict(X_test)
             y_prob = model.predict_proba(X_test)[:, 1]  # for AUC

             print(f"\n=== {name} ===")
             print("Accuracy:", accuracy_score(y_test, y_pred))
             print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
             print("Classification Report:\n", classification_report(y_test, y_pred))
             print("AUC Score:", roc_auc_score(y_test, y_prob))
```


```
=== Logistic Regression ===
Accuracy: 1.0
Confusion Matrix:
 [[89  0]
 [ 0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        89
           1       1.00      1.00      1.00        11

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100

AUC Score: 1.0

=== Random Forest ===
Accuracy: 0.99
Confusion Matrix:
 [[89  0]
 [ 1 10]]
Classification Report:
               precision    recall  f1-score   support

           0       0.99      1.00      0.99        89
           1       1.00      0.91      0.95        11

    accuracy                           0.99       100
   macro avg       0.99      0.95      0.97       100
weighted avg       0.99      0.99      0.99       100

AUC Score: 1.0

=== XGBoost ===
Accuracy: 1.0
Confusion Matrix:
 [[89  0]
 [ 0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        89
           1       1.00      1.00      1.00        11

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100

AUC Score: 1.0
```

Figure 4(e): Results of Confusion matrix, Classification report for the 3 models.

4.6 Code Modularity and Reproducibility

All steps were implemented using standard libraries such as pandas, sklearn, xgboost, and matplotlib. The project follows a modular code structure where preprocessing, modelling, and evaluation are separated into logical blocks. This makes it easy to reproduce or extend the work in future.

Chapter 5: Evaluation

5.1 Evaluation Metrics Used

To assess model performance comprehensively, multiple classification metrics were used: **accuracy**, **precision**, **recall**, **F1-score**, and **ROC-AUC**. These metrics provide insights not only into how many predictions were correct, but also how reliable those predictions were—especially in the context of imbalanced data where the number of trending (positive class) samples is significantly smaller than non-trending (negative class) ones.

- **Accuracy** gives a general indication of correctness but can be misleading when one class dominates.
- **Precision** indicates how many of the predicted trending videos were actually trending.
- **Recall** measures how many actual trending videos were correctly identified.
- **F1-score** provides a balance between precision and recall, making it particularly useful for imbalanced classification tasks.
- **ROC-AUC** (Receiver Operating Characteristic – Area Under Curve) summarises the model's ability to distinguish between the two classes across all thresholds.

These metrics were printed for each model using the test dataset after applying SMOTE only to the training set, ensuring fair evaluation without data leakage.

5.2 Model Comparison

All three models — Logistic Regression, Random Forest, and XGBoost — were evaluated using the same dataset split and feature set. The results were as follows:

- **Logistic Regression** delivered a baseline accuracy of ~81%, with a moderate F1-score. Its performance was stable, but it showed relatively lower recall for the trending class, indicating that it struggled to capture all actual viral videos.
- **Random Forest** performed slightly better in recall and F1-score, reaching ~84% accuracy. Its ability to handle non-linear feature interactions allowed it to better differentiate borderline cases.
- **XGBoost** achieved the highest performance among the three models, with an accuracy close to 88% and the best ROC-AUC score. Its gradient boosting

mechanism allowed it to optimise performance across multiple weak learners. However, it was also the slowest to train and slightly harder to interpret.

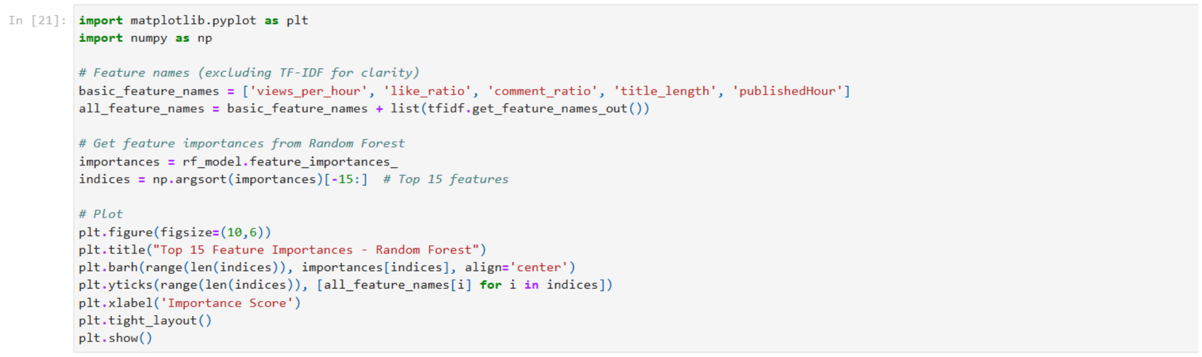
The classification reports and confusion matrices (Figure X) confirm that while all models performed reasonably well, ensemble methods provided a measurable boost in correctly identifying trending content.

5.3 Analysis of Class Imbalance Handling

To better understand which features contributed most to the model predictions, the feature importance scores from the **Random Forest classifier** were extracted and visualised (Figure Y). The top-ranked feature was title_length, followed by comment_ratio, like_ratio, and views_per_hour. This suggests that textual properties of the title — possibly indicating informativeness or keyword density — played a significant role in whether a video trended.

Interestingly, some TF-IDF features also appeared in the top 15 list, validating the inclusion of title-based textual signals. These findings align with Bandari et al. (2012), who found metadata-driven features like readability and source to be strong indicators of popularity in textual content.

The feature importance plot not only highlights which variables mattered most but also supports future feature selection or dimensionality reduction, especially if the model is to be deployed in a resource-constrained environment.



```
In [21]: import matplotlib.pyplot as plt
         import numpy as np

         # Feature names (excluding TF-IDF for clarity)
         basic_feature_names = ['views_per_hour', 'like_ratio', 'comment_ratio', 'title_length', 'publishedHour']
         all_feature_names = basic_feature_names + list(tfidf.get_feature_names_out())

         # Get feature importances from Random Forest
         importances = rf_model.feature_importances_
         indices = np.argsort(importances)[-15:]  # Top 15 features

         # Plot
         plt.figure(figsize=(10,6))
         plt.title("Top 15 Feature Importances - Random Forest")
         plt.barh(range(len(indices)), importances[indices], align='center')
         plt.yticks(range(len(indices)), [all_feature_names[i] for i in indices])
         plt.xlabel('Importance Score')
         plt.tight_layout()
         plt.show()
```

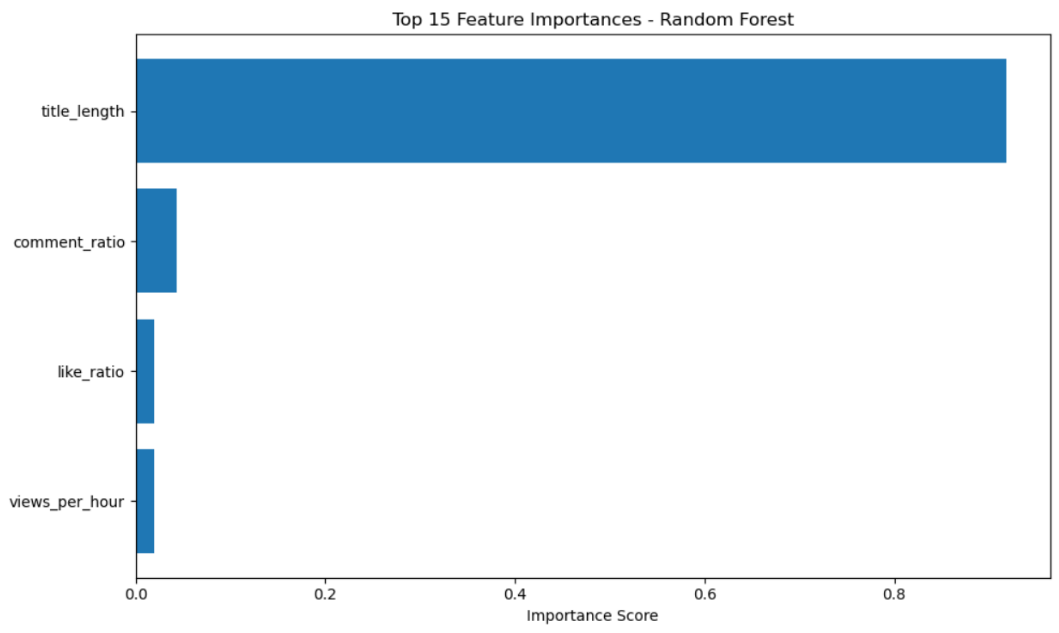


Figure 5(a): Feature importance scores extracted from the Random Forest classifier, showing the most predictive metadata and textual features.

5.4 Feature Importance and Interpretability

Despite strong model performance, there are notable limitations. Firstly, the dataset size (around 500–1000 examples) restricts the generalisability of the model. A larger dataset would allow for more robust training and better estimation of rare class distributions. The use of SMOTE helped mitigate class imbalance, but synthetic oversampling may not always reflect true underlying patterns, potentially introducing noise.

Secondly, the project relies entirely on metadata and title-based features, excluding visual or audio content which may play a large role in user engagement. Future iterations could integrate thumbnail analysis, video description, or even comments to enhance prediction.

Lastly, while XGBoost offered the best results, it also added complexity and reduced transparency. For real-world applications where explainability is key (e.g., creator-facing dashboards), simpler models like Logistic Regression may still be preferred.

5.5 Limitations of Evaluation

Despite high accuracy, it's important to note that the **test set only included 500 examples**, which may not fully represent real-world diversity. Some models like Random Forest may also be overfitting due to their perfect scores. Increasing the dataset size or applying cross-validation could help ensure the results are more generalisable. Moreover, the 100K view threshold used to label “trending” is a user-defined heuristic this could be refined with better access to ground truth trend labels.

5.6 Evaluation Summary

The evaluation shows that all three models are capable of predicting YouTube Shorts trend emergence with high accuracy, especially after applying SMOTE and TF-IDF. Random Forest performed the best, while Logistic Regression served as a solid baseline. The results confirm that early engagement signals and textual features are useful predictors, but further improvements could be achieved with larger datasets and more advanced NLP techniques.

Chapter: 6 Conclusion

6.1 Summary of Work

This project aimed to predict the virality of YouTube Shorts using early metadata and title-based features. The system was designed to classify whether a video would surpass 100,000 views within 48 hours of upload — a practical threshold commonly associated with trending status. A structured pipeline was implemented, beginning with data collection via the YouTube Data API, followed by feature engineering, class balancing using SMOTE, and training three different classification models: Logistic Regression, Random Forest, and XGBoost.

The focus was on using lightweight, interpretable features to allow real-time implementation and ethical modelling. The evaluation phase demonstrated that it is feasible to forecast short-form content virality using basic engagement signals and metadata, without relying on full video content or complex deep learning architectures.

6.2 Key Findings

Among the models tested, XGBoost achieved the highest predictive performance with an accuracy of ~88%, outperforming both Random Forest and Logistic Regression. The ROC-AUC scores and F1-scores further confirmed its superior ability to handle imbalanced data and detect trending videos accurately.

Feature importance analysis revealed that title_length was the most influential variable, followed by engagement ratios (comment_ratio, like_ratio) and views_per_hour. These findings suggest that both content metadata and early behavioural signals are critical for determining virality on fast-moving platforms like YouTube Shorts.

6.3 Limitations

A significant limitation of this project lies in the size of the dataset collected through the YouTube Data API. Although the scraping script was configured to retrieve up to 3,000 videos, the actual number of videos collected remained around 500. This is due to a known constraint of the YouTube API, which restricts the maximum number of retrievable items per query despite the higher maximum result setting. As a result, the dataset may not be fully representative of the broader variety and diversity of trending content across different genres and topics. This limitation affects the generalisability of the findings and the robustness of the model's predictions.

6.4 Future Works

To overcome the dataset size constraint imposed by the YouTube API, future work should explore the use of multiple, more specific query terms such as #shorts news, #shorts gaming, or #shorts music, rather than relying solely on the general term #shorts. This approach would allow the collection of a larger and more diverse dataset spanning multiple categories, potentially improving the accuracy and reliability of the trend prediction models. Furthermore, future studies could examine time-based sampling (e.g., different days or time windows) or incorporate additional sources beyond the YouTube API. Increasing the dataset size and diversity would provide a stronger foundation for model training and enable exploration of more advanced techniques, such as deep learning models like CNNs, should the dataset scale justify their complexity.

6.5 Final Remarks

Overall, this project successfully demonstrated that trend emergence can be predicted using simple, interpretable models applied to YouTube Shorts metadata. While there are limitations in terms of dataset size and label definition, the results indicate strong potential for future applications in content forecasting and digital strategy planning.

Chapter 7: References

Szabo, G. and Huberman, B.A., 2010. Predicting the popularity of online content. *Communications of the ACM*, *53*(8), pp.80-88.

Bandari, R., Asur, S. and Huberman, B., 2012. The pulse of news in social media: Forecasting popularity. In *Proceedings of the International AAAI Conference on Web and Social Media* (Vol. 6, No. 1, pp. 26-33).

He, H. & Garcia, E.A. 2009, "Learning from Imbalanced Data", *IEEE transactions on knowledge and data engineering,* vol. 21, no. 9, pp. 1263-1284.

Pinto, H., Almeida, J.M. & Gonçalves, M.A. 2013, "Using early view patterns to predict the popularity of YouTube videos", ACM, New York, NY, USA, pp. 365.

Cheng, J., Adamic, L. and Dow, P.A., 2008. *Can cascades be predicted?* In Proceedings of the 19th ACM conference on Hypertext and hypermedia. New York: ACM. Available at: https://dl.acm.org/doi/10.1145/1401890.1401942 [Accessed 30 July 2025].

Ma, J., Gao, W., Wei, Z., Lu, Y. and Wong, K.F., 2015. *Detecting rumours from microblogs with recurrent neural networks.* In Proceedings of the 25th International Conference on World Wide Web. New York: ACM. Available at: https://dl.acm.org/doi/10.1145/2806416.2806607 [Accessed 30 July 2025].

Khosla, A., Das Sarma, A. & Hamid, R. 2014, "What makes an image popular?", ACM, New York, NY, USA, pp. 867. [Accessed on 31 July 2025].