Final Report: Best Time and Best Trend to Post (TikTok and Instagram)

Date: December 9, 2025
Team: Jesse Graham (jg1801), Orland Geronimo (ogg9)
Repository: peak-content

1. Project Definition (Problem and Strategy)

Problem

This project focuses on two major creator needs: identifying the best times to post and determining which trends, such as hashtags and audio, actually help short-form videos gain traction on TikTok and Instagram. These platforms rarely explain their analytics clearly, and most online “best time” advice is vague or not data-driven. To address this, we created a complete pipeline that collects public post data, cleans it, engineers timing and trend features, trains a model to estimate virality, and presents the results through both a dashboard and a prediction tool.

Strategy

Our approach emphasized clear and actionable suggestions for creators, such as specific posting windows and recognizable trend signals. We prioritized transparency by keeping all code open, documenting how features were built, and explaining every modeling decision. The pipeline is also designed so it can be refreshed daily when new data is available.

Course Alignment

This project directly connects to topics from the course, including relational schema design, ETL workflows, data cleaning, feature scaling and encoding, cross-validation, overfitting detection, and visualization using Streamlit and matplotlib.

2. Novelty and Importance

Short-form platforms are extremely competitive, and creators often rely on unreliable or overly general advice. Many existing analytics tools are closed-off and do not show how their predictions are created. Our system is entirely open and reproducible, including processed parquet files, trend summaries, and full documentation of modeling decisions such as the switch from Random Forest to Logistic Regression after identifying overfitting.

Trend detection is also transparent in our pipeline. Instead of relying on hidden scoring systems, we compare recent hashtag and audio activity to long-term activity, making true spikes easy to identify. While research exists on content timing, few solutions provide creators with an accessible, reproducible, end-to-end tool, which is what this project aims to deliver.

3. Data, ETL, and Feature Engineering

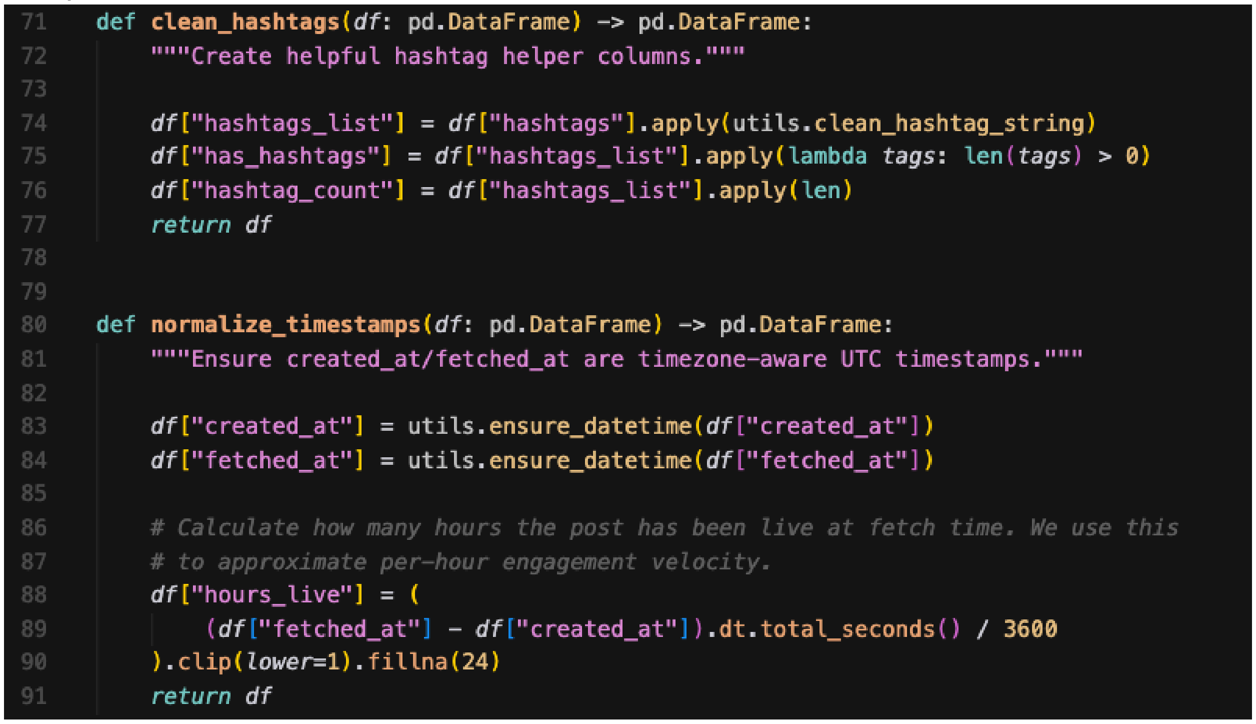
3.1 Data Source

The dataset contains 7,225 TikTok posts with fields such as video ID, captions, hashtags, audio information, engagement metrics, timestamps, and URLs. We removed duplicates and kept real viral outliers since they represent authentic behavior.

3.2 ETL

The ETL workflow standardizes timestamps, cleans hashtag strings, converts numeric fields, removes invalid IDs, and calculates both hours_live and engagement-per-hour metrics. We also ensured that missing or zero values could not distort per-hour rates.

Sample of ETL Code:



```python
def clean_hashtags(df: pd.DataFrame) -> pd.DataFrame:
    """Create helpful hashtag helper columns."""

    df["hashtags_list"] = df["hashtags"].apply(utils.clean_hashtag_string)
    df["has_hashtags"] = df["hashtags_list"].apply(lambda tags: len(tags) > 0)
    df["hashtag_count"] = df["hashtags_list"].apply(len)
    return df


def normalize_timestamps(df: pd.DataFrame) -> pd.DataFrame:
    """Ensure created_at/fetched_at are timezone-aware UTC timestamps."""

    df["created_at"] = utils.ensure_datetime(df["created_at"])
    df["fetched_at"] = utils.ensure_datetime(df["fetched_at"])

    # Calculate how many hours the post has been live at fetch time. We use this
    # to approximate per-hour engagement velocity.
    df["hours_live"] = (
        (df["fetched_at"] - df["created_at"]).dt.total_seconds() / 3600
    ).clip(lower=1).fillna(24)
    return df
```

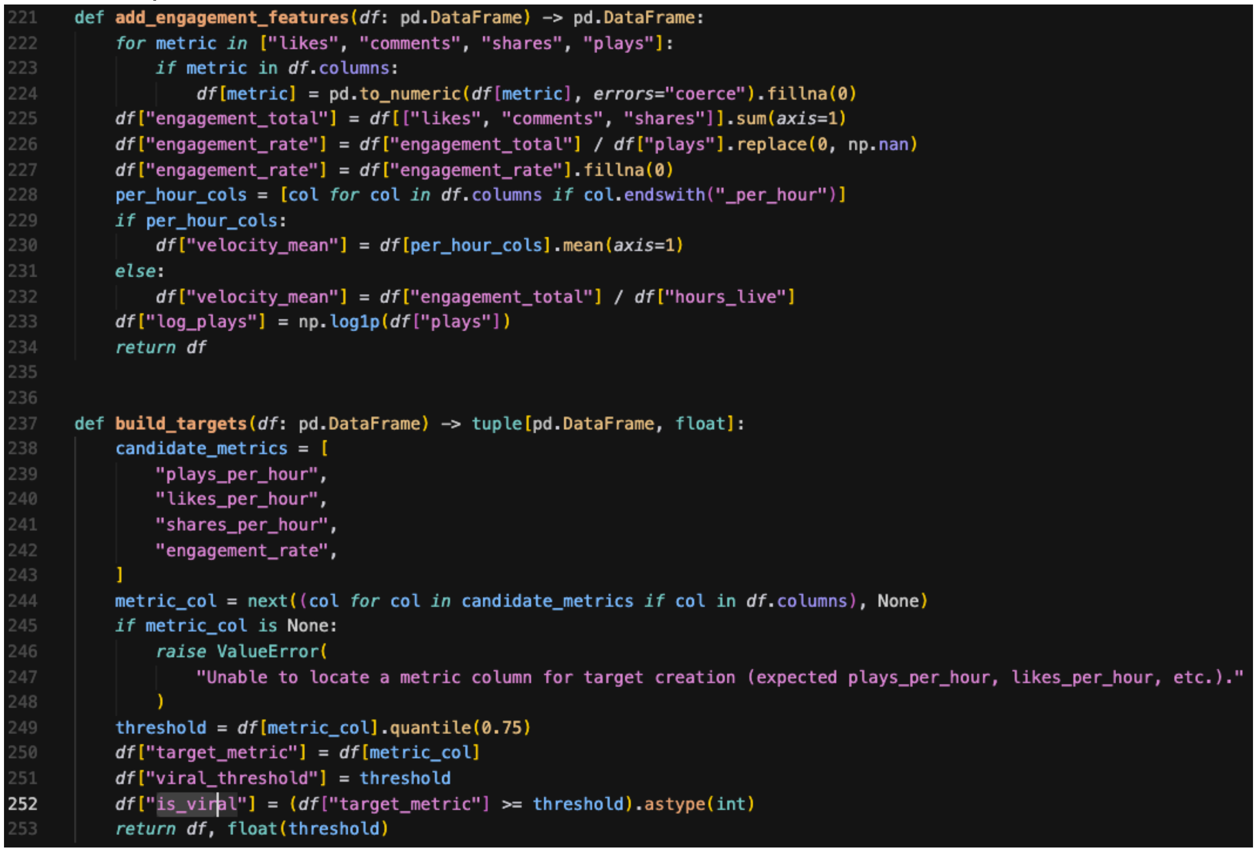
To run this step, the user should enter the following command in the terminal:

```
python3 -m src.etl.clean_tiktok --input data/raw/tiktok_merged_data_deduplicated.csv --output data/processed/posts.parquet
```

3.3 Feature Engineering

Feature engineering includes timing features such as posting hour and weekday, caption-length measures, trend spike scores for hashtags and audio, engagement velocity, and log-scaled metrics for heavy-tailed distributions. The viral label represents the top 25 percent of plays-per-hour to create a more balanced target.

Code Sample:

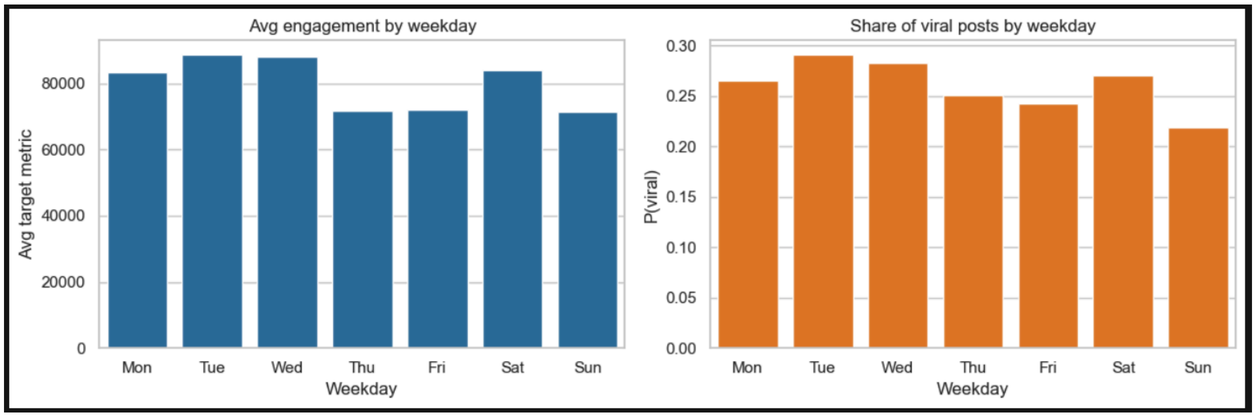


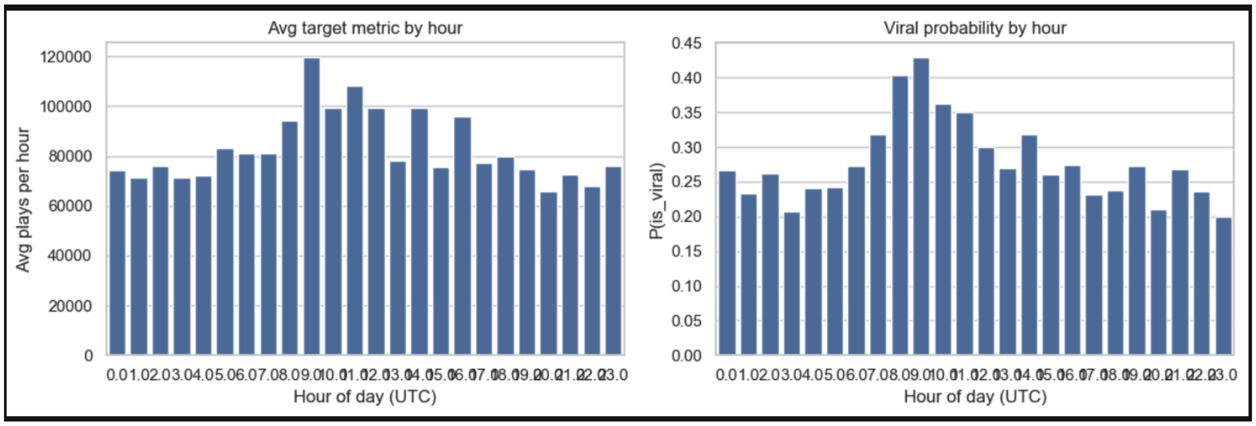
```python
def add_engagement_features(df: pd.DataFrame) -> pd.DataFrame:
    for metric in ["likes", "comments", "shares", "plays"]:
        if metric in df.columns:
            df[metric] = pd.to_numeric(df[metric], errors="coerce").fillna(0)
    df["engagement_total"] = df[["likes", "comments", "shares"]].sum(axis=1)
    df["engagement_rate"] = df["engagement_total"] / df["plays"].replace(0, np.nan)
    df["engagement_rate"] = df["engagement_rate"].fillna(0)
    per_hour_cols = [col for col in df.columns if col.endswith("_per_hour")]
    if per_hour_cols:
        df["velocity_mean"] = df[per_hour_cols].mean(axis=1)
    else:
        df["velocity_mean"] = df["engagement_total"] / df["hours_live"]
    df["log_plays"] = np.log1p(df["plays"])
    return df


def build_targets(df: pd.DataFrame) -> tuple[pd.DataFrame, float]:
    candidate_metrics = [
        "plays_per_hour",
        "likes_per_hour",
        "shares_per_hour",
        "engagement_rate",
    ]
    metric_col = next((col for col in candidate_metrics if col in df.columns), None)
    if metric_col is None:
        raise ValueError(
            "Unable to locate a metric column for target creation (expected plays_per_hour, likes_per_hour, etc.)."
        )
    threshold = df[metric_col].quantile(0.75)
    df["target_metric"] = df[metric_col]
    df["viral_threshold"] = threshold
    df["is_viral"] = (df["target_metric"] >= threshold).astype(int)
    return df, float(threshold)
```

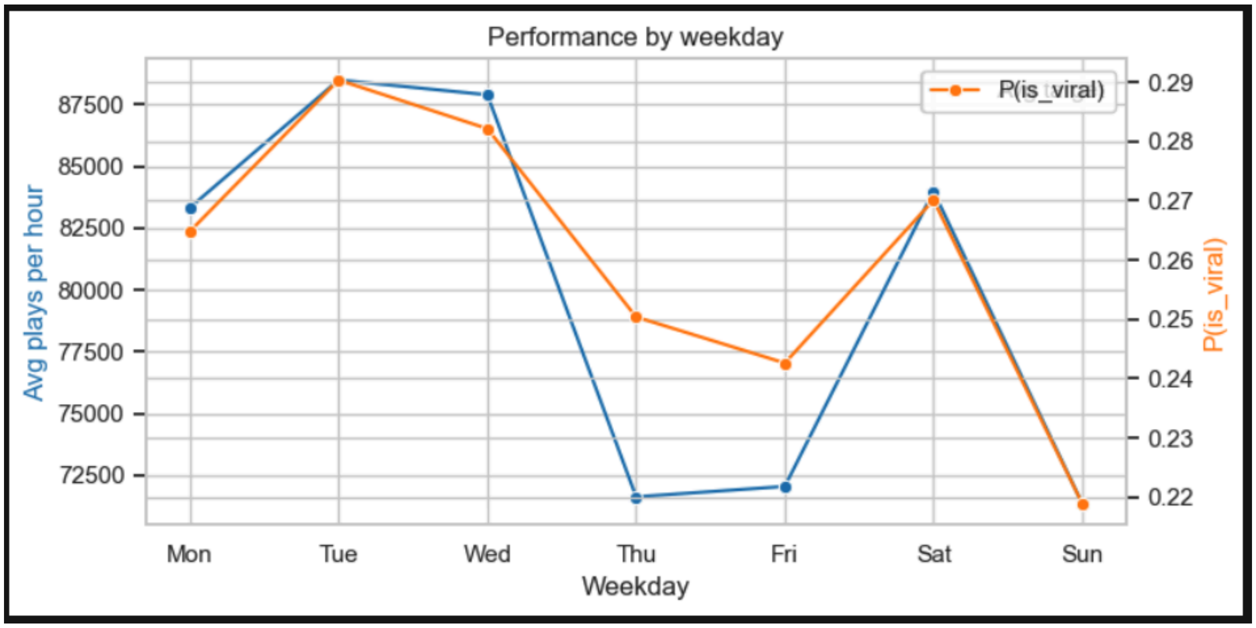
To build features, the user should run:
python3 -m src.features.build_features --input data/processed/posts.parquet --output data/features/training_set.parquet --trend-summary reports/trend_metrics.json

3.4 Exploratory Data Analysis

Exploratory analysis shows that posting between 9 and 11 UTC generally performs best, and Tuesday and Wednesday outperform weekends. The EDA notebook also highlights emerging hashtags and audio that correlate with stronger engagement.

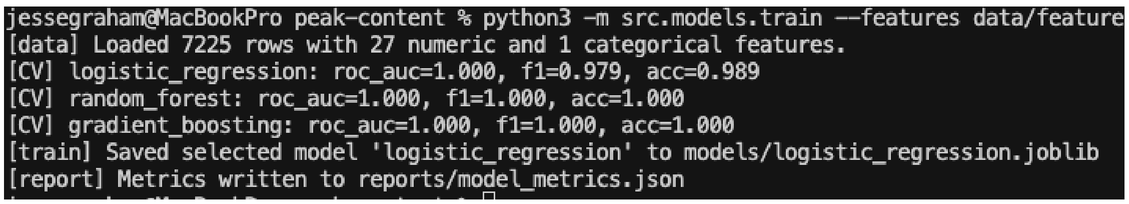






4. Modeling, Evaluation, and Prediction

4.1 Models Tested



```
jessegraham@MacBookPro peak-content % python3 -m src.models.train --features data/feature
[data] Loaded 7225 rows with 27 numeric and 1 categorical features.
[CV] logistic_regression: roc_auc=1.000, f1=0.979, acc=0.989
[CV] random_forest: roc_auc=1.000, f1=1.000, acc=1.000
[CV] gradient_boosting: roc_auc=1.000, f1=1.000, acc=1.000
[train] Saved selected model 'logistic_regression' to models/logistic_regression.joblib
[report] Metrics written to reports/model_metrics.json
```

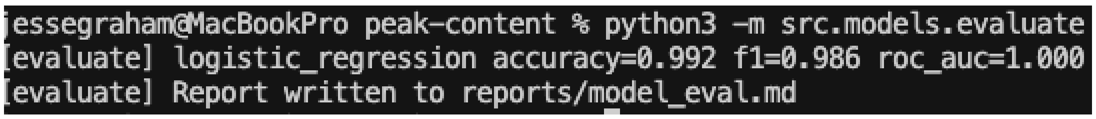
We evaluated Logistic Regression, Random Forest, and Gradient Boosting using consistent preprocessing. Stratified 5-fold cross-validation measured ROC-AUC, F1, and accuracy.

4.2 Overfitting Pivot

Random Forest and Gradient Boosting both achieved perfect scores, which is unrealistic and signals overfitting. Logistic Regression was chosen as the final model because it generalizes better to unseen data.

4.3 Evaluation

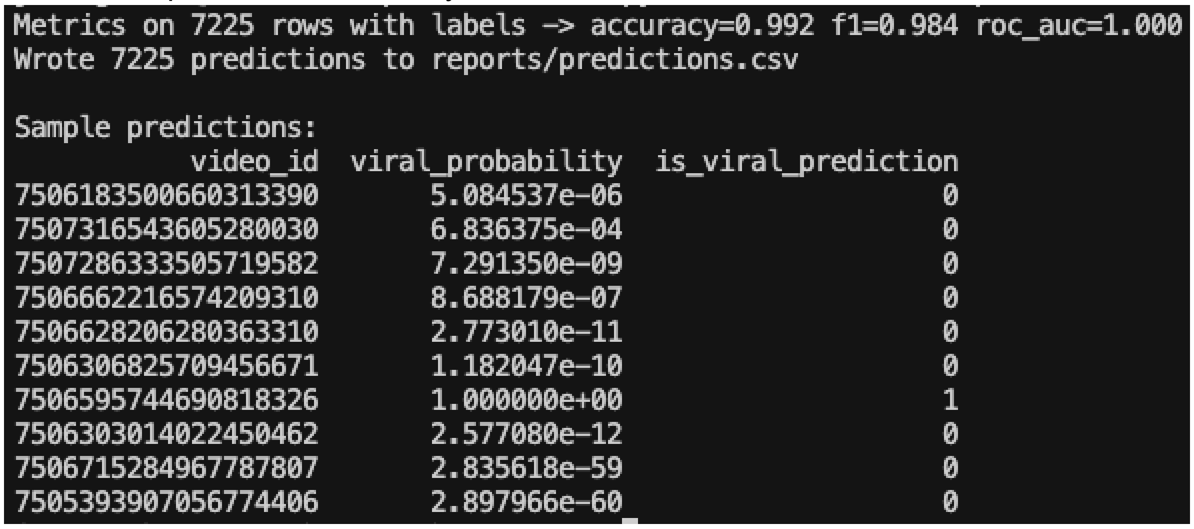
We generated accuracy scores, F1 values, ROC-AUC, confusion matrices, and ROC curves. These results reinforce the need for time-segmented validation in future versions.



```
jessegraham@MacBookPro peak-content % python3 -m src.models.evaluate
[evaluate] logistic_regression accuracy=0.992 f1=0.986 roc_auc=1.000
[evaluate] Report written to reports/model_eval.md
```

4.4 Prediction Tool

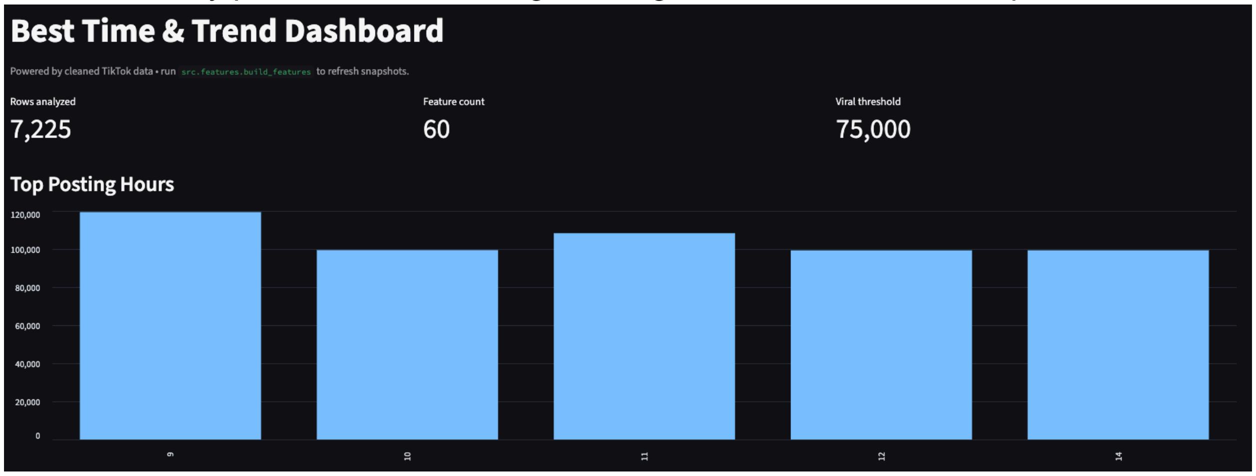
The prediction tool loads a trained model and outputs both viral probabilities and predicted labels. If true labels are included in the input file, batch-level evaluation metrics are printed automatically.



```
Metrics on 7225 rows with labels -> accuracy=0.992 f1=0.984 roc_auc=1.000
Wrote 7225 predictions to reports/predictions.csv

Sample predictions:
           video_id  viral_probability  is_viral_prediction
7506183500660313390       5.084537e-06                    0
7507316543605280030       6.836375e-04                    0
7507286333505719582       7.291350e-09                    0
7506662216574209310       8.688179e-07                    0
7506628206280363310       2.773010e-11                    0
7506306825709456671       1.182047e-10                    0
7506595744690818326       1.000000e+00                    1
7506303014022450462       2.577080e-12                    0
7506715284967787807       2.835618e-59                    0
7505393907056774406       2.897966e-60                    0
```

4.5 Dashboard

The dashboard, which runs in both CLI and Streamlit modes, displays best posting hours, weekday performance, trending hashtags and audio, and model predictions.



Best Time & Trend Dashboard

Powered by cleaned TikTok data • run `src.features.build_features` to refresh snapshots.

Rows analyzed: 7,225

Feature count: 60

Viral threshold: 75,000

Top Posting Hours

5. Experiments, Failures, and Pivots

We replaced the original hashtag parser with a regex-based version after noticing it broke on certain formats. Trend detection originally relied only on global counts, so we redesigned it using recent-to-global ratios to highlight emerging trends. The viral label was expanded from the top 10 percent to the top 25 percent to reduce noise. We also pivoted away from overfitted models and switched to Logistic Regression. Several practical issues, such as incorrect notebook paths and cache errors, were also resolved.

6. Results and Findings

6.1 Best Time to Post

Posting between 9 and 11 UTC consistently delivers the highest engagement. Mondays and Tuesdays outperform other weekdays.

6.2 Trend Signals

Recent-versus-global ratios surfaced trending hashtags and audio effectively. Many viral posts use original sound, although specific tracks occasionally rise sharply.

6.3 Model Performance

Logistic Regression offered strong performance without overfitting and is easy to interpret for creators.

6.4 Prediction Outputs

The system returns predicted labels and viral probabilities. When real labels are included, accuracy, F1, and ROC-AUC are also provided.

7. Advantages and Limitations

Advantages

The pipeline is fully open and reproducible. Features directly match creator needs, the architecture supports daily refresh, and both the dashboard and CLI tools are implemented.

Limitations

The dataset only represents one snapshot from TikTok. There is no personalization for individual creators, plays-per-hour is an imperfect metric, cross-platform validation has not been performed, and time-based validation still needs to be added.

8. Changes After Proposal and Oral Exam

Major updates included adopting Logistic Regression, improving hashtag parsing, refining trend detection logic, and resolving path and cache issues. The overall objective of the project remained consistent.

9. How to Run

To run the entire system from start to finish, each step must be executed in sequence.

To clean the raw data, the user should run:

```
python3 -m src.etl.clean_tiktok --input
data/raw/tiktok_merged_data_deduplicated.csv --output
data/processed/posts.parquet
```

To build the feature set, the user should run:

```
python3 -m src.features.build_features --input data/processed/posts.parquet
--output data/features/training_set.parquet --trend-summary
reports/trend_metrics.json
```

To train the model, the user should run:

```
python3 -m src.models.train --features data/features/training_set.parquet
--model-dir models --report reports/model_metrics.json
```

To evaluate the model, the user should run:

```
MPLCONFIGDIR=.matplotlib python3 -m src.models.evaluate --features
data/features/training_set.parquet --metrics reports/model_metrics.json --report
reports/model_eval.md --figures-dir reports/figures
```

To generate predictions, the user should run:

```
python3 -m src.models.predict --model models/random_forest.joblib --features data/features/training_set.parquet --output reports/predictions.csv
```

To launch the dashboard in CLI mode, the user should run:

```
python3 -m src.visualization.dashboard --mode cli
```

To launch the Streamlit version of the dashboard, the user should run:

```
streamlit run src/visualization/dashboard.py -- --mode streamlit
```

To explore the data visually, the user should open the notebook titled **eda_best_time.ipynb**.

10. Future Work

Future improvements include collecting larger and time-separated datasets, adding personalized creator recommendations, incorporating SHAP explanations, calibrating probability outputs, deploying a public dashboard with automated refresh, containerizing the environment, and experimenting with time-series models to track trend drift over time.

11. Conclusion

This project resulted in a complete end-to-end system that analyzes TikTok and Instagram posting behavior. It handles data ingestion, cleaning, feature engineering, modeling, evaluation, and visualization in one reproducible workflow. Throughout development, we documented several important pivots such as redefining the viral label, improving trend detection, and replacing overfitted models. Logistic Regression currently provides the most stable and interpretable results. With more data and stronger time-based validation, this system can evolve into a dependable, creator-friendly posting assistant