40-Hour YouTube Analytics Project Workflow

Max Giesinger & Wincent Weiss Analysis

WEEK 1: Setup & Data Collection (10-12 hours)

Day 1-2: Environment Setup (3-4 hours)

Hour 1-2: Project initialization

```bash
# Create folder structure
mkdir -p data/{raw,processed} scripts models/{staging,marts} notebooks outputs

# Create files
touch .env .gitignore README.md requirements.txt
touch scripts/{00_init_database.py,01_fetch_data.py,02_load_to_duckdb.py}
```

Checklist:

- [ ] Install Python packages: `pip install duckdb dbt-duckdb google-api-python-client python-dotenv pandas tqdm matplotlib seaborn plotly`
- [ ] Get YouTube API key from Google Cloud Console
- [ ] Find channel IDs for Max Giesinger & Wincent Weiss
- [ ] Create `.env` file with API key
- [ ] Initialize git repository

Hour 3-4: Database setup

```bash
# Run initialization
python scripts/00_init_database.py
```

Deliverable: Working database with empty tables

Day 3-4: Initial Data Collection (4-5 hours)

Hour 5-7: Build & test API crawler

- Modify `01_fetch_data.py` with correct channel IDs
- Test API connection

- Fetch first batch of data:
  - 50 videos per artist (100 total)
  - 50 comments per video (5,000 total)
  - Channel metadata (2 rows)

Hour 8-9: Load to DuckDB

```bash
python scripts/02_load_to_duckdb.py
```

Validate data:

```python
import duckdb
conn = duckdb.connect('data/youtube_analytics.duckdb')
print(conn.execute("SELECT COUNT(*) FROM raw.videos").fetchone())
print(conn.execute("SELECT COUNT(*) FROM raw.comments").fetchone())
```

Deliverable: ~5,100 rows loaded into DuckDB

Day 5-7: Daily Tracking Setup (3 hours)

Hour 10-11: Modify scripts for daily collection

Update `01_fetch_data.py` to:

- Check existing video IDs
- Only fetch NEW videos (incremental)
- Re-fetch statistics for existing videos (track growth)
- Add `snapshot_date` column

Hour 12: Set up automation

Option A: Manual (simpler)

- Run script daily for 7-14 days
- Set phone reminder

Option B: GitHub Actions (better)

```yaml
```

```
# .github/workflows/daily_fetch.yml
name: Daily Data Collection
on:
  schedule:
    - cron: '0 12 * * *'  # Daily at noon
  workflow_dispatch:
```

Decision point:

- If Week 1-2 overlaps with project start, use Option A
- Set up GitHub Actions in Week 2 for remainder

Deliverable: Automated daily collection running

WEEK 2: Cleaning & Feature Engineering (12-14 hours)

Day 8-10: dbt Setup & Staging Models (6-7 hours)

Hour 13-14: dbt configuration

Create `profiles.yml`:

```yaml
youtube_analytics:
  target: dev
  outputs:
    dev:
      type: duckdb
      path: data/youtube_analytics.duckdb
```

Create `dbt_project.yml`:

```yaml
name: 'youtube_analytics'
models:
  youtube_analytics:
    staging:
      +materialized: view
    marts:
      +materialized: table
```

Hour 15-17: Staging models (Bronze layer)

`models/staging/stg_videos.sql`:

```sql
SELECT
    video_id,
    channel_id,
    channel_name,
    title,
    CAST(published_at AS TIMESTAMP) as published_at,
    CAST(view_count AS INTEGER) as view_count,
    CAST(like_count AS INTEGER) as like_count,
    CAST(comment_count AS INTEGER) as comment_count,
    duration,
    tags,
    snapshot_date
FROM {{ source('raw', 'videos') }}
WHERE video_id IS NOT NULL
```

`models/staging/stg_comments.sql`:

```sql
SELECT
    comment_id,
    video_id,
    author,
    text,
    CAST(like_count AS INTEGER) as like_count,
    CAST(published_at AS TIMESTAMP) as published_at
FROM {{ source('raw', 'comments') }}
WHERE comment_id IS NOT NULL
```

Hour 18-19: Run & test

```bash
dbt run
dbt test
```

Deliverable: Clean staging tables

Day 11-14: Feature Engineering (6-7 hours)

Hour 20-22: Intermediate models (Silver layer)

models/intermediate/int_video_features.sql:

```
sql
```

```
WITH video_data AS (
  SELECT * FROM {{ ref('stg_videos') }}
)

SELECT
  *,
  -- Time features
  DATE_PART('dow', published_at) as day_of_week,
  DATE_PART('hour', published_at) as hour_of_day,
  CASE
    WHEN DATE_PART('dow', published_at) IN (0,6) THEN 'Weekend'
    ELSE 'Weekday'
  END as weekend_flag,

  -- Text features
  LENGTH(title) as title_length,
  ARRAY_LENGTH(STRING_SPLIT(tags, '|')) as tag_count,
  CASE
    WHEN LOWER(title) LIKE '%feat%'
     OR LOWER(title) LIKE '%ft.%'
     OR LOWER(title) LIKE '%with%'
    THEN 'Collaboration'
    ELSE 'Solo'
  END as collaboration_type,

  -- Content type detection
  CASE
    WHEN LOWER(title) LIKE '%official video%' THEN 'Music Video'
    WHEN LOWER(title) LIKE '%live%' THEN 'Live Performance'
    WHEN LOWER(title) LIKE '%acoustic%' THEN 'Acoustic'
    WHEN LOWER(title) LIKE '%behind%' THEN 'Behind The Scenes'
    ELSE 'Other'
  END as content_type,

  -- Engagement metrics
  ROUND(like_count * 100.0 / NULLIF(view_count, 0), 2) as like_rate,
  ROUND(comment_count * 100.0 / NULLIF(view_count, 0), 4) as comment_rate,
  ROUND((like_count + comment_count) * 100.0 / NULLIF(view_count, 0), 2) as engagement_rate,

  -- Video age
  DATE_DIFF('day', published_at, CURRENT_DATE) as days_since_published

FROM video_data
```

Hour 23-24: Comment features

models/intermediate/int_comment_features.sql:

```sql
WITH comments AS (
  SELECT * FROM {{ ref('stg_comments') }}
)

SELECT
  comment_id,
  video_id,
  LENGTH(text) as comment_length,
  ARRAY_LENGTH(REGEXP_EXTRACT_ALL(text, '[!]')) as exclamation_count,
  ARRAY_LENGTH(REGEXP_EXTRACT_ALL(text, '[?]')) as question_count,
  CASE
    WHEN LOWER(text) LIKE '%love%' OR LOWER(text) LIKE '%amazing%'
    THEN 'Positive'
    WHEN LOWER(text) LIKE '%hate%' OR LOWER(text) LIKE '%bad%'
    THEN 'Negative'
    ELSE 'Neutral'
  END as sentiment_simple
FROM comments
```

Hour 25-26: Run & validate

```bash
dbt run
dbt test --select intermediate
```

Deliverable: Enriched feature tables

WEEK 3: Analysis & Visualization (12-14 hours)

Day 15-17: Analytical Models (5-6 hours)

Hour 27-29: Marts layer (Gold)

models/marts/fct_video_performance.sql:

```sql
```

```
SELECT
    v.*,
    -- Add ranking
    ROW_NUMBER() OVER (
        PARTITION BY channel_name
        ORDER BY engagement_rate DESC
    ) as engagement_rank,

    -- Performance buckets
    CASE
        WHEN view_count > 1000000 THEN 'Viral'
        WHEN view_count > 500000 THEN 'High'
        WHEN view_count > 100000 THEN 'Medium'
        ELSE 'Low'
    END as performance_tier

FROM {{ ref('int_video_features') }} v
```

`models/marts/fct_artist_comparison.sql`:

```sql
SELECT
    channel_name as artist,
    COUNT(*) as total_videos,
    SUM(view_count) as total_views,
    AVG(view_count) as avg_views_per_video,
    AVG(engagement_rate) as avg_engagement_rate,
    AVG(like_rate) as avg_like_rate,
    MAX(view_count) as best_video_views,
    MIN(view_count) as worst_video_views
FROM {{ ref('int_video_features') }}
GROUP BY channel_name
```

Hour 30-32: Answer research questions

Create `scripts/03_analyze.py`:

```python
```

```python
import duckdb
import pandas as pd

conn = duckdb.connect('data/youtube_analytics.duckdb')

# DV1: Video characteristics vs engagement
q1 = conn.execute("""
    SELECT
        content_type,
        collaboration_type,
        COUNT(*) as video_count,
        AVG(engagement_rate) as avg_engagement,
        AVG(view_count) as avg_views
    FROM marts.fct_video_performance
    GROUP BY content_type, collaboration_type
    ORDER BY avg_engagement DESC
""").df()

# DV2: Upload timing
q2 = conn.execute("""
    SELECT
        day_of_week,
        hour_of_day,
        COUNT(*) as videos,
        AVG(engagement_rate) as avg_engagement
    FROM marts.fct_video_performance
    GROUP BY day_of_week, hour_of_day
    HAVING COUNT(*) >= 3
    ORDER BY avg_engagement DESC
""").df()

# DV3: Artist comparison
q3 = conn.execute("""
    SELECT * FROM marts.fct_artist_comparison
""").df()

# DV4: Comment sentiment
q4 = conn.execute("""
    SELECT
        v.channel_name,
        c.sentiment_simple,
        COUNT(*) as comment_count,
        AVG(c.comment_length) as avg_comment_length
    FROM marts.fct_video_performance v
    JOIN {{ ref('int_comment_features') }} c
        ON v.video_id = c.video_id
    GROUP BY v.channel_name, c.sentiment_simple
""").df()

# Save results
q1.to_csv('outputs/q1_characteristics.csv', index=False)
q2.to_csv('outputs/q2_timing.csv', index=False)
q3.to_csv('outputs/q3_comparison.csv', index=False)
q4.to_csv('outputs/q4_sentiment.csv', index=False)

conn.close()
```

Deliverable: Analysis results in CSV

Day 18-21: Visualization (7-8 hours)

Hour 33-36: Create visualizations

Create `notebooks/analysis.ipynb`:

python

```
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Set style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# Load data
q1 = pd.read_csv('outputs/q1_characteristics.csv')
q2 = pd.read_csv('outputs/q2_timing.csv')
q3 = pd.read_csv('outputs/q3_comparison.csv')

# Chart 1: Engagement by content type
fig, ax = plt.subplots()
sns.barplot(data=q1, x='content_type', y='avg_engagement',
            hue='collaboration_type', ax=ax)
plt.title('Engagement Rate by Content Type & Collaboration')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('outputs/fig1_engagement_by_type.png', dpi=300)

# Chart 2: Heatmap of upload timing
pivot = q2.pivot(index='hour_of_day',
                 columns='day_of_week',
                 values='avg_engagement')
sns.heatmap(pivot, annot=True, fmt='.2f', cmap='YlOrRd')
plt.title('Engagement Rate by Upload Day & Hour')
plt.savefig('outputs/fig2_timing_heatmap.png', dpi=300)

# Chart 3: Artist comparison
fig = px.bar(q3, x='artist', y=['avg_engagement_rate', 'avg_like_rate'],
             title='Artist Performance Comparison',
             barmode='group')
fig.write_html('outputs/fig3_artist_comparison.html')

# Chart 4: Time series (if daily tracking)
# ... add if you have snapshot data
```

Hour 37-40: Key insights document

Create `outputs/insights.md`:

- Summary of findings for each DV
- Actionable recommendations

- Data quality notes
- Limitations

Deliverable: 5-7 professional charts + insights doc

WEEK 4: Dashboard & Storytelling (Week 4 can be light)

Day 22-24: Documentation (4-5 hours)

Hour 41-43: README.md

Structure:

```markdown
# YouTube Analytics: Max Giesinger vs Wincent Weiss

## Project Overview
[Your persona description]

## Key Findings
- Finding 1...
- Finding 2...

## Tech Stack
- Python, DuckDB, dbt, pandas...

## Data Pipeline
[Architecture diagram]

## How to Run
[Step-by-step instructions]

## Project Structure
[Folder tree]
```

Hour 44-45: Code cleanup

- Add docstrings
- Remove unused code
- Format with black
- Update requirements.txt

Day 25-28: Presentation (Optional polish)

If you have time:

- Create 5-slide presentation (PowerPoint/Google Slides)
- Record 3-min video walkthrough
- Create simple dashboard with Plotly Dash or Streamlit

Deliverable: Professional GitHub repository

Daily Tracking Impact

By running daily collection for 14-21 days, you'll add:

- **Time-series analysis** capability
- **Growth tracking** (views/likes over time)
- **Trending detection** (which videos gaining momentum)
- **Demonstrates production mindset** (scheduled data pipelines)

This adds 100-200 rows per day = 1,400-4,200 additional rows by project end.

Final Checklist

Week 1:

- [ ] Database initialized
- [ ] API working
- [ ] Initial data loaded (~5k rows)
- [ ] Daily collection running

Week 2:

- [ ] dbt configured
- [ ] Staging models done
- [ ] Features engineered
- [ ] All tests passing

Week 3:

- [ ] Analytical queries written
- [ ] 4 DVs answered
- [ ] 5-7 charts created
- [ ] Insights documented

Week 4:

- [ ] README complete
- [ ] Code documented
- [ ] GitHub repository published
- [ ] Portfolio piece ready

Time Budget Reality Check

Activity	Planned	Likely Actual
Setup	3-4h	5-6h (troubleshooting)
Data collection	4-5h	4h (once working)
dbt modeling	8-10h	10-12h (learning curve)
Analysis	5-6h	6-7h
Visualization	7-8h	8-9h
Documentation	4-5h	5-6h
TOTAL	**40h**	**45-50h (realistic)**

Build in buffer time. First dbt project always takes longer than expected.