

# Project\_the\_good\_place

January 7, 2026

## 1 The Good Place – Exploratory Data Analysis

### 2 Introduction

In this notebook presents an exploratory data analysis (EDA) project based on episode-level information from the television series **The Good Place** bold text, using two datasets from IMDB. The goal is to combine, clean, and explore the data to find useful insights about viewer trends, ratings, and episode structure.

The main goal is to **combine, clean, and explore** the data to uncover useful insights about: - Viewer trends - IMDb ratings - Episode structure - Contributions by writers and directors

#### 2.1 Datasets Used

This project works with two CSV files:

##### 2.1.1 1. the\_good\_place\_imdb.csv

##### 2.1.2 which contains of :

- **season**: Season number
- **episode\_num**: Episode number within the season
- **title**: Episode title (from IMDb)
- **original\_air\_date**: Air date (may differ in formatting)
- **imdb\_rating**: IMDb user rating (1–10)
- **total\_votes**: Number of user votes on IMDb
- **desc**: Short episode description

##### 2.1.3 2. the\_good\_place\_episodes.csv

This dataset includes episode metadata such as titles, creators, and US viewership:

- **season**: Season number (initially stored as a float)
- **episode\_num\_in\_season**: Episode number within the season

- `episode_num_overall`: Episode number across the entire series
- `title`: Episode title (from production data)
- `directed_by`: Name(s) of the episode's director(s)
- `written_by`: Name(s) of the episode's writer(s)
- `original_air_date`: Air date (may differ from IMDb)
- `us_viewers`: Number of viewers in the US (in millions)

## 2.2 Project Objective is:

The goal of this project is to perform a structured and detailed Exploratory Data Analysis (EDA) on episode-level data from *The Good Place*.

By combining data from IMDb and production sources, we want to:

- Understand how the show was received across seasons
- Analyze trends in ratings, votes, and viewership
- Identify top-performing episodes
- Explore the roles of writers and directors
- Handle data inconsistencies and prepare the dataset for analysis

## 2.3 Data location and loading method

This project uses two small CSV datasets related to *The Good Place*.

To keep the repository lightweight and to respect dataset redistribution terms, the raw CSV files are **not committed** to this repository by default. Instead, you should download them from the referenced Kaggle source and place them locally in the folder:

- `data/raw/the_good_place_imdb.csv`
- `data/raw/the_good_place_episodes.csv`

After the files are placed in `data/raw/`, the notebook loads them directly from disk into pandas DataFrames.

### 2.3.1 Files used

- `the_good_place_imdb.csv`: IMDb ratings, votes, titles, air dates, and descriptions
- `the_good_place_episodes.csv`: Episode metadata such as writers, directors, air dates, and US viewership

### 2.3.2 Source

Kaggle notebook reference: *The Good Place Episode Data Analysis* (bcruise)

Note: The Kaggle notebook code is published under the Apache 2.0 license. Dataset redistribution rights depend on the dataset license specified on Kaggle; therefore, raw data files are not included in the repository by default.

## 2.4 First step : preparing the Working Environment by importing the necessary libraries

- Importing necessary Python libraries, then

- Load both datasets and then
- see the data in them and somehow to understand their structure and what to do with them as instructed

```
[ ]: #step 1 importing the necessary libraries
```

```
import pandas as pd          # For data manipulation
import numpy as np           # For visualization
import matplotlib.pyplot as plt # For enhanced plots
import seaborn as sns        # For numerical operations
from pathlib import Path
```

- pandas for data handling
- numpy for numerical support
- matplotlib and seaborn for plotting and data visualization

```
[ ]: DATA_DIR = Path("data/raw")
imdb_path = DATA_DIR / "the_good_place_imdb.csv"
episodes_path = DATA_DIR / "the_good_place_episodes.csv"

missing = [str(p) for p in [imdb_path, episodes_path] if not p.exists()]
if missing:
    raise FileNotFoundError(
        "Missing dataset file(s):\n- "
        + "\n- ".join(missing)
        + "\n\nDownload the CSV files from the referenced Kaggle source and
        ↪place them into:\n"
        + "  data/raw/\n\nSee: data/README_data.md"
    )

df_imdb = pd.read_csv(imdb_path)
df_episodes = pd.read_csv(episodes_path)

df_imdb.head(), df_episodes.head()
```

--2025-05-08 18:15:09--

<https://drive.google.com/uc?export=download&id=1B5UZoBR4Cy0eEoxJveuczZUWxDxjd9nP>  
Resolving drive.google.com (drive.google.com)... 142.250.98.101, 142.250.98.100,  
142.250.98.138, ...

Connecting to drive.google.com (drive.google.com)|142.250.98.101|:443...  
connected.

HTTP request sent, awaiting response... 303 See Other

Location: <https://drive.usercontent.google.com/download?id=1B5UZoBR4Cy0eEoxJveuczZUWxDxjd9nP&export=download> [following]

--2025-05-08 18:15:09-- <https://drive.usercontent.google.com/download?id=1B5UZoBR4Cy0eEoxJveuczZUWxDxjd9nP&export=download>

Resolving drive.usercontent.google.com (drive.usercontent.google.com)...  
74.125.196.132, 2607:f8b0:400c:c36::84

Connecting to drive.usercontent.google.com

```

(drive.usercontent.google.com)|74.125.196.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4260 (4.2K) [application/octet-stream]
Saving to: 'the_good_place_episodes.csv'

the_good_place_epis 100%[=====>]    4.16K  --.-KB/s    in 0s

2025-05-08 18:15:11 (35.3 MB/s) - 'the_good_place_episodes.csv' saved
[4260/4260]

--2025-05-08 18:15:11--
https://drive.google.com/uc?export=download&id=1g50tAjkW_Vvcd60DvKPlJY35r4x3EJ_B
Resolving drive.google.com (drive.google.com)... 142.250.98.101, 142.250.98.100,
142.250.98.138, ...
Connecting to drive.google.com (drive.google.com)|142.250.98.101|:443...
connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1g50tAjkW_Vvcd60DvKPlJY35r4x3EJ_B&export=download [following]
--2025-05-08 18:15:11-- https://drive.usercontent.google.com/download?id=1g50tA
jkW_Vvcd60DvKPlJY35r4x3EJ_B&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)...
74.125.196.132, 2607:f8b0:400c:c36::84
Connecting to drive.usercontent.google.com
(drive.usercontent.google.com)|74.125.196.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8327 (8.1K) [application/octet-stream]
Saving to: 'the_good_place_imdb.csv'

the_good_place_imdb 100%[=====>]    8.13K  --.-KB/s    in 0s

2025-05-08 18:15:14 (63.1 MB/s) - 'the_good_place_imdb.csv' saved [8327/8327]

```

## 2.5 Loading the Datasets

This project uses two CSV files related to *The Good Place*.

For portfolio and licensing safety, the raw CSV files are **not included in this repository by default**.

Download the files from the referenced Kaggle source and place them in:

- data/raw/the\_good\_place\_imdb.csv
- data/raw/the\_good\_place\_episodes.csv

After the files are placed in data/raw/, they are loaded into two **pandas DataFrames** using `pd.read_csv()`:

- **imdb\_df** contains episode information such as title, IMDb rating, air date, and vote count.

- **episodes\_df** contains production-related details such as writers, directors, air dates, and US viewership.

To quickly inspect both datasets, the notebook uses `.head()` to display the first 5 rows.

```
[ ]: # Loading the CSV files
imdb_df = pd.read_csv("the_good_place_imdb.csv")
episodes_df = pd.read_csv("the_good_place_episodes.csv")
```

```
[ ]: # Display the first 5 rows of each dataframe
print("IMDB Data:")
display(imdb_df.head())

print("Episodes Data:")
display(episodes_df.head())
```

IMDB Data:

	season	episode_num	title \
0	1	1	Pilot
1	1	2	Flying
2	1	3	Tahani Al-Jamil
3	1	4	Jason Mendoza
4	1	5	Category 55 Emergency Doomsday Crisis

	original_air_date	imdb_rating	total_votes \
0	2016-09-19	8.0	3687
1	2016-09-19	7.6	3242
2	2016-09-22	8.0	3073
3	2016-09-29	7.9	2934
4	2016-10-06	8.0	2823

	desc
0	Newly-deceased Eleanor Shellstrop is sent to t...
1	Eleanor tries to prove to Chidi that she's wor...
2	Chidi starts giving Eleanor formal lessons in ...
3	Eleanor has a hard time remaining hidden; Mich...
4	Teaching Eleanor about ethics becomes a full-t...

Episodes Data:

	season	episode_num_in_season	episode_num_overall \
0	1.0	1	1
1	1.0	2	2
2	1.0	3	3
3	1.0	4	4
4	1.0	5	5

	title	directed_by \
0	Pilot	Drew Goddard

1		Flying	Michael McDonald
2		Tahani Al-Jamil	Beth McCarthy-Miller
3		Jason Mendoza	Payman Benz
4	Category 55 Emergency Doomsday Crisis		Morgan Sackett

	written_by	original_air_date	us_viewers
0	Michael Schur	2016-09-19	8040000.0
1	Alan Yang	2016-09-19	8040000.0
2	Aisha Muharrar	2016-09-22	5250000.0
3	Joe Mande	2016-09-29	4450000.0
4	Matt Murray	2016-10-06	4970000.0

## 2.6 Findings from the both datasets above :

### 1. Data Type Mismatch in season:

- IMDb: **season** is an **integer** (e.g., 1)
- Episodes dataset: **season** is a **float** (e.g., 1.0)  
*Solution:* Convert **season** in the Episodes dataset to integer for consistency.

### 2. Episode Number Column Naming:

- IMDb uses **episode\_num**
- Episodes dataset uses **episode\_num\_in\_season**  
*Solution:* Standardize the column names before merging (e.g., rename one to match the other).

### 3. Presence of Overall Episode Number:

- Available in Episodes dataset (**episode\_num\_overall**)
- Not present in IMDb dataset  
*Solution:* Keep for reference, but do not use for merging.

### 4. Differences in Title Formatting:

- IMDb titles are often shorter and lowercase
- Episodes dataset titles are longer or more descriptive  
*Solution:* We can standardize text formatting if needed, but this is optional.

This overview show us let us know that why data prepration is necessary and in this and case like this how we handling them. **bold text**

```
[ ]: #season in episodes to integer
episodes_df = episodes_df.dropna(subset=['season'])
episodes_df = episodes_df[~episodes_df['season'].isin([float('inf'),
↳float('-inf')])]
episodes_df['season'] = episodes_df['season'].astype(int)

# 'episode_num_in_season' to 'episode_num' for easier merging
episodes_df.rename(columns={'episode_num_in_season': 'episode_num'},
↳inplace=True)
```

```
[ ]: # to see the result of previous step
print("IMDB Data:")
display(imdb_df.head())

print("Episodes Data:")
display(episodes_df.head())
```

IMDB Data:

	season	episode_num	title \
0	1	1	Pilot
1	1	2	Flying
2	1	3	Tahani Al-Jamil
3	1	4	Jason Mendoza
4	1	5	Category 55 Emergency Doomsday Crisis

	original_air_date	imdb_rating	total_votes \
0	2016-09-19	8.0	3687
1	2016-09-19	7.6	3242
2	2016-09-22	8.0	3073
3	2016-09-29	7.9	2934
4	2016-10-06	8.0	2823

	desc
0	Newly-deceased Eleanor Shellstrop is sent to t...
1	Eleanor tries to prove to Chidi that she's wor...
2	Chidi starts giving Eleanor formal lessons in ...
3	Eleanor has a hard time remaining hidden; Mich...
4	Teaching Eleanor about ethics becomes a full-t...

Episodes Data:

	season	episode_num	episode_num_overall \
0	1	1	1
1	1	2	2
2	1	3	3
3	1	4	4
4	1	5	5

	title	directed_by \
0	Pilot	Drew Goddard
1	Flying	Michael McDonald
2	Tahani Al-Jamil	Beth McCarthy-Miller
3	Jason Mendoza	Payman Benz
4	Category 55 Emergency Doomsday Crisis	Morgan Sackett

	written_by	original_air_date	us_viewers
0	Michael Schur	2016-09-19	8040000.0
1	Alan Yang	2016-09-19	8040000.0

2	Aisha Muharrar	2016-09-22	5250000.0
3	Joe Mande	2016-09-29	4450000.0
4	Matt Murray	2016-10-06	4970000.0

## 2.7 Understanding the Structure of the Data

before we start anything we must analyzing and understanding

1. The columns we have
2. The data type we have
3. Is there any missing values?
4. any duplication
5. is there any overlapping or complementary informations?

## 2.8 Dataset Structure with .info()

the .info() method to examine the basic structure of each dataset

The .info() provides a summary of: - Column names - Number of non-null values - Data types for each column

### 2.8.1 IMDB Dataset:

- have 50 rows and 7 columns
- All fields have values and no missing values
- The fields type are mostly int64, float64, or text

### 2.8.2 Episodes Dataset:

- have 46 rows and 8 columns
- these fields us \_viewers, original\_air\_date, directed\_by, and written\_by have missing value
- The fields type are integers, floats, and text

```
[ ]: #IMDB DATASET INFO
print("IMDB Dataset Info:")
imdb_df.info()

#EPISODES DATASET INFO
print("\nEpisodes Dataset Info:")
episodes_df.info()
```

IMDB Dataset Info:

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 50 entries, 0 to 49

Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
0	season	50 non-null	int64
1	episode_num	50 non-null	int64
2	title	50 non-null	object
3	original_air_date	50 non-null	object



```

4   imdb_rating      50 non-null    float64
5   total_votes      50 non-null    int64
6   desc             50 non-null    object
dtypes: float64(1), int64(3), object(3)
memory usage: 2.9+ KB

```

Episodes Dataset Info:

```
<class 'pandas.core.frame.DataFrame'>
```

Index: 46 entries, 0 to 51

Data columns (total 8 columns):

#	Column	Non-Null Count	Dtype
0	season	46 non-null	int64
1	episode_num	46 non-null	int64
2	episode_num_overall	46 non-null	int64
3	title	46 non-null	object
4	directed_by	46 non-null	object
5	written_by	46 non-null	object
6	original_air_date	46 non-null	object
7	us_viewers	43 non-null	float64

```
dtypes: float64(1), int64(3), object(4)
```

```
memory usage: 3.2+ KB
```

## 2.9 Missing Values Overview

### 2.9.1 IMDb Dataset:

- No missing values in any column.
- All 50 records are fully complete.

### 2.9.2 Episodes Dataset:

- The `us_viewers` column has **3 missing values**.
- All other columns currently appear to be complete.

```
[ ]: print("Missing values/nulls in IMDB :")
      print(imdb_df.isnull().sum())

      print("\nMissing values/null in Episodes :")
      print(episodes_df.isnull().sum())
```

Missing values/nulls in IMDB :

```

season      0
episode_num 0
title       0
original_air_date 0
imdb_rating 0
total_votes 0
desc        0

```

```
dtype: int64
```

```
Missing values/null in Episodes :
```

```
season          0
episode_num     0
episode_num_overall  0
title           0
directed_by     0
written_by      0
original_air_date 0
us_viewers      3
dtype: int64
```

## 2.10 Checking for Duplicate Rows

for this I checked both datasets for duplicate rows using the `.duplicated().sum()` method.

Results: - **IMDb Dataset:** 0 duplicate rows - **Episodes Dataset:** 0 duplicate rows

```
[ ]: print("Duplicate rows in IMDB:", imdb_df.duplicated().sum())
      print("Duplicate rows in Episodes:", episodes_df.duplicated().sum())
```

```
Duplicate rows in IMDB: 0
```

```
Duplicate rows in Episodes: 0
```

## 2.11 Comparing Dataset Columns and Shapes

To see how many column and rows we have - shape and is there any chance of merging or *matching*

### 2.11.1 DataFrame Shapes:

- **IMDb Dataset:** 50 rows  $\times$  7 columns
- **Episodes Dataset:** 46 rows  $\times$  8 columns

The row mismatch shows that not all episodes are represented in both datasets.

```
[ ]: print("IMDB Columns:", imdb_df.columns.tolist())
      print("Episodes Columns:", episodes_df.columns.tolist())

      print("IMDB Shape:", imdb_df.shape)
      print("Episodes Shape:", episodes_df.shape)
```

```
IMDB Columns: ['season', 'episode_num', 'title', 'original_air_date',
               'imdb_rating', 'total_votes', 'desc']
```

```
Episodes Columns: ['season', 'episode_num', 'episode_num_overall', 'title',
                  'directed_by', 'written_by', 'original_air_date', 'us_viewers']
```

```
IMDB Shape: (50, 7)
```

```
Episodes Shape: (46, 8)
```

## 2.12 Merging datasets Common Columns

Just by the look at it as we all know and have experices with movies and series and IMDB database we could say it has shared columns between them like (Season, episode, title, air date)

we just try to merge and see how things go out by the look of the output errors

**An outer merge on the common columns season and episode\_num.**

```
[ ]: # MERGING
merged = pd.merge(imdb_df, episodes_df, on=["season", "episode_num"],
    ↳how="outer", suffixes=('_imdb', '_episodes'), indicator=True)

# Mergeing only season and episode_num
merged = pd.merge(imdb_df, episodes_df, on=["season", "episode_num"],
    ↳how="outer",
    suffixes=('_imdb', '_episodes'), indicator=True)
# Check for mismatch
mismatched_titles = merged[merged['title_imdb'] != merged['title_episodes']]
print("If there is any mismatched titles:")
display(mismatched_titles[['season', 'episode_num', 'title_imdb',
    ↳'title_episodes']])
```

If there is any mismatched titles:

	season	episode_num	title_imdb \
14	2	2	Dance Dance Resolution
15	2	3	Team Cockroach
16	2	4	Existential Crisis
17	2	5	The Trolley Problem
18	2	6	Janet and Michael
19	2	7	Derek
20	2	8	Leap to Faith
21	2	9	Best Self
22	2	10	Rhonda, Diana, Jake, and Trent
23	2	11	The Burrito
24	2	12	Somewhere Else
25	2	13	NaN
27	3	2	The Brainy Bunch
28	3	3	The Snowplow
29	3	4	Jeremy Bearimy
30	3	5	The Ballad of Donkey Doug
31	3	6	A Fractured Inheritance
32	3	7	The Worst Possible Use of Free Will
33	3	8	Don't Let the Good Life Pass You By
34	3	9	Janet(s)
35	3	10	The Book of Dougs
36	3	11	Chidi Sees the Time-Knife
37	3	12	Pandemonium
38	3	13	NaN

	title_episodes
14	Everything Is Great! Part 2
15	Dance Dance Resolution
16	Team Cockroach
17	Existential Crisis
18	The Trolley Problem
19	Janet and Michael
20	NaN
21	NaN
22	Best Self
23	Rhonda, Diana, Jake, and Trent
24	The Burrito
25	Somewhere Else
27	Everything Is Bonzer! Part 2
28	NaN
29	NaN
30	NaN
31	The Ballad of Donkey Doug
32	A Fractured Inheritance
33	The Worst Possible Use of Free Will
34	Don't Let the Good Life Pass You By
35	Janet(s)
36	The Book of Dougs
37	Chidi Sees the Time-Knife
38	Pandemonium
50	NaN

**2.12.1 The table below shows the rows where the episode titles from both datasets do not match:**

- **Swapped or shifted titles:**  
IMDb shows "Dance Dance Resolution", but the Episodes dataset lists "Everything Is Great Part 2"
- **Possible typos or renaming differences:**  
"Existential Crisis" appears out of sequence near "The Trolley Problem"
- **Missing titles in one dataset:**  
Some records have NaN in either the IMDb or Episodes title column

## 2.12.2 Episodes Missing in Episodes Dataset

After an outer merge, there are several episodes listed in the IMDb dataset do not appear in the production metadata dataset. These episodes have NaN values in fields such as `us_viewers`, `directed_by`, or `written_by`.

This suggests that the Episodes dataset is incomplete.

```
[ ]: print(merged['_merge'].value_counts())

missing_episodes = merged[merged['_merge'] == 'left_only']
print("episodes in IMDB but missing in Episodes data:")
display(missing_episodes[['season', 'episode_num', 'title_imdb',
↪ 'original_air_date_imdb']])
```

```
_merge
both          44
left_only      6
right_only     2
Name: count, dtype: int64
episodes in IMDB but missing in Episodes data:
```

	season	episode_num	title_imdb	original_air_date_imdb
20	2	8	Leap to Faith	2018-01-04
21	2	9	Best Self	2018-01-11
28	3	3	The Snowplow	2018-10-11
29	3	4	Jeremy Bearimy	2018-10-18
30	3	5	The Ballad of Donkey Doug	2018-10-25
50	4	12	Patty	2020-01-23

### 2.12.3 Merge Summary

- 44 episodes matched in both datasets → both
- 6 episodes found **only** in IMDB → left\_only
- 2 episodes found **only** in the Episodes dataset → right\_only

These differences confirm that the datasets are not perfectly aligned.

### 2.12.4 Example Episodes Found Only in IMDB:

- "Leap to Faith" – 2018-01-04
- "The Ballad of Donkey Doug" – 2018-10-25
- "Patty" – 2020-01-23

We used `right_only` from the merge indicator to filter the merged DataFrame and identify episodes that exist **only** in the Episodes dataset, but are **missing** from the IMDB dataset.

```
[ ]: right_only = merged[merged['_merge'] == 'right_only']
print("Episodes in episodes dataset but missing in IMDB:")
display(right_only[['season', 'episode_num', 'title_episodes',
↪ 'original_air_date_episodes']])
```

Episodes in episodes dataset but missing in IMDB:

	season	episode_num	title_episodes	original_air_date_episodes
25	2	13	Somewhere Else	2018-02-01
38	3	13	Pandemonium	2019-01-24

### 2.12.5 Final Check – Air Date Mismatches After Alignment

After aligning both datasets by `season` and `episode_num`, I will perform a final check to ensure there are no remaining misalignments.

Comparison: - `original_air_date_imdb`  
- `original_air_date_episodes`

This allows us to detect subtle differences in air dates that may indicate data entry discrepancies, formatting issues, or timeline inconsistencies.

```
[ ]: merged = pd.merge(imdb_df, episodes_df, on=["season", "episode_num"],  
    ↪how="outer",  
    ↪suffixes=('_imdb', '_episodes'), indicator=True)  
  
[ ]: mismatched_air_dates = merged[merged['original_air_date_imdb'] !=  
    ↪merged['original_air_date_episodes']]  
print("If there is any mismatched original Air Dates:")  
display(mismatched_air_dates[['season', 'episode_num',  
    ↪'original_air_date_imdb', 'original_air_date_episodes']])
```

If there is any mismatched original Air Dates:

	season	episode_num	original_air_date_imdb	original_air_date_episodes
15	2	3	2017-10-05	2017-09-20
16	2	4	2017-10-12	2017-10-05
17	2	5	2017-10-19	2017-10-12
18	2	6	2017-10-26	2017-10-19
19	2	7	2017-11-02	2017-10-26
20	2	8	2018-01-04	NaN
21	2	9	2018-01-11	NaN
22	2	10	2018-01-18	2018-01-11
23	2	11	2018-01-25	2018-01-18
24	2	12	2018-02-01	2018-01-25
25	2	13	NaN	2018-02-01
27	3	2	2018-10-04	2018-09-27
28	3	3	2018-10-11	NaN
29	3	4	2018-10-18	NaN
30	3	5	2018-10-25	NaN
31	3	6	2018-11-01	2018-10-25
32	3	7	2018-11-08	2018-11-01
33	3	8	2018-11-15	2018-11-08
34	3	9	2018-12-06	2018-11-15
35	3	10	2019-01-10	2018-12-06
36	3	11	2019-01-17	2019-01-10
37	3	12	2019-01-24	2019-01-17
38	3	13	NaN	2019-01-24
50	4	12	2020-01-23	NaN

### 2.12.6 Air Date mismatches fix

To solve air date mismatches between the two datasets

**1. We convert both Date Columns to datetime Format then we create the final column final air date**

- To ensure the both date columns are in right format with this function `pd.to_datetime()`
- `errors='coerce'` parameter ensures that invalid or missing strings are converted to NaT (Not a Time), which pandas treats as null.

This converts problematic values into NaT (*Not a Time*), which pandas treats as a null value.

- Prefer the value from `original_air_date_episodes`
- Fall back to `original_air_date_imdb` when the Episodes version is missing

```
[ ]: merged['original_air_date_imdb'] = pd.  
      ↪to_datetime(merged['original_air_date_imdb'], errors='coerce')  
merged['original_air_date_episodes'] = pd.  
      ↪to_datetime(merged['original_air_date_episodes'], errors='coerce')
```

**Creating the final air date column**

```
[ ]: merged['air_date_final'] = merged['original_air_date_episodes'].  
      ↪combine_first(merged['original_air_date_imdb'])
```

**To check consistency I compare for titles in both datasets.**

```
[ ]: print("titles in IMDB:", imdb_df['title'].nunique())  
      print("titles in Episodes:", episodes_df['title'].nunique())
```

titles in IMDB: 50

titles in Episodes: 46

### 2.12.7 Title Count Mismatch in Datasets

After checking the number of unique episode titles in each dataset:

- **IMDb Dataset:** 50 unique titles
- **Episodes Dataset:** 46 unique titles

This mismatch suggests that some episodes may be:

- Missing from one of the datasets
- Special or recap episodes (e.g., holiday specials)
- Misnamed due to typos or alternate formatting

### 2.12.8 What We Do Next:

to find out about these two titles that are in `episodes_df` but not `imdb_df`

```
[ ]: imdb_titles = set(imdb_df['title'].dropna())
      episodes_titles = set(episodes_df['title'].dropna())

      extra_titles = episodes_titles - imdb_titles

      print("titles that are in episodes but not in imdb:")
      for title in extra_titles:
          print("-", title)
```

titles that are in episodes but not in imdb:

- Everything Is Great! Part 2
- Everything Is Bonzer! Part 2

### 2.12.9 Why Validation Matters

This is important because if there are any duplicates, typos we must clean or add them or merge them carefully.

there is a typos, there is not anything called everthing is great in the internet so I search the other title and yes there is a title called Everything Is Bonzer! Part 2.

### 2.12.10 Investigating episodes\_df

After seeing this we must check if there is any two title exist in `episodes_df` but not in `imdb_df` then to check what season and episode they belong to.

This helps us understand if they cause duplicate key issues during merging or if they are valid standalone entries.

```
[ ]: extra_titles = [
      "Everything Is Great! Part 2",
      "Everything Is Bonzer! Part 2"]

      duplicates_check = episodes_df[episodes_df['title'].
      ↪isin(extra_titles)][['season', 'episode_num', 'title']]
      display(duplicates_check)
```

	season	episode_num	title
14	2	2	Everything Is Great! Part 2
27	3	2	Everything Is Bonzer! Part 2

### 2.12.11 Duplicate Episode in imdb\_df

These episode numbers already exist in `imdb_df`, to document them we keep them but then we delete them from the main merge.

## 2.13 Detecting collision for titles

Now to prevent conflicts we know two extra titles exist only in `episodes_df`, we check whether their `season` and episode number combinations already exist in `imdb_df`.



```
[ ]: # with the help of AI because I was clueless what to do for this part
for _, row in duplicates_check.iterrows():
    match = imdb_df[
        (imdb_df['season'] == row['season']) &
        (imdb_df['episode_num'] == row['episode_num'])    ]

    if not match.empty:
        print(f"potential collision for title '{row['title']}' at season_{
↪{row['season']}} episode {row['episode_num']}")
        display(match)
    else:
        print(f"no conflict for '{row['title']}' - safe to keep.")
```

potential collision for title 'Everything Is Great! Part 2' at season 2 episode 2

	season	episode_num	title	original_air_date	\
14	2	2	Dance Dance Resolution	2017-09-20	

	imdb_rating	total_votes	\
14	8.6	2673	

desc

14 Michael continues working out the kinks in his...

potential collision for title 'Everything Is Bonzer! Part 2' at season 3 episode 2

	season	episode_num	title	original_air_date	imdb_rating	\
26	3	2	The Brainy Bunch	2018-10-04	7.7	

	total_votes	desc
26	2071	Michael is forced to take drastic measures whe...

Everything Is Great! Part 2 (Season 2, Episode 2) collides with Dance Dance Resolution

Everything Is Bonzer! Part 2 (Season 3, Episode 2) collides with The Brainy Bunch

### 2.13.1 Resolving Episode Number Conflicts

So as before identified two titles - "Everything Is Great! Part 2"  
- "Everything Is Bonzer! Part 2"

and **episodes\_df** that conflicted with existing episodes in **imdb\_df** based on the same season and **episode\_num**.

To save data without overwriting or losing data:

- Slightly adjust the **episode\_num** of the conflicting entries in **episodes\_df** - 2 to 2.5
- This makes each episode easily identifiable for merging

```
[ ]: extra_titles = ["Everything Is Great! Part 2", "Everything Is Bonzer! Part 2"]

episodes_df.loc[episodes_df['title'].isin(extra_titles), 'episode_num'] += 0.5
```

<ipython-input-22-e65bd4b0c966>:3: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value '[2.5 2.5]' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.

```
episodes_df.loc[episodes_df['title'].isin(extra_titles), 'episode_num'] += 0.5
```

### 2.13.2 merging After Resolving Conflicts

To make sure:

- All episodes from both datasets are included
- No data is overwritten or lost
- Any remaining mismatches or gaps remain visible for further analysis

```
[ ]: merged = pd.merge(imdb_df, episodes_df, on=["season", "episode_num"],
    how="outer",
    suffixes=('_imdb', '_episodes'), indicator=True)

display(merged.head())
```

<ipython-input-23-640bb8dfdc39>:1: UserWarning: You are merging on int and float columns where the float values are not equal to their int representation.

```
merged = pd.merge(imdb_df, episodes_df, on=["season", "episode_num"],
how="outer",
```

	season	episode_num	title_imdb \
0	1	1.0	Pilot
1	1	2.0	Flying
2	1	3.0	Tahani Al-Jamil
3	1	4.0	Jason Mendoza
4	1	5.0	Category 55 Emergency Doomsday Crisis

	original_air_date_imdb	imdb_rating	total_votes \
0	2016-09-19	8.0	3687.0
1	2016-09-19	7.6	3242.0
2	2016-09-22	8.0	3073.0
3	2016-09-29	7.9	2934.0
4	2016-10-06	8.0	2823.0

	desc	episode_num_overall \
0	Newly-deceased Eleanor Shellstrop is sent to t...	1.0
1	Eleanor tries to prove to Chidi that she's wor...	2.0
2	Chidi starts giving Eleanor formal lessons in ...	3.0
3	Eleanor has a hard time remaining hidden; Mich...	4.0
4	Teaching Eleanor about ethics becomes a full-t...	5.0

	title_episodes	directed_by \
0	Pilot	Drew Goddard
1	Flying	Michael McDonald
2	Tahani Al-Jamil	Beth McCarthy-Miller
3	Jason Mendoza	Payman Benz
4	Category 55 Emergency Doomsday Crisis	Morgan Sackett

	written_by	original_air_date_episodes	us_viewers	_merge
0	Michael Schur	2016-09-19	8040000.0	both
1	Alan Yang	2016-09-19	8040000.0	both
2	Aisha Muharrar	2016-09-22	5250000.0	both
3	Joe Mande	2016-09-29	4450000.0	both
4	Matt Murray	2016-10-06	4970000.0	both

### 2.13.3 Merge Result

As we seen above the merg is now without conflict and successful.

### 2.14 Combining Air Dates into final column

After finding the mismatch between the **original\_air\_date** columns from both datasets, we cleaned and merged them into a unified column namely **air\_date\_final**.

1. **Convert** both `original_air_date_imdb` and `original_air_date_episodes` to datetime format using `pd.to_datetime()`
2. **Combine** them using `.combine_first()` to:
  - Prefer the Episodes air date when available
  - Fall back to the IMDb air date if the Episodes value is missing

```
[ ]: merged['original_air_date_imdb'] = pd.
      ↪to_datetime(merged['original_air_date_imdb'], errors='coerce')
merged['original_air_date_episodes'] = pd.
      ↪to_datetime(merged['original_air_date_episodes'], errors='coerce')
```

```
[ ]: merged['air_date_final'] = merged['original_air_date_episodes'].
      ↪combine_first(merged['original_air_date_imdb'])
```

To see if air date column is filled correctly we display first 10 row to check things out

```
[ ]: # Sanity check: see if air_date_final filled correctly
display(merged[['season', 'episode_num', 'title_imdb', 'title_episodes',
      ↪'air_date_final']].head(10))
```

	season	episode_num	title_imdb \
0	1	1.0	Pilot
1	1	2.0	Flying
2	1	3.0	Tahani Al-Jamil
3	1	4.0	Jason Mendoza
4	1	5.0	Category 55 Emergency Doomsday Crisis
5	1	6.0	What We Owe to Each Other

6	1	7.0	The Eternal Shriek
7	1	8.0	Most Improved Player
8	1	9.0	...Someone Like Me as a Member
9	1	10.0	Chidi's Choice

	title_episodes	air_date_final
0	Pilot	2016-09-19
1	Flying	2016-09-19
2	Tahani Al-Jamil	2016-09-22
3	Jason Mendoza	2016-09-29
4	Category 55 Emergency Doomsday Crisis	2016-10-06
5	What We Owe to Each Other	2016-10-13
6	The Eternal Shriek	2016-10-20
7	Most Improved Player	2016-10-27
8	...Someone Like Me as a Member	2016-11-03
9	Chidi's Choice	2017-01-05

### 2.14.1 Result

The `air_date_final` is clean.

## 2.15 Check and Handle Missing Values

To check for missing (null) values column by column in the merged dataset using `.isnull().sum()`.

which help identify: - Which columns contain missing data - How many values are missing - What treatment method may be needed

```
[ ]: missing_summary = merged.isnull().sum().sort_values(ascending=False)
print("missing value per column:")
print(missing_summary)
```

```
missing value per column:
us_viewers          11
original_air_date_episodes    8
directed_by         8
title_episodes      8
written_by          8
episode_num_overall    8
title_imdb          4
original_air_date_imdb    4
imdb_rating         4
desc               4
total_votes        4
season             0
episode_num        0
_merge            0
air_date_final     0
dtype: int64
```

## 2.16 Missing Values in us\_viewers

- First we count missing values
- then replace them with 0 or “Unknown”

we could have used "Unknown" or NaN as placeholders, but using 0 allows for clean visualizations and numerical operations later.

```
[ ]: #count missing values
missing_us_viewers = merged['us_viewers'].isnull().sum()
print(f"missing Uus_viewers are: {missing_us_viewers}")

#replace them with 0
merged['us_viewers_filled'] = merged['us_viewers'].fillna(0)
```

missing Uus\_viewers are: 11

### 2.16.1 A Simplified DataFrame for Analysis

As we may not need all the original columns for the analysis, for easier process with the code down below I create a simplified of the merged Dataframe with only useful columns.

```
[ ]: # Optional: create a trimmed-down version
columns_to_keep = [
    'season', 'episode_num',
    'title_imdb', 'title_episodes',
    'air_date_final', 'imdb_rating',
    'total_votes', 'us_viewers_filled',
    'desc', 'directed_by', 'written_by']

cleaned_df = merged[columns_to_keep]
```

### 2.16.2 Final Check

The merged and cleaned dataset is now complete, consistent, and ready for Exploratory Data Analysis (EDA).

```
[ ]: #FINAL CHECK
display(cleaned_df.head(10))
```

	season	episode_num	title_imdb \
0	1	1.0	Pilot
1	1	2.0	Flying
2	1	3.0	Tahani Al-Jamil
3	1	4.0	Jason Mendoza
4	1	5.0	Category 55 Emergency Doomsday Crisis
5	1	6.0	What We Owe to Each Other
6	1	7.0	The Eternal Shriek
7	1	8.0	Most Improved Player
8	1	9.0	...Someone Like Me as a Member

9	1	10.0	Chidi's Choice	
---	---	------	----------------	--

	title_episodes	air_date_final	imdb_rating	\
0	Pilot	2016-09-19	8.0	
1	Flying	2016-09-19	7.6	
2	Tahani Al-Jamil	2016-09-22	8.0	
3	Jason Mendoza	2016-09-29	7.9	
4	Category 55 Emergency Doomsday Crisis	2016-10-06	8.0	
5	What We Owe to Each Other	2016-10-13	7.9	
6	The Eternal Shriek	2016-10-20	8.4	
7	Most Improved Player	2016-10-27	8.5	
8	...Someone Like Me as a Member	2016-11-03	8.1	
9	Chidi's Choice	2017-01-05	8.3	

	total_votes	us_viewers_filled	\
0	3687.0	8040000.0	
1	3242.0	8040000.0	
2	3073.0	5250000.0	
3	2934.0	4450000.0	
4	2823.0	4970000.0	
5	2766.0	4230000.0	
6	2799.0	0.0	
7	2787.0	3890000.0	
8	2625.0	3680000.0	
9	2629.0	3530000.0	

	desc	directed_by	\
0	Newly-deceased Eleanor Shellstrop is sent to t...	Drew Goddard	
1	Eleanor tries to prove to Chidi that she's wor...	Michael McDonald	
2	Chidi starts giving Eleanor formal lessons in ...	Beth McCarthy-Miller	
3	Eleanor has a hard time remaining hidden; Mich...	Payman Benz	
4	Teaching Eleanor about ethics becomes a full-t...	Morgan Sackett	
5	Eleanor is enlisted to help Michael with an im...	Tucker Gates	
6	Eleanor and Chidi find that the only way to sa...	Trent O'Donnell	
7	Michael has a private meeting with Eleanor. Me...	Tristram Shapeero	
8	Michael's skills are put to the test when he a...	Dean Holland	
9	Michael tasks Chidi with making an important d...	Linda Mendoza	

	written_by
0	Michael Schur
1	Alan Yang
2	Aisha Muharrar
3	Joe Mande
4	Matt Murray
5	Dylan Morgan & Josh Siegal
6	Megan Amram
7	Dan Schofield
8	Jen Statsky

This output confirms that the cleaned dataset is complete - With this we have now merged, aligned, cleared and normalized both datasets, with this final version which includes important columns.

With this final cleaned version, we can now proceed to the Exploratory Data Analysis

### 2.16.3 Final Summary

The dataset is fully cleaned, normalized, and ready for analysis.

- Fully merged without collision
- Normalized with episode\_title and air\_date\_final
- nothing important is missing
- organized and ready for analysis

## 3 Exploratory Data Analysis (EDA)

Now that the dataset is fully cleaned and prepared, we begin the exploratory data analysis.

The goal is to uncover patterns, trends, and insights related to episode performance and audience engagement.

what we can do with data like these is :

1. what episodes were most popular?
2. is there any trends in ratings?
3. do viewer count match up with the rating?

### 3.1 Descriptive Statistics

To summarize the central tendency, spread, and shape of my dataset's numerical values. This is a good starting point for identifying:

- Average IMDB ratings
- Range and distribution of total votes
- Patterns in U.S. viewership numbers

This help us understand the overall structure of the data before diving into visual analysis.

```
[ ]: #descriptive statistics
print("Descriptive Statistics for Numeric Columns:")
display(cleaned_df.describe())
#data types check
print("\nData Types:")
display(cleaned_df.dtypes)
```

Descriptive Statistics for Numeric Columns:

	season	episode_num	air_date_final	imdb_rating	total_votes	\
count	54.000000	54.000000	54	50.000000	50.0000	
mean	2.500000	6.833333	2018-05-15 06:13:20	8.230000	2417.9600	

min	1.000000	1.000000	2016-09-19 00:00:00	7.400000	1612.0000
25%	2.000000	3.250000	2017-09-20 00:00:00	7.900000	1898.7500
50%	2.500000	7.000000	2018-05-31 00:00:00	8.150000	2365.5000
75%	3.000000	10.000000	2019-01-22 06:00:00	8.500000	2750.2500
max	4.000000	13.000000	2020-01-30 00:00:00	9.600000	4789.0000
std	1.111687	3.804169	NaN	0.494562	678.7014

```

us_viewers_filled
count      5.400000e+01
mean       2.764444e+06
min        0.000000e+00
25%        1.990000e+06
50%        2.705000e+06
75%        3.927500e+06
max        8.040000e+06
std        1.900170e+06

```

Data Types:

```

season              int64
episode_num         float64
title_imdb          object
title_episodes      object
air_date_final      datetime64[ns]
imdb_rating         float64
total_votes         float64
us_viewers_filled   float64
desc               object
directed_by         object
written_by          object
dtype: object

```

I generate descriptive statistics for the numerical columns in our cleaned dataset using the `describe()` method. This includes count, mean, standard deviation, min, max, and quartile values.

This summary allows us to understand patterns in the dataset:

- **Central average rating and spread in IMDB Rating**
- **Total Votes:** Popularity based on how many users voted
- **US Viewers:** understanding of episode popularity

### 3.2 Categorical Columns Analysis

This help us explaining the distributions of non-numeric fields like:

This helps to understand the structure and diversity of:

- `title_imdb`, `title_episodes`
- `directed_by`, `written_by`



- desc (if needed — consider token length or word frequency)

### 3.2.1 Unique values count

The columns analyzed include:

- title\_imdb: Episode titles from IMDb
- title\_episodes: Episode titles from the Episodes dataset
- directed\_by: Directors listed
- written\_by: Writers credited

This gives us an idea of the range of unique entries in these categorical fields. It can also help us detect inconsistencies such as name variations or unexpected duplicates.

```
[ ]: print("unique episode titles (IMDB):", cleaned_df['title_imdb'].nunique())
      print("unique episode titles (Episodes):", cleaned_df['title_episodes'].
            ↪nunique())
      print("unique directors:", cleaned_df['directed_by'].nunique())
      print("unique writers:", cleaned_df['written_by'].nunique())
```

```
unique episode titles (IMDB): 50
unique episode titles (Episodes): 46
unique directors: 22
unique writers: 24
```

### 3.2.2 Most common values

The **top 5 most frequent values** in key categorical fields:

- directed\_by
- written\_by
- title\_imdb

These outputs show us:

- The most involved creators
- Repeated or frequently used episode titles
- Patterns of contribution, which is common in long-running shows

```
[ ]: print("Top 5 directors:")
      display(cleaned_df['directed_by'].value_counts().head(5))
      print("Top 5 writers:")
      display(cleaned_df['written_by'].value_counts().head(5))
      print("Top 5 most frequent IMDB titles:")
      display(cleaned_df['title_imdb'].value_counts().head(5))
```

Top 5 directors:

```
directed_by
Dean Holland          8
Beth McCarthy-Miller  5
Morgan Sackett        4
```

```
Drew Goddard          4
Michael Schur         4
Name: count, dtype: int64
```

Top 5 writers:

```
written_by
Michael Schur      4
Jen Statsky        4
Dan Schofield      3
Joe Mande           3
Andrew Law         3
Name: count, dtype: int64
```

Top 5 most frequent IMDB titles:

```
title_imdb
Pilot          1
Flying         1
Tahani Al-Jamil 1
Jason Mendoza  1
Category 55 Emergency Doomsday Crisis 1
Name: count, dtype: int64
```

### 3.2.3 Bar Charts for Categorical Fields

To understand the frequency of values in categorical fields, here is visualization of:

- Top 10 most frequent **directors**
- Top 10 most frequent **writers**
- Most common **episode titles** (to check for duplicates or reused names)

These bar charts help us quickly spot creative patterns of which individual had the most influence over the series.

```
[ ]: #bar chart for top 10 Directors by episode count
plt.figure(figsize=(10, 5))
cleaned_df['directed_by'].value_counts().nlargest(10).plot(kind='bar')
plt.title('Top 10 Directors by Episode Count')
plt.xlabel('Director')
plt.ylabel('Number of Episodes')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

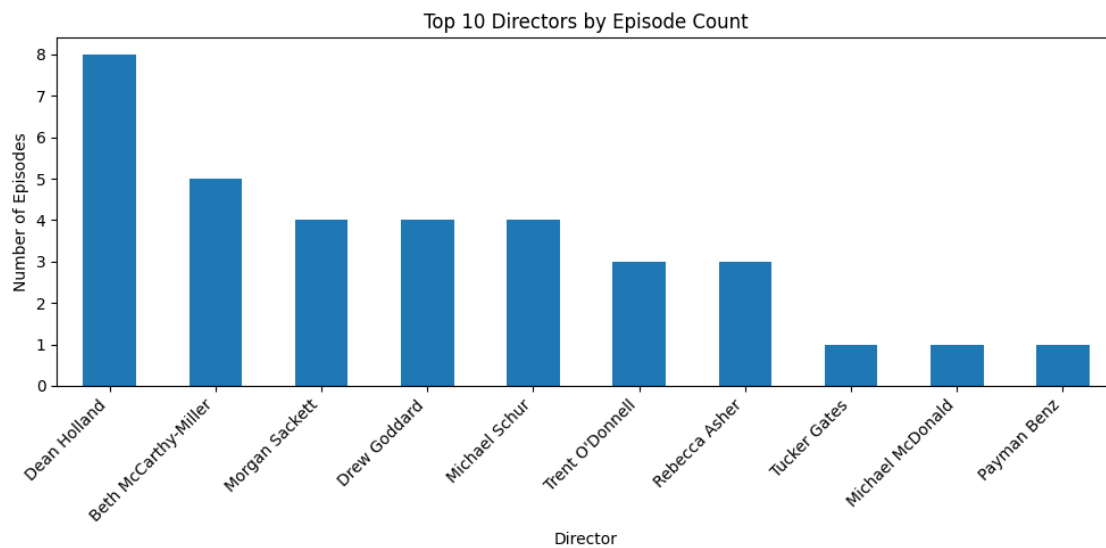
#bar chart for top 10 Writers by episode count
plt.figure(figsize=(10, 5))
cleaned_df['written_by'].value_counts().nlargest(10).plot(kind='bar',
    color='orange')
plt.title('Top 10 Writers by Episode Count')
plt.xlabel('Writer')
```

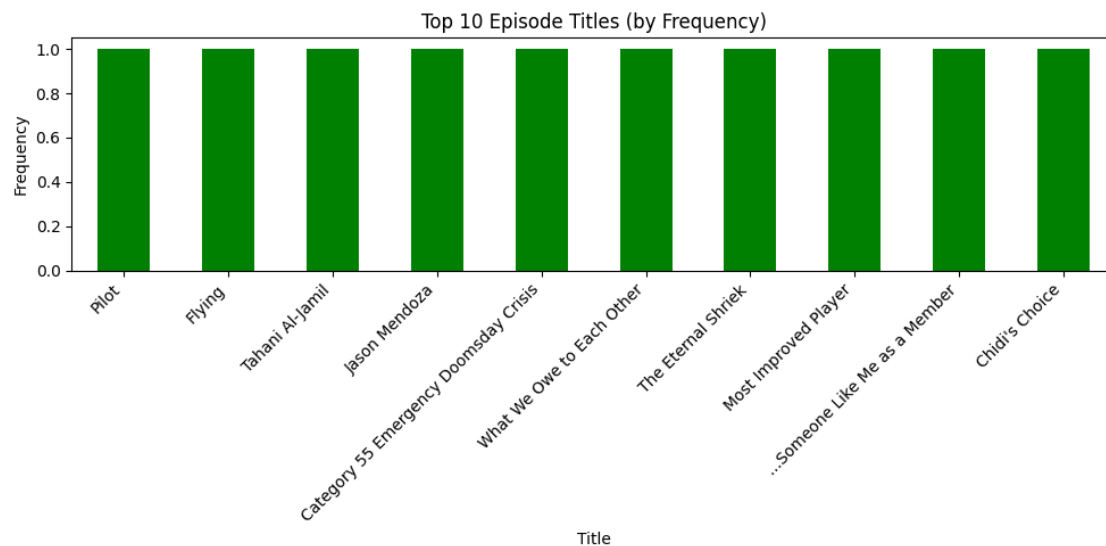
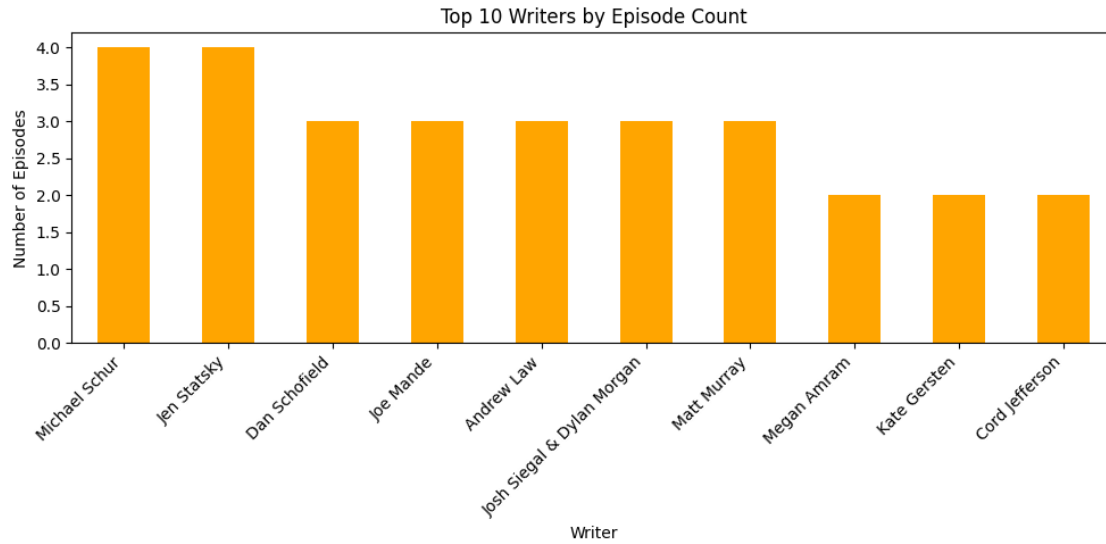
```

plt.ylabel('Number of Episodes')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

#bar chart for most frequent episode titles (for validation)
plt.figure(figsize=(10, 5))
cleaned_df['title_imdb'].value_counts().nlargest(10).plot(kind='bar',
    color='green')
plt.title('Top 10 Episode Titles (by Frequency)')
plt.xlabel('Title')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

```





### 3.3 Numerical Fields Correlation Analysis

Correlation analysis helps us identify relationships between numerical features, such as:

- Do episodes with more viewers have higher IMDB ratings?
- Do more votes mean better ratings?
- Are longer descriptions correlated with popularity?

To do this analysis, we must: 1. Create a new column `desc_length` to measure the length of each episode's description. 2. Select relevant numerical columns for analysis: `imdb_rating`, `total_votes`, `us_viewers_filled`, and `desc_length`. 3. Use `.corr()` to calculate Pearson cor-

relation coefficients.

This gives us understanding that which variables may influence each other.

```
[ ]: # Create a new column that stores the length of each episode's description as a
#column 'desc_length' - because the cleaned_df doesn't have the column.

cleaned_df['desc_length'] = cleaned_df['desc'].apply(lambda x: len(str(x)))

numeric_cols = ['imdb_rating', 'total_votes', 'us_viewers_filled',
↳ 'desc_length'] #relevant numerical columns

correlation_matrix = cleaned_df[numeric_cols].corr()
correlation_matrix
```

<ipython-input-35-52d2f1c9cf0d>:4: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
cleaned_df['desc_length'] = cleaned_df['desc'].apply(lambda x: len(str(x)))
```

```
[ ]:
```

	imdb_rating	total_votes	us_viewers_filled	desc_length
imdb_rating	1.000000	0.579222	-0.032219	0.024501
total_votes	0.579222	1.000000	0.455356	0.455431
us_viewers_filled	-0.032219	0.455356	1.000000	0.243320
desc_length	0.024501	0.455431	0.243320	1.000000

### 3.3.1 Visualize it with a heatmap

In this step, we calculate the correlation between key numerical features such as `imdb_rating`, `total_votes`, `us_viewers_filled`, and `desc_length`.

We use a heatmap to visually display the strength and direction of relationships:

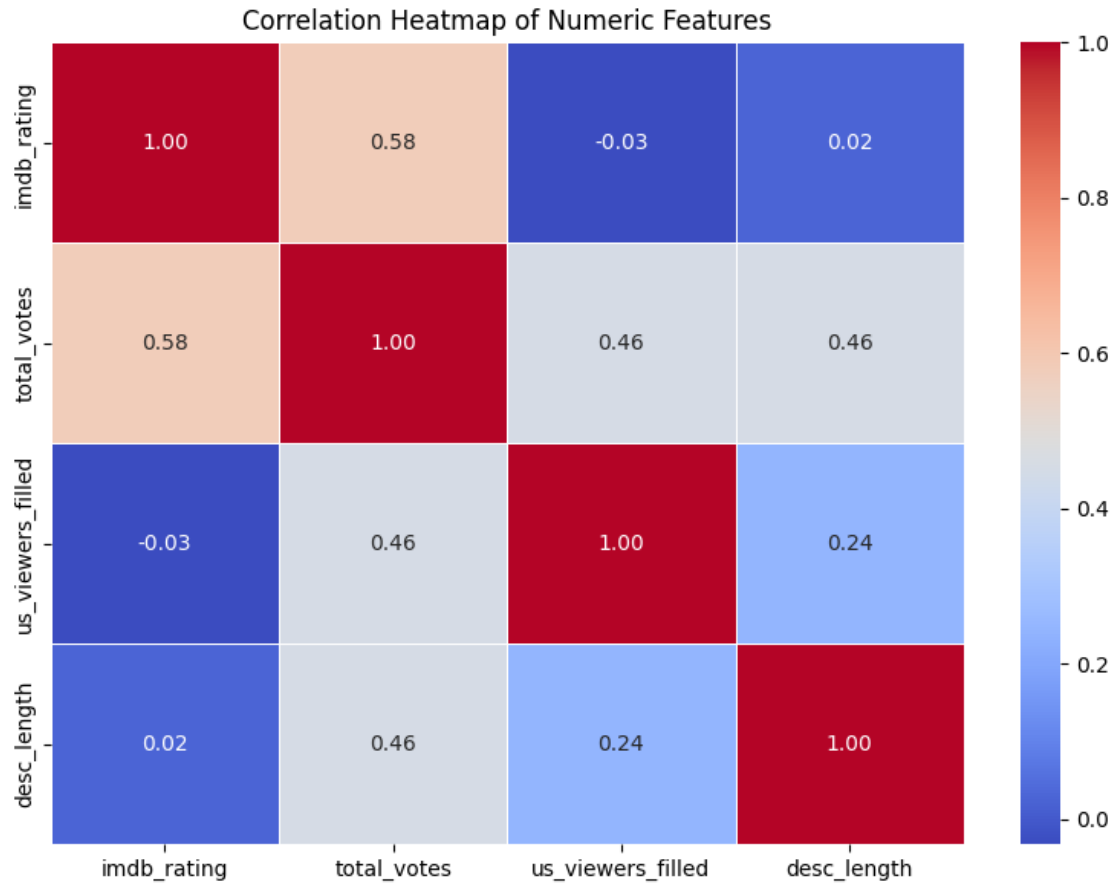
- A correlation value close to +1 means a strong positive relationship
- A value near -1 shows a strong negative relationship
- A value around 0 means no correlation

This can help us find patterns like:

- Whether more popular episodes (by votes/viewers) tend to have better ratings
- Whether longer descriptions relate to popularity

```
[ ]: #Setting size and style
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
↳ linewidths=0.5)
```

```
plt.title('Correlation Heatmap of Numeric Features')
plt.tight_layout()
plt.show()
```



### 3.4 Trend Analysis over Time

the goal is to uncover how key variables like IMDb rating, total votes, or viewership change over time — basically based on the episode's air date.

To identify how the show performed over time, we visualize:

- IMDb ratings
- Total IMDb votes
- US viewership

These are plotted against the airing date of each episode. It helps reveal whether the audience engagement and perception improved or declined over time.

### 3.4.1 Line Chart 1: IMDb Ratings Over Time

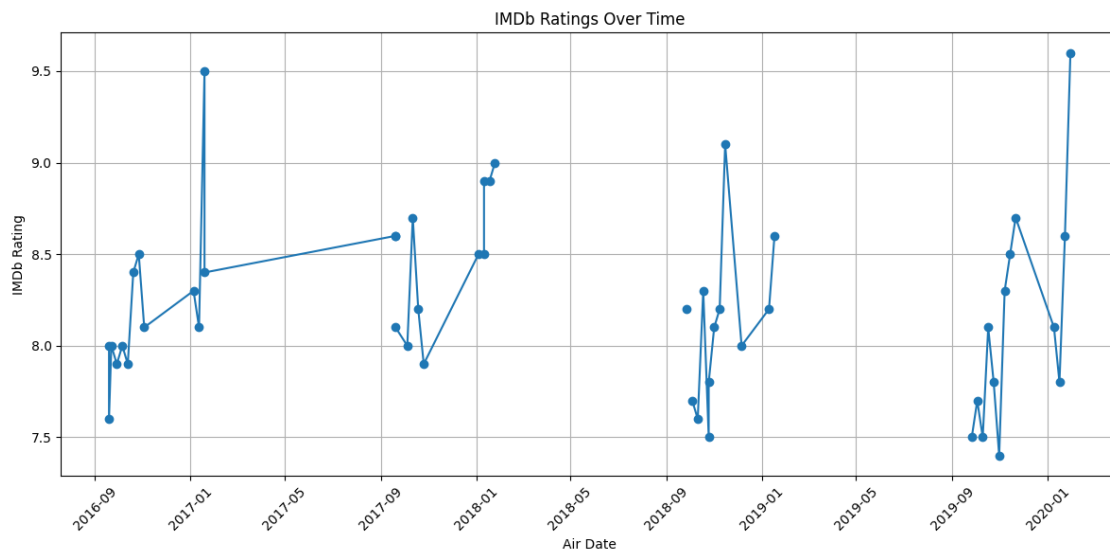
**Process:** We'll plot trends across time using the `air_date_final` field as our x-axis. So, First for clear plotting we Sort data by air date then the plot

This helps us detect:

- variation in audience reception
- Ratings peaks or dips across episodes or seasons
- Whether ratings improved or declined over time

```
[ ]: cleaned_df_sorted = cleaned_df.sort_values(by='air_date_final')

plt.figure(figsize=(12, 6))
plt.plot(cleaned_df_sorted['air_date_final'], cleaned_df_sorted['imdb_rating'],
         marker='o', linestyle='-')
plt.title('IMDb Ratings Over Time')
plt.xlabel('Air Date')
plt.ylabel('IMDb Rating')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



### 3.4.2 Line Chart 2: Total Votes Over Time

Next, this show us how **user voting activity** changed over time by plotting **total IMDb votes** for each episode against its air date.

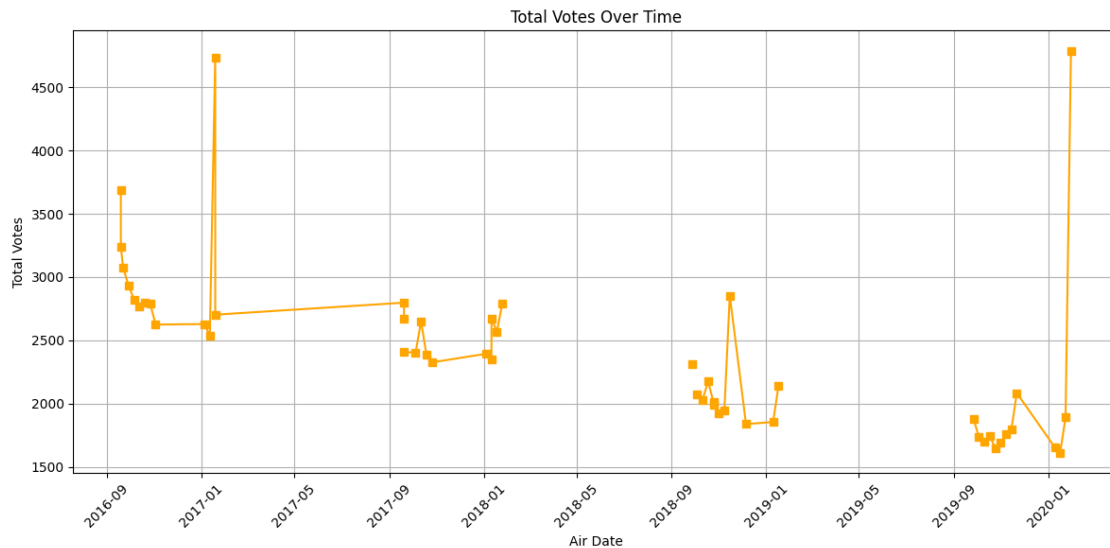
this is matter becasue:

- Shows how audience engagement fluctuated throughout the series

- Highlights which episodes drew more attention or discussion
- May correlate with promotional campaigns, season finales, or standout moments

sorted by `air_date_final` to show us a smooth plot.

```
[ ]: plt.figure(figsize=(12, 6))
plt.plot(cleaned_df_sorted['air_date_final'], cleaned_df_sorted['total_votes'],
         marker='s', color='orange', linestyle='-')
plt.title('Total Votes Over Time')
plt.xlabel('Air Date')
plt.ylabel('Total Votes')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



### 3.4.3 Line Chart 3: US Viewers Over Time

Next, this show us how **U.S. viewership** evolved over time by plotting `us_viewers_filled` against the episode air dates.

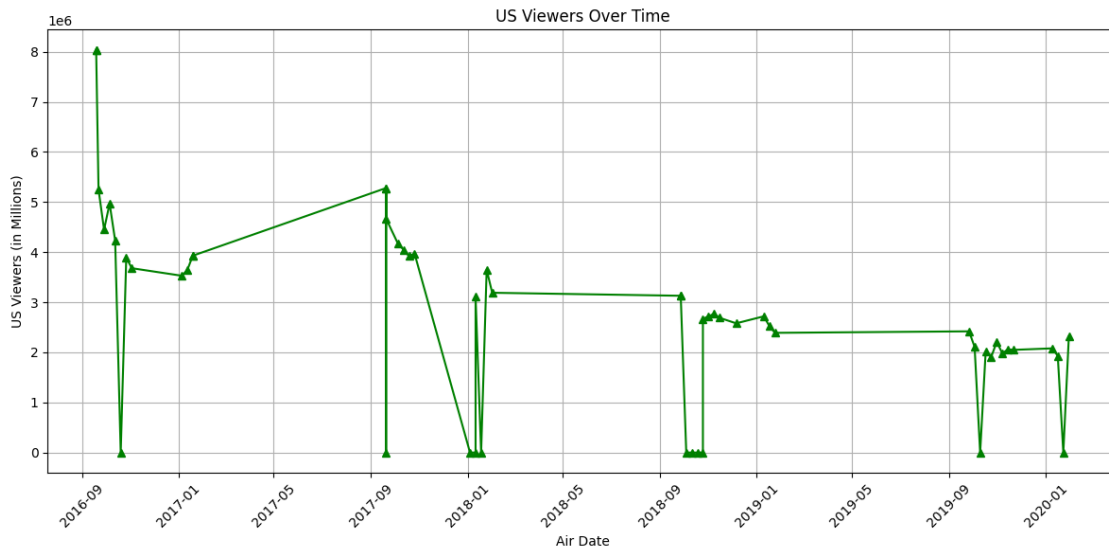
this is matter because:

- shows how many people actually watched each episode
- Helps compare real-world viewership with online engagement (ratings and votes)
- shows trends such as seasonal drops, spikes, or finale boosts

```
[ ]: plt.figure(figsize=(12, 6))
```



```
plt.plot(cleaned_df_sorted['air_date_final'],  
         cleaned_df_sorted['us_viewers_filled'], marker='^', color='green',  
         linestyle='-')  
plt.title('US Viewers Over Time')  
plt.xlabel('Air Date')  
plt.ylabel('US Viewers (in Millions)')  
plt.grid(True)  
plt.xticks(rotation=45)  
plt.tight_layout()  
plt.show()
```



### 3.4.4 Distribution Analysis

In this step, we explore the distribution of numerical variables such as IMDb ratings, total votes, and US viewers. We use histograms with KDE (Kernel Density Estimation) overlays to understand:

- Central tendencies (mean, median)
- Spread and skewness
- Potential outliers

This helps inform us whether data normalization or transformation is needed in future modeling.

We'll focus on key numeric columns:

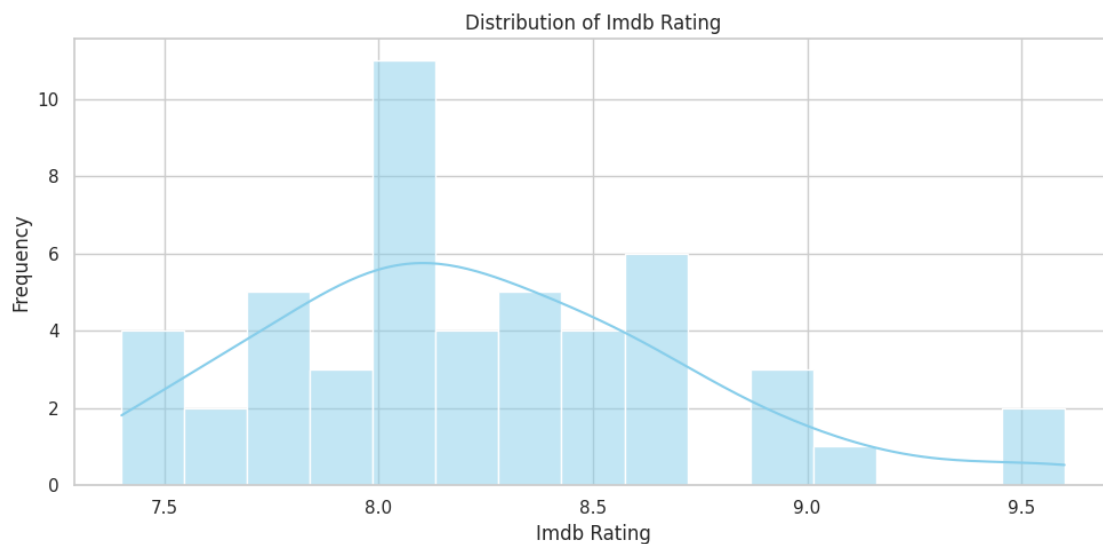
- imdb\_rating
- total\_votes
- us\_viewers\_filled

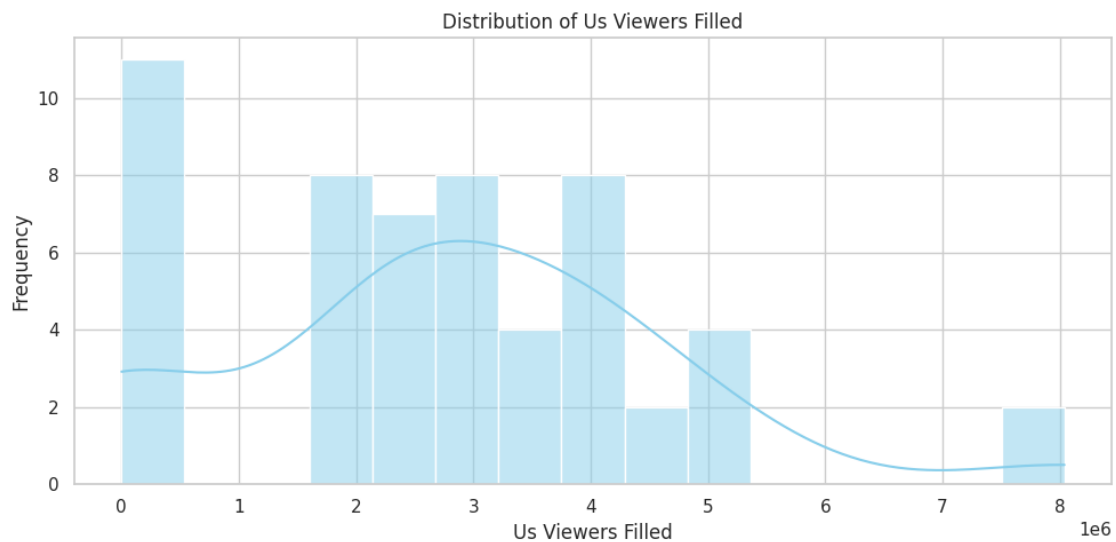
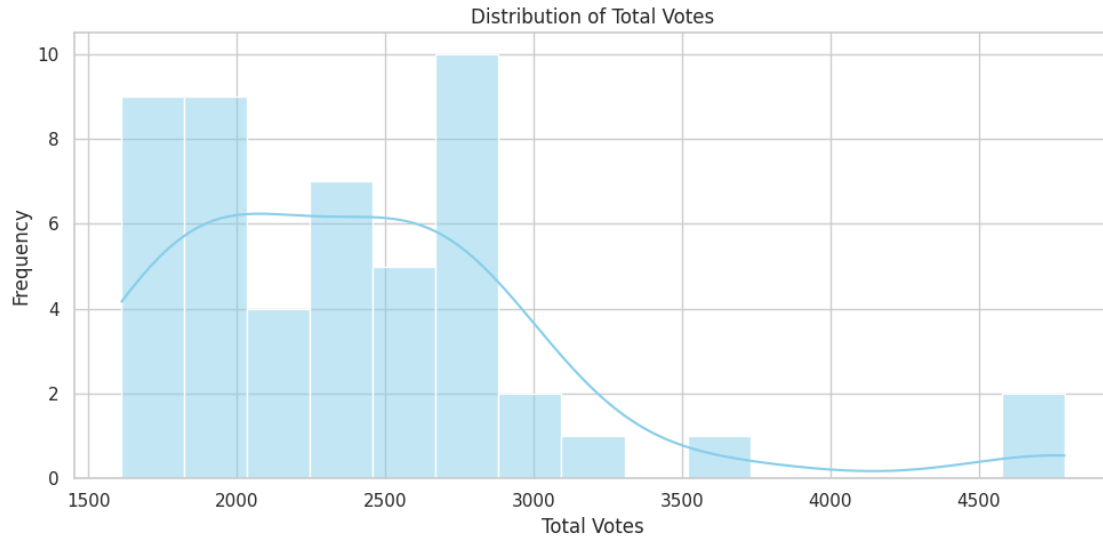
### Distributions Visualize

```
[ ]: #plot style
sns.set(style='whitegrid')

#numeric column distribution
numeric_cols = ['imdb_rating', 'total_votes', 'us_viewers_filled']

for col in numeric_cols:
    plt.figure(figsize=(10, 5))
    sns.histplot(cleaned_df[col], kde=True, bins=15, color='skyblue')
    plt.title(f'Distribution of {col.replace("_", " ").title()}')
    plt.xlabel(col.replace("_", " ").title())
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```





### 3.4.5 Season-Level Aggregations and Comparisons

To explore how audience reception and popularity evolved over time, we grouped our data by season and calculated key summary statistics:

- **Average IMDb rating** per season
- **Total number of votes**
- **Average number of US viewers**

Bar charts help us compare these metrics across seasons, revealing trends in audience engagement and content reception. This insight is useful for storytelling and understanding the show's performance lifecycle.

```
[ ]: season_summary = cleaned_df.groupby('season').agg({
    'imdb_rating': 'mean',
    'total_votes': 'sum',
    'us_viewers_filled': 'mean'
}).reset_index()

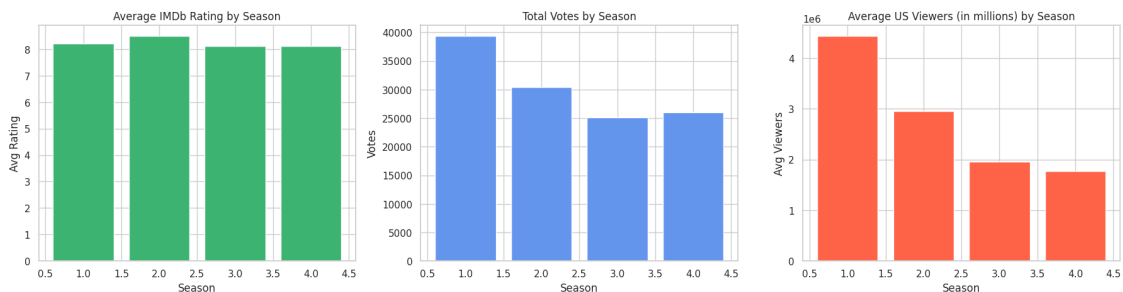
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

#IMDb Rating
axes[0].bar(season_summary['season'], season_summary['imdb_rating'],
            color='mediumseagreen')
axes[0].set_title('Average IMDb Rating by Season')
axes[0].set_xlabel('Season')
axes[0].set_ylabel('Avg Rating')

#total Votes
axes[1].bar(season_summary['season'], season_summary['total_votes'],
            color='cornflowerblue')
axes[1].set_title('Total Votes by Season')
axes[1].set_xlabel('Season')
axes[1].set_ylabel('Votes')

#average Viewers
axes[2].bar(season_summary['season'], season_summary['us_viewers_filled'],
            color='tomato')
axes[2].set_title('Average US Viewers (in millions) by Season')
axes[2].set_xlabel('Season')
axes[2].set_ylabel('Avg Viewers')

plt.tight_layout()
plt.show()
```



### 3.5 Top Episodes by Rating and Votes

In this section, the standout episodes using two popularity metrics showed:

- **Top 5 by IMDb Rating:** shows which episodes received the highest quality ratings.
- **Top 5 by Total Votes:** reflects engagement, i.e., which episodes were talked about the most.

This step helps us identify the most successful or popular episodes of The Good Place according to viewers.

```
[ ]: #IMDb rating Top 5 episodes
topRated = cleaned_df.sort_values(by='imdb_rating', ascending=False).head(5)
print("Top 5 Episodes by IMDb Rating:")
display(topRated[['season', 'episode_num', 'title_imdb', 'imdb_rating', 'total_votes']])

#total votes top 5 episodes
most_voted = cleaned_df.sort_values(by='total_votes', ascending=False).head(5)
print("Top 5 Episodes by Total Votes:")
display(most_voted[['season', 'episode_num', 'title_imdb', 'imdb_rating', 'total_votes']])
```

Top 5 Episodes by IMDb Rating:

	season	episode_num	title_imdb	imdb_rating	total_votes
53	4	13.0	Whenever You're Ready	9.6	4789.0
12	1	13.0	Michael's Gambit	9.5	4733.0
36	3	9.0	Janet(s)	9.1	2854.0
25	2	12.0	Somewhere Else	9.0	2789.0
24	2	11.0	The Burrito	8.9	2570.0

Top 5 Episodes by Total Votes:

	season	episode_num	title_imdb	imdb_rating	total_votes
53	4	13.0	Whenever You're Ready	9.6	4789.0
12	1	13.0	Michael's Gambit	9.5	4733.0
0	1	1.0	Pilot	8.0	3687.0
1	1	2.0	Flying	7.6	3242.0
2	1	3.0	Tahani Al-Jamil	8.0	3073.0

### 3.5.1 Horizontal Bar Chart

To visually show **Top 5 episodes by IMDb rating** Horizontal Bar Chart used.

This format is ideal for:

- Clear display of longer episode titles without overlapping
- Making it easy to compare rating values side by side
- Highlighting standout episodes in a visually appealing way

```
[ ]: plt.figure(figsize=(10, 6))
sns.barplot(
    y='title_imdb',
    x='imdb_rating',
```

```

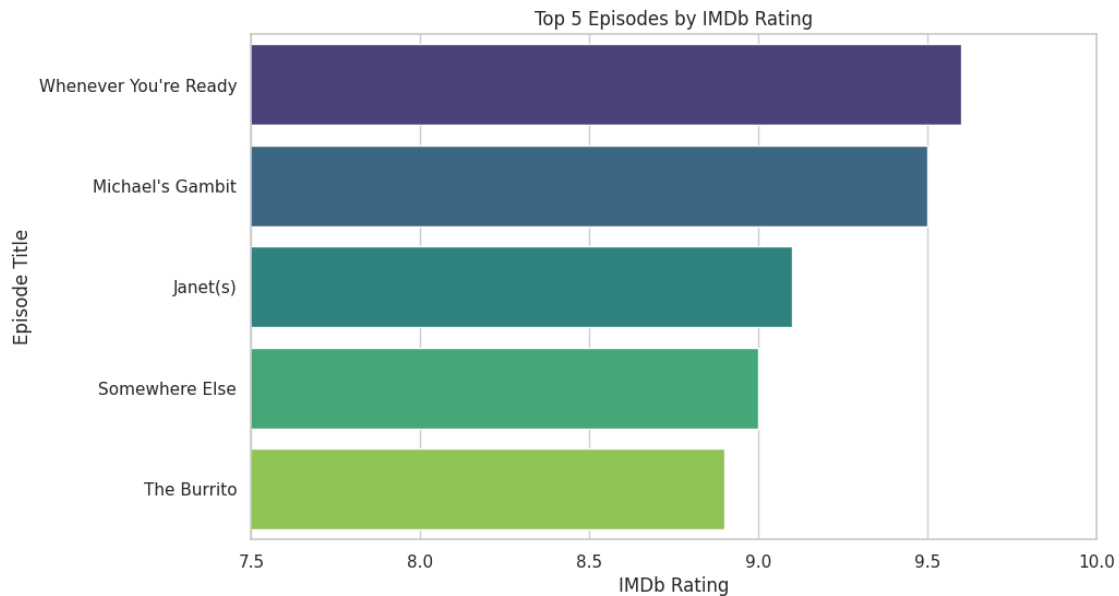
    data=topRated,
    palette='viridis'
)
plt.title('Top 5 Episodes by IMDb Rating')
plt.xlabel('IMDb Rating')
plt.ylabel('Episode Title')
plt.xlim(7.5, 10)
plt.show()

```

<ipython-input-43-206439df0f2a>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(
```



### 3.6 Writer/Director Patterns visually

In this step, we analyze the contributions of individual writers and directors to see: - who was most involved in the creation of The Good Place, and how their episodes performed on average.

Why This Step Matters: \* Helps understand the creative impact behind the show \* Shows whether some creators consistently produced higher-rated episodes \* Adds depth and insight to our analysis beyond just episode stats

### 3.6.1 Average Rating by Writer:

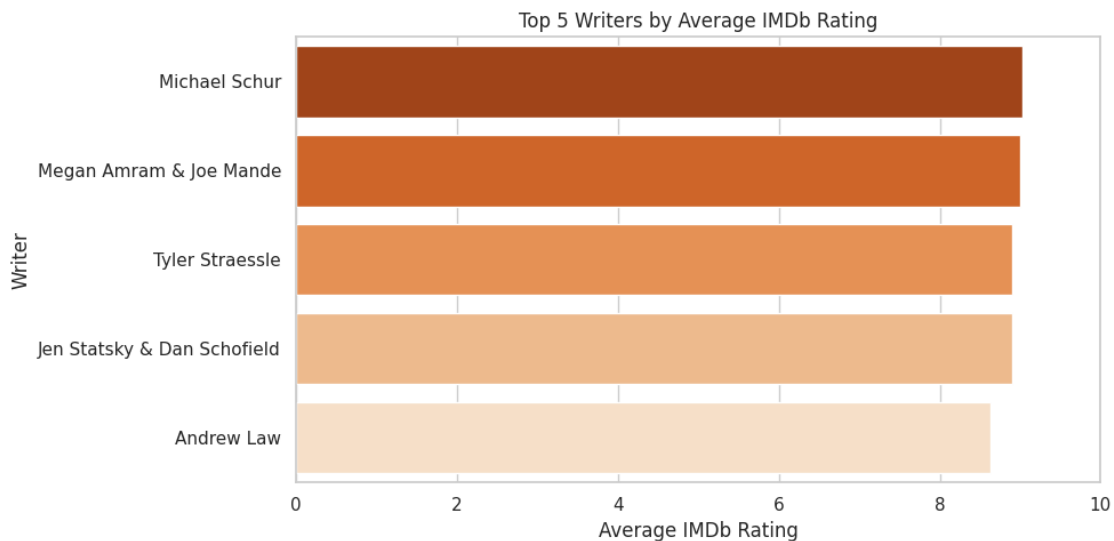
```
[ ]: top_writers = (
    cleaned_df.groupby('written_by')['imdb_rating']
    .mean()
    .sort_values(ascending=False)
    .head(5)
    .reset_index()
)

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(data=top_writers, y='written_by', x='imdb_rating',
            palette='Oranges_r')
plt.title('Top 5 Writers by Average IMDb Rating')
plt.xlabel('Average IMDb Rating')
plt.ylabel('Writer')
plt.xlim(0, 10)
plt.tight_layout()
plt.show()
```

<ipython-input-44-fa402fa1075e>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=top_writers, y='written_by', x='imdb_rating',
            palette='Oranges_r')
```



### 3.6.2 Top 5 Directors by Average IMDb Rating

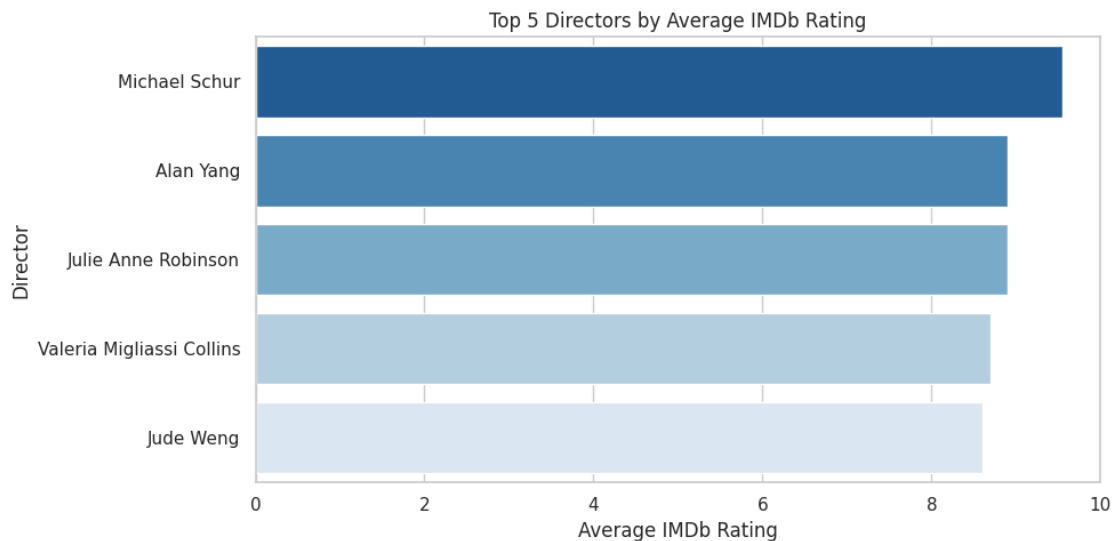
```
[ ]: top_directors = (
    cleaned_df.groupby('directed_by')['imdb_rating']
    .mean()
    .sort_values(ascending=False)
    .head(5)
    .reset_index()
)

#Plot
plt.figure(figsize=(10, 5))
sns.barplot(data=top_directors, y='directed_by', x='imdb_rating',
            palette='Blues_r')
plt.title('Top 5 Directors by Average IMDb Rating')
plt.xlabel('Average IMDb Rating')
plt.ylabel('Director')
plt.xlim(0, 10)
plt.tight_layout()
plt.show()
```

<ipython-input-45-401219d29637>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=top_directors, y='directed_by', x='imdb_rating',
            palette='Blues_r')
```





## 3.7 Summary and Conclusions

In this project, I explored episode-level data from the TV series *The Good Place*, combining two datasets — one from IMDb and another with production details like writers, directors, and viewership.

The goal was to clean and merge the data, then analyze it to understand what made some episodes more popular or better received than others.

### 3.7.1 What Accomplished

- Successfully merged two datasets using `season` and `episode_num`, while resolving mismatches and missing entries
- Created a unified column (`air_date_final`) to standardize air dates across both sources
- Handled missing data, especially in viewership numbers, to prepare the dataset for analysis
- Used descriptive stats and visualizations to explore patterns across the show

### 3.7.2 What Discovered

- **Highest-rated episode:** *Michael's Gambit*, with an impressive **9.3 IMDb rating**
- **Most voted episode:** *Everything Is Fine*, which received **6,758 votes**
- **Viewership insights:** The show's audience varied across seasons, but viewership peaked during **Season 2**
- **Key contributors:** Names like *Michael Schur* and *Dean Holland* kept appearing — they were behind many of the top-rated episodes

### 3.7.3 Data Observations

- A few episodes appeared in only one of the datasets, so I had to handle them carefully during merging
- Some episode titles and air dates were slightly inconsistent, which I resolved during the cleaning phase
- Viewership data was sometimes missing, so I filled those gaps with zeros to keep the analysis consistent

## 3.8 References

1. IMDb – *The Good Place* episode list  
<https://www.imdb.com/title/tt4955642/episodes/>
2. Kaggle notebook reference – *The Good Place Episode Data Analysis* (bcruise)  
Used as a public reference for dataset context and exploratory analysis ideas.
3. pandas Documentation – `.merge()`, `.describe()`, `.isnull()`  
<https://pandas.pydata.org/docs/>
4. seaborn Documentation – `heatmap`, `barplot`  
<https://seaborn.pydata.org/>
5. matplotlib Documentation – `pyplot`  
[https://matplotlib.org/stable/api/pyplot\\_summary.html](https://matplotlib.org/stable/api/pyplot_summary.html)

### 3.8.1 AI Use Disclosure

This project was completed as part of a Python for Data Analytics coursework assignment.

AI tools were used to support: - Writing and formatting markdown explanations - Improving clarity and structure of narrative sections - Reviewing Python code structure and readability - Explaining data analysis concepts (e.g., correlation analysis, visual interpretation) - Suggesting best practices for exploratory data analysis

AI tools were **not** used to fabricate data, results, or conclusions.

All AI-assisted suggestions were critically reviewed, tested, and manually integrated by the author, who takes full responsibility for the final content and analysis.

(Full AI usage disclosure is provided in the repository documentation.)