##EPPS 6302 Data Collection Project, Fall 2024

# [PACKAGE CITATIONS FOR THIS PROJECT] #
# > citation("rvest")
# Wickham H (2024). _rvest: Easily Harvest (Scrape) Web Pages_. R package version 1.0.4, <https://CRAN.R-project.org/package=rvest>.
#
# > citation("dplyr")
#   Wickham H, François R, Henry L, Müller K, Vaughan D (2023). _dplyr: A Grammar of Data Manipulation_. R package version 1.1.4,
# <https://CRAN.R-project.org/package=dplyr>.
# > citation("chromote")
#   Chang W, Schloerke B, Aden-Buie G (2024). _chromote: Headless Chrome Web Browser Interface_. R package version 0.3.1,
# <https://CRAN.R-project.org/package=chromote>.
# > citation("httr")
#   Wickham H (2023). _httr: Tools for Working with URLs and HTTP_. R package version 1.4.7, <https://CRAN.R-project.org/package=httr>.
# > citation("jsonlite")
#   Ooms J (2014). "The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects." _arXiv:1403.2805 [stat.CO]_.
# <https://arxiv.org/abs/1403.2805>.
# > citation("tidyr")
#   Wickham H, Vaughan D, Girlich M (2024). _tidyr: Tidy Messy Data_. R package version 1.3.1, <https://CRAN.R-project.org/package=tidyr>.
# > citation("readr")
#   Wickham H, Hester J, Bryan J (2024). _readr: Read Rectangular Text Data_. R package version 2.1.5,
# <https://CRAN.R-project.org/package=readr>.
# > citation("tidyverse")
#   Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache
# SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the tidyverse."
# _Journal of Open Source Software_, *4*(43), 1686. doi:10.21105/joss.01686 <https://doi.org/10.21105/joss.01686>.
# > citation("stringr")
#   Wickham H (2023). _stringr: Simple, Consistent Wrappers for Common String Operations_. R package version 1.5.1,
#   <https://CRAN.R-project.org/package=stringr>.

```
#[STEP 1, part 1]#
#################################################################
###########[WEB SCRAPING FROM IMDB PAGE USING rvest]#############
#################################################################
# First Attempt: I am using rvest to scrape movies from the IMDb
install.packages("rvest")
install.packages("dplyr")
library(rvest)
library(dplyr)

# Define IMDb URL for scraping; in this case we were using IMDB
advance search showing us movies from 2022, movies that were in
English and had over 10,000 IMDB votes
url <-
"https://www.imdb.com/search/title/?title_type=feature&release_date=20
22-01-01,2022-12-31&num_votes=10000,&countries=US&languages=en"

# Read the HTML content of the IMDb page
webpage <- read_html(url)

# Extract movie titles from the specified HTML tag
titles <- webpage %>%
  html_nodes("h3") %>%
  html_text()

# Convert extracted titles to a data frame for easier handling
titles_df <- data.frame(Titles = titles)

# View the extracted titles
print(titles_df)

# Note: rvest captures only the first 25 results due to pagination,
requiring a workaround for additional data.
```

```
#[STEP 1, part 2]#
#################################################################
#############[PIVOT TO BOX OFFICE MOJO FOR MOVIE DATA]###########
#################################################################
# I looked at a list of movies for 'Domestic Box Office For 2022' on
the boxofficemojo.com page, filtered movies that were from 2022, in-
year releases only, and also wide releases (meaning they were shown in
more than 600 theaters)
# This is the URL for said movie list:
https://www.boxofficemojo.com/year/2022/?grossesOption=totalGrosses&re
leaseScale=wide
# This resulted in a list of 92 movies. I then inspected the element
to get the tags needed to scrape this data using the package rvest
# Items Gathered:
      # Movie title (as text) from the html div called 'td.mojo-field-
type-release'
      # Movie releaseID from the attached movie title link (as link)
from div called '.mojo-field-type-release a'
      ## Theaters indo from the div 'td.mojo-field-type-
positive_integer' (as text)

# Variables introduced:
# - release_ids:        Unique release IDs associated with each movie.
# - theaters:           Maximum number of theaters each movie was
shown in.
# - movies_df:          A data frame combining titles, release IDs,
and theater counts.

# Define the URL for the BoxOfficeMojo 2022 page
url <-
"https://www.boxofficemojo.com/year/2022/?grossesOption=totalGrosses&r
eleaseScale=all&sort=maxNumTheaters&ref_=bo_yld__resort#table"

webpage <- read_html(url)

# 1) Extract movie titles and their text
titles <- webpage %>%
  html_nodes("td.mojo-field-type-release a") %>%
  html_text(trim = TRUE)

# 2) Extract release IDs from the href attribute of the movie title
links
release_ids <- webpage %>%
  html_nodes("td.mojo-field-type-release a") %>%
  html_attr("href") %>%
  gsub("/release/", "", .) %>%
  gsub("/.*", "", .)
#  Used chat GPT here to figure out how we can optimize the web
scraping to clean up and only extract the linked id

# 3) Extract the maximum theater counts
theaters <- webpage %>%
```

```
  html_nodes("td.mojo-field-type-positive_integer") %>%
  html_text(trim = TRUE) %>%
  gsub(",", "", .) %>%
  as.numeric()

# Combine extracted data into a data frame, we used chat GPT here to
help format our scraped data in a way we could read and then use for
later data collection
movies_df <- data.frame(
  Title = titles,
  releaseID = release_ids,
  Max_Theaters = theaters,
  stringsAsFactors = FALSE
)

# Filter out movies with Max_Theaters less than 600 to give us a
better set of data, we will filter out more later on in this process
movies_df <- movies_df %>%
  filter(Max_Theaters >= 600)

write_csv(movies_df,
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movies_List_BoxOffice_
step1.csv")
```

```
#[STEP 2]#
#################################################################
####################[GETTING DATA FROM OMDB]#####################
#################################################################
library(httr)       # For API requests (This is the OMDB API Package)
library(jsonlite)   # For JSON parsing
library(dplyr)      # For data manipulation
library(tidyr)      # For reshaping data
library(readr)      # For reading/writing CSV files


# Variables introduced:
# - db_endpoint:                    The base URL for OMDb API
requests.
# - your_api_key:                   API key for OMDb.
# - fetch_movie_omdb:               A function to query OMDb API for
movie data.
# - movies_csv:                     A data frame containing movies
loaded from the BoxOfficeMojo CSV.
# - movies_data:                    A list of movie data fetched from
OMDb API.
# - movies_data_clean:              Cleaned and flattened version of
the fetched movie data.
# - movies_combined:                Combined and deduplicated movie
data.
# - movies_combined_with_extra:     Final data set with release ID and
theater count added.


# Define API endpoint and API key (Requested from the developer)
db_endpoint <- "https://www.omdbapi.com/?"
your_api_key <- "4b6ae90"


# Function to fetch movie data from OMDb API with year and type
filtering
# Used chatGPT here to help construct a valid function to grab the
data we needed within addation to using the omdb guidelines
fetch_movie_omdb <- function(title, year = NULL, type = NULL, plot =
"short", api_key) {
  # Construct the base URL
  url <- paste0(db_endpoint, "t=", URLencode(title), "&apikey=",
api_key)

  # Add optional parameters
  if (!is.null(year)) url <- paste0(url, "&y=", year)  # Filter by
year
  if (!is.null(type)) url <- paste0(url, "&type=", type)  # Filter by
type
  if (!is.null(plot)) url <- paste0(url, "&plot=", plot)  # Specify
plot length
```

```
  # Send the GET request
  response <- GET(url)

  # Check for HTTP errors
  if (status_code(response) != 200) {
    message("Error fetching data for: ", title, " - HTTP Status: ",
status_code(response))
    return(NULL)
  }

  movie_data <- content(response, as = "text", encoding = "UTF-8") %>%
    fromJSON(flatten = TRUE)   # Parse the JSON response

  if (movie_data$Response == "False") {
    message("No data found for: ", title, " - Error: ",
movie_data$Error)
    return(NULL)   # Check if the response is valid
  }

  return(as_tibble(movie_data))   # Convert to tibble
}

# Read movie titles and related data from a CSV file
movies_csv <-
read_csv("/Users/olivermyers/MyWebsite/EPPS_Project_data/Movies_List_B
oxOffice_step1.csv")
message("Movies loaded from CSV: ", nrow(movies_csv))

movies_csv <- movies_csv %>%
  distinct(Title, .keep_all = TRUE)
message("Movies after deduplicating original CSV: ", nrow(movies_csv))

# Fetch data for each movie in the CSV file, filtered by year 2022 and
type "movie"
movies_data <- lapply(movies_csv$Title, function(title) {
  fetch_movie_omdb(title = title, year = 2022, type = "movie", api_key
= your_api_key)
})

# Remove NULL entries (movies that failed the query)
movies_data_edit <- Filter(Negate(is.null), movies_data)
message("Movies after OMDb API fetch: ", length(movies_data_edit))

# Flatten the Ratings column and merge it back with the movie data
movies_data_clean <- lapply(movies_data_edit, function(movie) {
  if (!is.null(movie$Ratings) && is.data.frame(movie$Ratings)) {
    # Unnest Ratings into a flat format
    ratings_flat <- movie$Ratings %>%
      as_tibble() %>%
      pivot_wider(names_from = Source, values_from = Value,
names_prefix = "Rating_")
```

```
    # Combine Ratings with the rest of the movie data
    movie_flat <- movie %>%
      select(-Ratings) %>%  # Drop the original Ratings column
      bind_cols(ratings_flat)  # Add the flattened Ratings as new
columns

    return(movie_flat)
  } else {
    # If Ratings is NULL or not a data frame, return the movie as-is
    return(movie %>% select(-Ratings))
  }
})


# Used chatGPT for portions to help de-bug why there were was a large
loss of rows in our data and crafted output messeges to help identify
those errors
# Combine all processed movie data into a single data frame
movies_combined <- bind_rows(movies_data_clean) %>%
  distinct(Title, imdbID, .keep_all = TRUE)
message("Movies after combining and deduplicating data: ",
nrow(movies_combined))

# Add the releaseID and Max_Theaters from the original CSV to the
combined data
movies_combined_with_extra <- movies_combined %>%
  inner_join(movies_csv %>% select(Title, releaseID, Max_Theaters), by
= "Title")
message("Movies after joining releaseID and Max_Theaters: ",
nrow(movies_combined_with_extra))

# Remove rows with at least one valid rating
movies_combined_with_extra <- movies_combined_with_extra %>%
  filter(if_any(starts_with("Rating_"), ~ !is.na(.)))
message("Movies after filtering for at least one valid rating: ",
nrow(movies_combined_with_extra))

# Final deduplication by unique identifiers (e.g., Title, imdbID)
movies_combined_with_extra <- movies_combined_with_extra %>%
  distinct(Title, imdbID, .keep_all = TRUE)
message("Movies after final deduplication based on Title and imdbID:
", nrow(movies_combined_with_extra))

write_csv(movies_combined_with_extra,
"/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_combined_with_r
atings_step2.csv")
```

```
#[STEP 3]#
#####################################################################
########################[CLEANING MOVIE DATA]#######################
#####################################################################
install.packages("tidyverse")
install.packages("jsonlite")
library(tidyverse)
library(jsonlite)
library(dplyr)
#we needed to clean up the csv files we already had and re-format it
to be able to gather data from google trends later on which has
specfic parameters
# Variables introduced:
# - movies_clean_step:            A data frame containing movies and
their metadata loaded from the combined ratings CSV.
# - Released:                     Reformatted release date column in
YYYY-MM-DD format for consistency.
# - Released_21DaysLater:         A new column showing the release
date plus 21 days for Google Trends analysis.
# - movies_subset:                A cleaned and filtered subset of
movies with selected columns for easier analysis.
movies_clean_step <-
read_csv("/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_combin
ed_with_ratings_step2.csv")
View(movies_clean_step)

# First Convert "Released" column to YYYY-MM-DD format so we can feed
it into google trends later
movies_clean_step <- movies_clean_step %>%
  mutate(Released = as.Date(as.character(Released), format = "%d %b
%Y"))
View(movies_clean_step)

# Then Clean and format imdbVotes and BoxOffice columns for easier
analysis later on
movies_clean_step <- movies_clean_step %>%
  mutate(
    imdbVotes = formatC(as.numeric(gsub("[^0-9]", "", imdbVotes)),
format = "f", big.mark = ",", digits = 0), # Remove non-numeric
characters and format with commas
    BoxOffice = ifelse(!is.na(BoxOffice), paste0("$",
formatC(as.numeric(gsub("[^0-9]", "", BoxOffice)), format = "f",
big.mark = ",", digits = 0)), NA) # Format as currency
  )

# Create a new column that takes the newly formatted release column
and adds 21 days to Released for our later Google trends data analysis
movies_clean_step <- movies_clean_step %>%
  mutate(Released_21DaysLater = Released + 21) %>% # Add 21 days to
the release date
  relocate(Released_21DaysLater, .after = Released) # Move the new
column right after Released
```

```
# Filter rows where Language is "English" and Country is "United
States", this ensures we are getting our specified scope of movie
titles
movies_clean_step <- movies_clean_step %>%
  filter(
    str_detect(Language, "English"),  # Keep rows with "English" in
the Language column
    str_detect(Country, "United States")  # Keep rows with "United
States" in the Country column
  )
# Remove rows with NA in any of the ratings columns, we want to ensure
we can look at the moives ratings so this was a impotant must have
movies_clean_step <- movies_clean_step %>%
  filter(!if_any(starts_with("Rating_"), is.na)) # Remove rows with NA
in any column that starts with "Rating_"

# filted out movies that had less than 10,000 votes for the imdb
# Rename the Rotten Tomatoes column
movies_clean_step <- movies_clean_step %>%
  rename(RottenTomatoes_Rating = 'Rating_Rotten Tomatoes') %>%
  filter(is.na(Released) | format(Released, "%Y") != "2023") %>%
  filter(as.numeric(gsub(",", "", imdbVotes)) >= 10000)
# Create a subset with selected columns, we now want to make it easier
for us to look at with the title on the left with its dates ect
movies_subset <- movies_clean_step %>%
  select(
    Title,
    released_date = Released,
    endGTrends_date = Released_21DaysLater,
    imdbID,
    releaseID,
    Max_Theaters,
    BoxOfficeTotal = BoxOffice,
    imdbVotes,
    imdbRating,
    Metascore,
    RottenTomatoes_Rating,
    imdbRating,
    Genre,
    Runtime,
    Rated,
    Director,
    Awards,
    Language,
    Country
  )

# Save the filtered dataset to a new CSV file
write_csv(movies_subset,
"/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_filtered.csv")
```

```
#[STEP 4]#
####################################################################
####################################################################
###### [Get boxoffice earnings for each movie in our list ] #####
####################################################################
####################################################################
library(rvest)     # Web scraping
library(dplyr)     # Data manipulation
library(readr)     # Reading/writing CSV files
library(stringr)   # Helping clean up any spaces in CSV file
# we thought it would be helpful to ust go ahead and gather the daily
earnings from box office mojo because it has specific dates that we
could later match up with our google trends data for more intresting
or future analysis

# Variables introduced:
# - movies_dailey_earning:            A data frame containing cleaned
movie data loaded from the previous step.
# - release_ids:                      A list of release IDs extracted
from the data frame.
# - output_folder:                    The folder where daily earnings
data will be saved.
# - cleaned_table:                    A cleaned and formatted data
frame for the daily earnings of a movie.


movies_dailey_earning <-
read_csv("/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_filter
ed.csv")

# Extract the releaseID column as a list
release_ids <- movies_dailey_earning$releaseID

# Create a new folder for saving data if it doesn't exist
output_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movie_dailey_earning_A
LL_Data"
if (!dir.exists(output_folder)) {
  dir.create(output_folder)
}

# Loop through each released; this was very helpful to automate the
process and saved us a ton of time not having to go through each movie
for (release_id in release_ids) {
  # Construct the URL for the current releaseID
  url <- paste0("https://www.boxofficemojo.com/release/", release_id,
"/?ref_=bo_yld_table_1")

  # Try to scrape and process the table
  tryCatch({
    message("Processing: ", release_id) # message for progress to let
us know what releaseID it is on
```

```
    webpage <- read_html(url) # Reading in the current formed url

    table_data <- webpage %>%
      html_element("table.a-bordered.a-horizontal-stripes") %>%  # Use
the appropriate CSS selector for the table
      html_table()


    # Cleaning steps to get the data in, clean it up and format it
like we are wanting for each file, chatGPT was used to help craft this
block of code to get the function to work right.
    cleaned_table <- table_data %>%
      select(Date, DOW, Daily, Day) %>%  # Use the actual column names
from your table
      mutate(
        Date = gsub("[^a-zA-Z0-9 ]", "", Date),  # Remove special
characters from the Date column
        Date = paste0("2022 ", Date),  # Add the year to the date
        Date = as.Date(Date, format = "%Y %b %d"),  # Convert to YYYY-
MM-DD format
        Daily = gsub("\\$", "", Daily),  # Remove dollar signs from
Daily Earnings
        Daily = gsub(",", "", Daily),   # Remove commas from Daily
Earnings
        Daily = as.numeric(Daily),      # Convert Daily Earnings to
numeric
        Daily_Earning = paste0("$", formatC(Daily, format = "f",
big.mark = ",", digits = 0))  # Format as currency
      ) %>%
      select(-Daily) %>%  # Remove the old `Daily` column
      rename(Daily_Earning = Daily_Earning) %>%  # Rename the column
      filter(!is.na(Date))  # Remove rows where the Date could not be
parsed

    dataframe_name <- paste0(release_id, "_DS")

    output_path <- file.path(output_folder, paste0(dataframe_name,
".csv"))
    write_csv(cleaned_table, output_path)

    message("Successfully processed: ", release_id) # Print a message
for successful processing

  }, error = function(e) {
    message("Failed to process: ", release_id, " - ", e$message)
  })
}

message("Processing of all releaseIDs completed. Check the folder: ",
output_folder)
```

```
#[STEP 5]#
#####################################################################
####################[GOOGLE TRENDS DATA]#############################
#####################################################################
# This was the most complicated step as the google trends data threw
back many errors when we where trying to autmate this process
library(gtrendsR) #google trends API R-package
library(dplyr)
library(readr)

# Variables introduced:
# - FullSetInfo:                          Data frame containing
filtered movie data from the previous step.
# - csv_files:                            A list of existing Google
Trends CSV files.
# - successful_imdbIDs:                   IMDb IDs for which data has
already been processed.
# - movies_to_process:                    Movies yet to be processed.
# - process_movie_trends:                 Function to fetch and clean
Google Trends data for a single movie.


output_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/google_trends_data/"
if (!dir.exists(output_folder)) {
  dir.create(output_folder)
}

FullSetInfo <-
read_csv("/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_filter
ed.csv")

#Using chat gpt this section was made to help gather the files needed
to contintue processing the google trends as there was a lot of issues
regarding error 439 so we neededed a way to keep going though the list
of ungathered files
csv_files <- list.files(output_folder, pattern = "\\.csv$", full.names
= TRUE)

#Extract IMDb IDs by removing the "gt_" prefix and ".csv" suffix
successful_imdbIDs <- gsub("gt_|\\.csv", "", basename(csv_files))

# Filter FullSetInfo to exclude rows with IMDb IDs in the successful
list
movies_to_process <- FullSetInfo %>%
  filter(!imdbID %in% successful_imdbIDs)

# View movies left to process
View(movies_to_process)

# Function to fetch and process Google Trends data for one movie
process_movie_trends <- function(index, dataset, output_folder) {
```

```
  # Extract row data for the current index
  movie_row <- dataset[index, ]
  title <- movie_row$Title
  start_date <- movie_row$released_date
  end_date <- movie_row$endGTrends_date
  imdbID <- movie_row$imdbID

  # Log current movie being processed, this helped us view if there
was an issue in the automation
  message("Processing movie: ", title)

  tryCatch({
    trends_data <- gtrends(
      keyword = title,
      geo = c("CA", "US"),
      time = paste(start_date, end_date)
    )$interest_over_time

    # If data is successfully fetched
    if (!is.null(trends_data)) {
      cleaned_data <- trends_data %>%
        filter(geo != "CA") %>%             # Exclude rows where geo
== "CA", we had to include CA or else this loop didnt work, we tried
many ways, so we just deleted it after
        select(date, hits, keyword, geo)   # Keep only specified
columns, we only wanted to compare these feilts

      output_file <- paste0(output_folder, "gt_", imdbID, ".csv")
#adding the "gt_" helped us see which columns was from the "google
trends" fetched data
      write_csv(cleaned_data, output_file)
      message("Saved trends data to: ", output_file)
    } else {
      message("No data fetched for: ", title)
    }
  }, error = function(e) {
    message("Error processing movie: ", title, " - ", e$message)
  })
}

# Iterate through movies from 1 to nrow(movies_to_process), this
helped to ensure that the flow worked and would capture what we needed
for (n in 1:nrow(movies_to_process)) {
  print(c(n, movies_to_process$Title[n], movies_to_process$imdbID[n]))

  # Process the current movie
  process_movie_trends(n, movies_to_process, output_folder)

  # Add a delay to avoid rate-limiting, this was a constant issue so
we had to keep going throuhg the list several times
  Sys.sleep(20)
}
```

```
#[STEP 6]#
#####################################################################
#######
########### [ Combining Movie Money and Google Trends Data ]
###############
#####################################################################
#######
# We now wanted to craft a new dataset with all of the scraped data
from the dailey earnings, and google trends fetched data

# Variables introduced:
# - google_trends_folder:           Path to the folder containing
Google Trends data files.
# - daily_earnings_folder:          Path to the folder containing
daily earnings data files.
# - output_folder:                  Path to the folder where combined
data will be saved.
# - FullSetInfo:                    Data frame containing the final
list of movies with metadata.
# - combined_data:                  Data frame combining daily
earnings and Google Trends data for a movie.


# Define the input and output folder paths
google_trends_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/google_trends_data/"
daily_earnings_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movie_dailey_earning_A
LL_Data"
output_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movie_GT_Money_Combine
d/"
if (!dir.exists(output_folder)) {
  dir.create(output_folder)
}

# Read the FullSetInfo dataset, this is our origonal final long list
of movies with all of the attached meta data from omdb api
FullSetInfo <-
read_csv("/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_filter
ed.csv")

# Iterate over each movie in the FullSetInfo dataset
for (i in 1:nrow(FullSetInfo)) {
  # Extract movie details
  imdbID <- FullSetInfo$imdbID[i]
  releaseID <- FullSetInfo$releaseID[i]
  title <- FullSetInfo$Title[i]

  # Locate the corresponding files
  google_trends_file <- file.path(google_trends_folder, paste0("gt_",
imdbID, ".csv"))
```

```
  daily_earnings_file <- file.path(daily_earnings_folder,
paste0(releaseID, "_DS.csv"))

  # Check if both files exist
  if (file.exists(google_trends_file) &&
file.exists(daily_earnings_file)) {
    # Load the Google Trends data
    google_trends_data <- read_csv(google_trends_file, show_col_types
= FALSE) %>%
      rename_with(~ paste0("gt_", .)) %>%  # Add gt_ prefix to all
column names
      mutate(gt_date = as.Date(gt_date))  # Ensure date is in Date
format

    # Load the Daily Earnings data
    daily_earnings_data <- read_csv(daily_earnings_file,
show_col_types = FALSE) %>%
      mutate(date = as.Date(Date)) %>%  # Ensure date is in Date
format
      select(-Date)  # Remove the original Date column

    # Combine the datasets on the `date` column
    combined_data <- daily_earnings_data %>%
      left_join(google_trends_data, by = c("date" = "gt_date")) %>%
      mutate(
        imdbID = imdbID,    # Add imdbID column
        releaseID = releaseID,  # Add releaseID column
        Title = title       # Add Title column
      ) %>%
      filter(!is.na(gt_hits))  # Remove rows where gt_hits is NA

    # Save the combined data to the output folder
    output_file <- file.path(output_folder, paste0(imdbID, "_",
releaseID, ".csv"))
    write_csv(combined_data, output_file)
    message("Filtered combined data saved for movie: ", title, " -> ",
output_file)
  } else {
    # Log missing files
    if (!file.exists(google_trends_file)) {
      message("Google Trends data missing for: ", title, " (IMDb ID:
", imdbID, ")")
    }
    if (!file.exists(daily_earnings_file)) {
      message("Daily earnings data missing for: ", title, " (Release
ID: ", releaseID, ")")
    }
  }
}
```

```
########### [ Rearranging Columns for Combined Data Files ]
#################
#####################################################################
#######
# we wanted to have better clearity of our data so we preformed more
cleaning operations and rearrangment operations to format our combined
csv better

# Variables introduced:
# - combined_files:              List of all combined data files in
the input folder.
# - rearranged_data:             Data frame with rearranged columns
for better readability.


# Define the input and output folder paths
input_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movie_GT_Money_Combine
d/"
output_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movie_GT_Money_Combine
d/"  # Overwrite in the same folder

# Get a list of all combined files
combined_files <- list.files(input_folder, pattern = "\\.csv$",
full.names = TRUE)

# Iterate through each file and rearrange columns
for (file in combined_files) {
  # Read the combined data file
  combined_data <- read_csv(file, show_col_types = FALSE)

   # Rearrange columns in the specified order
  rearranged_data <- combined_data %>%
    select(
      Title,
      date,
      DOW,
      Day,
      Daily_Earning,
      gt_hits,
      gt_geo,
      gt_keyword,
      imdbID,
      releaseID
    )

  # Save the updated file back to the folder (overwriting the original
file)
  write_csv(rearranged_data, file)
  message("Updated file saved: ", file)
}
```

```
############ [ Combine All Files into One CSV ]
#####################################################################

# Variables introduced:
# - all_data:             A single data frame combining all movie
files.
# - output_file:          Path to save the final combined CSV file.


# Define the input and output folder paths
input_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Movie_GT_Money_Combine
d/"
output_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/All_Data_Combined/"
if (!dir.exists(output_folder)) {
  dir.create(output_folder)
}

# Define the path for the final combined CSV file
output_file <- file.path(output_folder, "All_Movies_Combined.csv")

# Get a list of all individual combined files
combined_files <- list.files(input_folder, pattern = "\\.csv$",
full.names = TRUE)

# Initialize an empty data frame to store the combined data
all_data <- NULL

# Iterate through each file and append data
for (file in combined_files) {
  # Read each file
  movie_data <- read_csv(file, show_col_types = FALSE)

  # Ensure column types are consistent across all files
  movie_data <- movie_data %>%
    mutate(
      Day = as.character(Day),               # Convert `Day` to
character, we where having issues here
      date = as.Date(date),                  # Ensure the `date`
column is in proper date format
      gt_hits = as.numeric(gt_hits)          # Convert `gt_hits` to a
numer value
    )

  # Handle `NA` values introduced during the conversion of `gt_hits`
to numeric
  if (any(is.na(movie_data$gt_hits))) {
    message("File with invalid gt_hits values: ", file)
  }
```

```
  # Append the data to the all_data data frame
  all_data <- bind_rows(all_data, movie_data)
}

# Save the combined data to the output file
write_csv(all_data, output_file)

# Print completion message
message("All combined data saved to: ", output_file)


#combined large file of all data to mess play with
library(readr)
All_Movies_Combined <-
read_csv("EPPS_Project_data/All_Data_Combined/All_Movies_Combined.csv"
)
View(All_Movies_Combined)



######################################################################
##### [ Combine All Data with Metadata from movies_filtered.csv ] ####
######################################################################

# Variables introduced:
# - combined_data_file:                 Path to the file containing
combined Google Trends and earnings data.
# - metadata_file:                      Path to the file containing
metadata for all movies.
# - final_combined_data:                Final data frame with metadata
merged into combined data.


# Define file paths
combined_data_file <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/All_Data_Combined/All_
Movies_Combined.csv"
metadata_file <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/movies_filtered.csv"
output_folder <-
"/Users/olivermyers/MyWebsite/EPPS_Project_data/Final_Combined_Data/"
if (!dir.exists(output_folder)) {
  dir.create(output_folder)
}

# Define the output file path
output_file <- file.path(output_folder, "Final_Combined_Data.csv")

# Load the combined Google Trends and earnings data
combined_data <- read_csv(combined_data_file, show_col_types = FALSE)

# Load the metadata
```

```
metadata <- read_csv(metadata_file, show_col_types = FALSE)

# Ensure both datasets have consistent column names for merging
# Here we assume the `imdbID` column is the common identifier
combined_data <- combined_data %>%
  mutate(imdbID = as.character(imdbID)) # Convert imdbID to character
for consistency

metadata <- metadata %>%
  mutate(imdbID = as.character(imdbID)) # Convert imdbID to character
for consistency

# Perform a left join: keep only rows in combined_data and add
metadata columns
final_combined_data <- combined_data %>%
  left_join(metadata, by = "imdbID") %>% # Join on imdbID
  relocate(starts_with("gt_"), .before = everything()) # Ensure Google
Trends columns are first

# Check if the "Title" column exists before attempting to relocate
if ("Title" %in% colnames(final_combined_data)) {
  final_combined_data <- final_combined_data %>%
    relocate(Title, .before = everything())
}
```

```
# we played around with the best readable format for us and this made
it better to identify what movie we were talking about for later
anlaysis
final_combined_data <- final_combined_data %>%
  select(
    Title = Title.y,
    date,
    DOW,
    Day,
    Daily_Earning,
    gt_hits,
    gt_geo,
    gt_keyword,
    imdbID,
    releaseID = releaseID.x,
    Max_Theaters,
    BoxOfficeTotal,
    imdbVotes,
    imdbRating,
    Metascore,
    RottenTomatoes_Rating,
    released_date,
    endGTrends_date,
    Genre,
    Runtime,
    Rated,
    Director,
    Awards,
    Language,
    Country
  )

# Save the final combined dataset to a CSV file
write_csv(final_combined_data, output_file)

# These files where then used in stata for further anlaysis, but this
ends the section on the data production and collection
```