WEEK 2 ASSIGNMENT REPORT

CYBER SHUJAA

ADM NO: CS-DA01-2507
COURSE: DATA AND AI
NAME: JEDIDAH WAVINYA MUSYOKA
DATE: 23rd MAY 2025
TASK: WEB WRANGLING

INTRODUCTION

- Platforms like Netflix collect and display a huge amount of data.
- From movie titles and genres to ratings, cast members, and release dates.
- While this data can tell interesting stories and help us make better decisions (like what to watch next), it doesn't always come in a clean, ready-to-use format.
- That's where data wrangling comes in.
- Data wrangling is the essential process of transforming and structuring raw, messy data into a clean, usable, and consistent format.
- The goal is to improve data quality and make it suitable for various downstream purposes, such as analysis, machine learning, or reporting.
- In this report, I walk through a data wrangling task using Python, focusing on a dataset of Netflix shows and movies.
- Using Python libraries such as Pandas and NumPy, I explored, cleaned, and prepared the Netflix dataset for analysis.
- This included handling missing values, correcting inconsistent data, converting data types, and more.
- The goal was to get the data into a state where it could be easily analyzed or visualized, whether to find trends in genres, compare release years, or understand Netflix's content distribution.

ASSIGNEMENT: DATA WRANGLING

OBJECTIVES

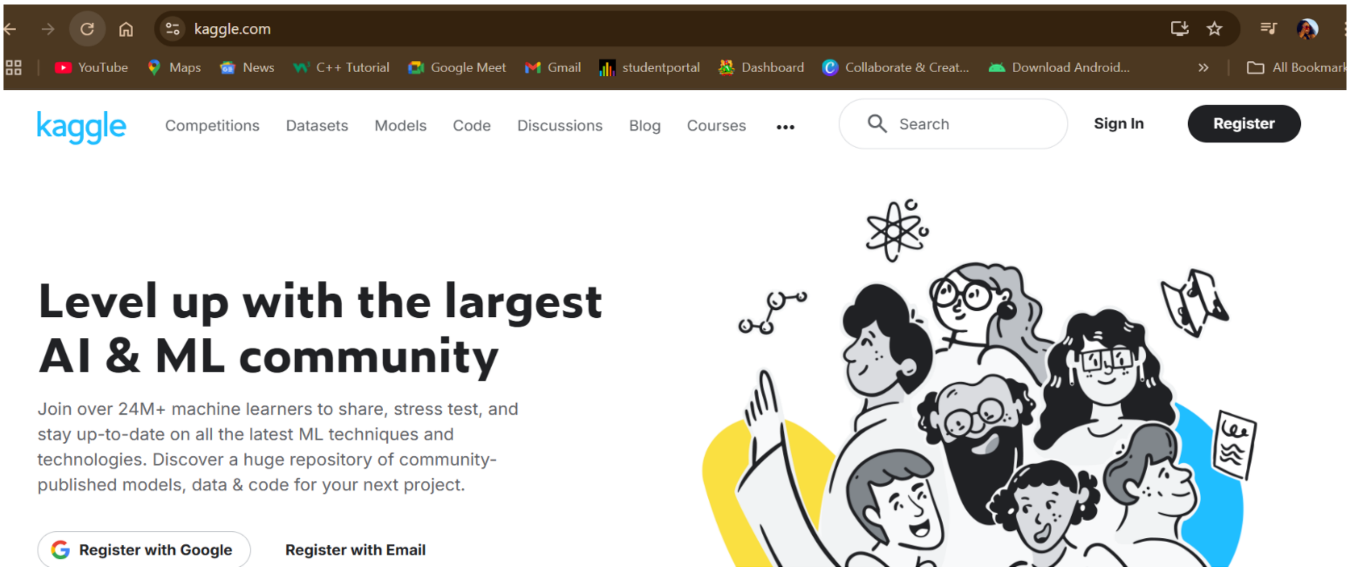
1) Load the Netflix dataset from a CSV file and explore its structure using pandas.
2) Perform data discovery to assess data types, missing values, and quality issues.
3) Clean the dataset by handling duplicates, missing values, and formatting inconsistencies.
4) Transform and enrich the dataset using techniques like filtering, sorting, grouping, and feature extraction.
5) Validate the final dataset by checking consistency, completeness, and logical accuracy.
6) Export the final cleaned dataset to a .csv file ready for analysis or visualization.

The following link is to the Netflix dataset that the data wrangling is going to take place on Kaggle

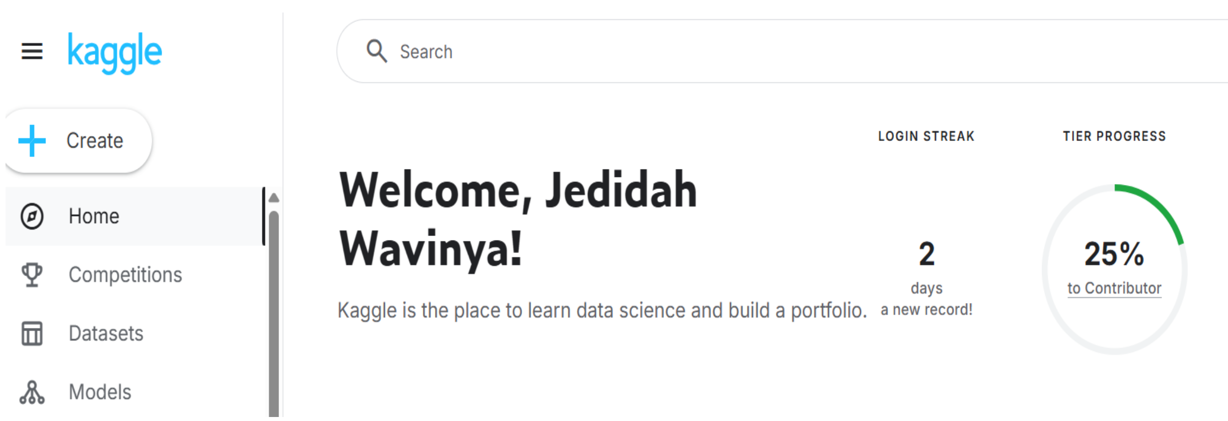
Link: https://www.kaggle.com/datasets/shivamb/netflix-shows

STARTING POINT

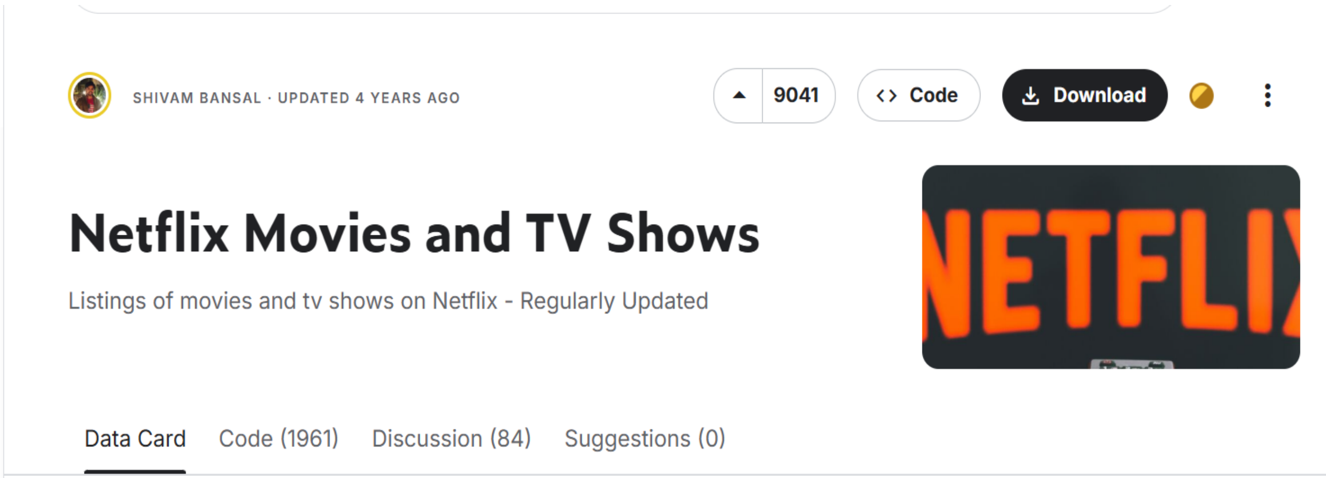
- Go to browser, search Kaggle.com as shown in the image below:



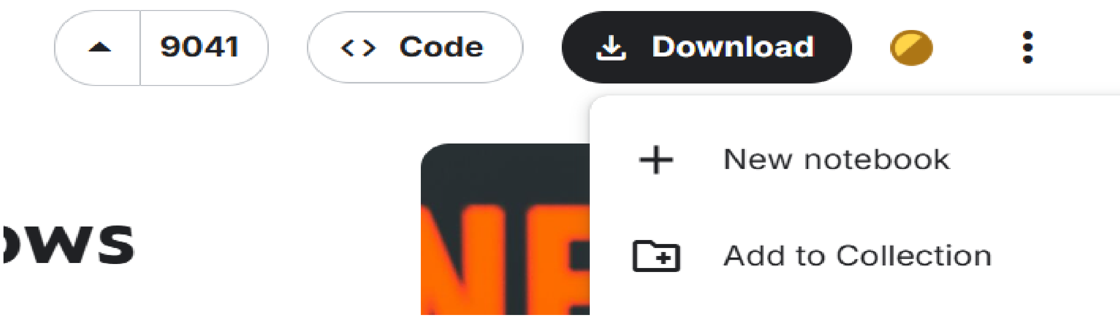
- Register with google
- This enables you to sign in with an email account.
- Once signed in to Kaggle, this redirection of page occurs as shown in the image below:



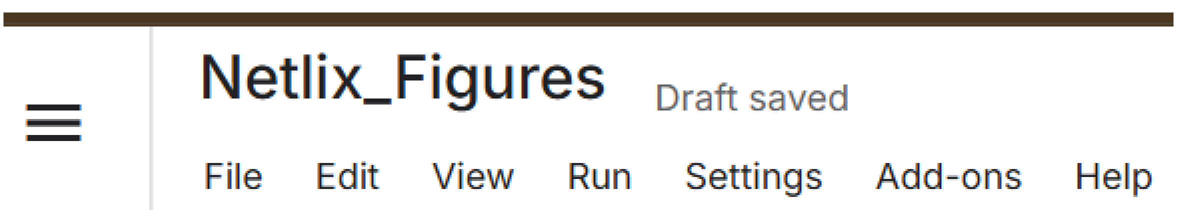
- Using the following link, https://www.kaggle.com/datasets/shivamb/netflix-shows to the dataset where the data wrangling and cleaning is to take place
- The image below shows the dataset we are using:



- Once I identified the correct dataset, I opened a new Notebook as you can see to the far right in the drop-down arrows:

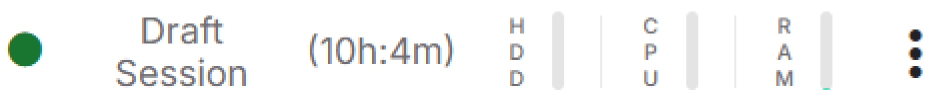


- The next step was to name the newly created Notebook, I named mine: “Netflix_Figures” as shown in the image below:



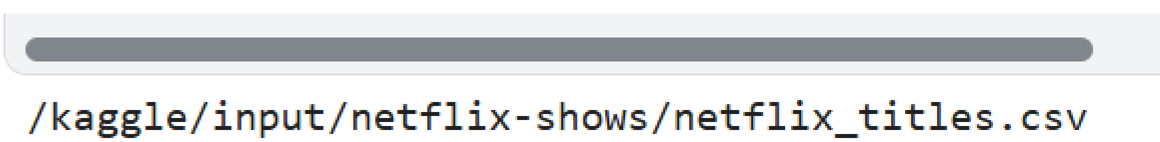
- Immediately, the following code environment is loaded. Everything is set up since it’s a programming environment using python.

- For this to run, I was allocated space, hosting disk space as seen at the top of the already-generated code environment, see below:

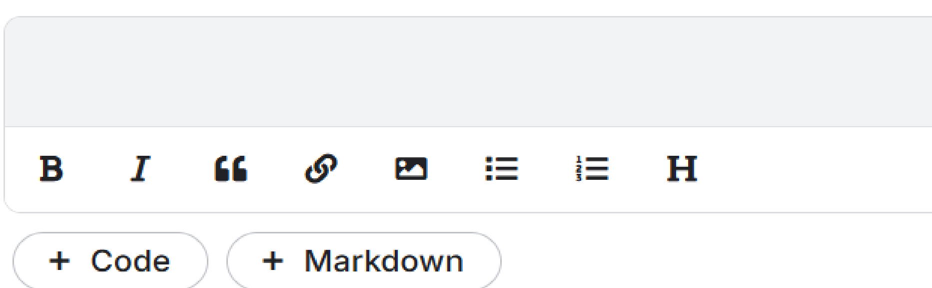


- Run the code

- The following output is portrayed:



- This show the path to where the file is saved.
- Note: In each code written, it is important to run the code.
- While writing code it is important to explain or give comments of what one is doing.
- In this case, I started off using a Markdown, as shown below:



- The details and comments about my project are as shown below:
- I described what my task included and the steps involved.

```
#   Title: Data Wrangling Project
  Name: Jedidah Wavinya
  Date: 20 May 2025

   This project demonstrates my walk through for data wrangling using python on Netflix.
  The steps that i will walk through are:
  1. Discovery to understand the data, its exixting format and quality issues to be addressed
  2. Structuring to understand the structure and standardize the formats.
  4. Cleaning
     * remove duplicates
     * remove irrelevant information
     * handle missing data
     * handle outliers
 1. Enriching
 2. Validating
 3. Publishing
```

- Since it's a Markdown, this is how it appeared after saving it:

Title: Data Wrangling Project

Name: Jedidah Wavinya Date: 20 May 2025

This project demonstrates my walk through for data wrangling using python on Netflix. The steps that i will walk through are:

1. Discovery to understand the data, its exixting format and quality issues to be addressed
2. Structuring to understand the structure and standardize the formats.
3. Cleaning
    - remove duplicates
    - remove irrelevant information
    - handle missing data
    - handle outliers

- Off to writing the code:

STEP 1: DISCOVERY

Step 1: Discovery

- Saved as a Markdown
- The next step as I started writing code, was to Import the data to a Panda DataFrame.

```python
#Import the data to a Panda DataFrame
df = pd.read_csv('/kaggle/input/netflix-shows/netflix_titles.csv')
```

- **df**: This is a variable name. In programming, variables are like containers that hold data.

- **pd**: This is a common alias for the pandas library.
- Before I could use pd.read_csv, I would typically have run this line earlier in my code as earlier seen:

```
import numpy as np # linear algebra, enables you to work with arrays an
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

- **read_csv**: This is a function provided by the pandas library.
- Its primary purpose is to read data from a Comma Separated Values (CSV) file and load it into a Pandas DataFrame.
- ('/kaggle/input/netflix-shows/netflix_titles.csv') is my CSB file.

- To have a quick overview of the data:

```
#Have a quick overview of the data
df.info()
```

- It gives you a fast and concise summary of your data without having to display the entire DataFrame.
- The output is as shown below:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   show_id       8807 non-null   object
 1   type          8807 non-null   object
 2   title         8807 non-null   object
 3   director      6173 non-null   object
 4   cast          7982 non-null   object
 5   country       7976 non-null   object
 6   date_added    8797 non-null   object
 7   release_year  8807 non-null   int64
 8   rating        8803 non-null   object
 9   duration      8804 non-null   object
 10  listed_in     8807 non-null   object
 11  description   8807 non-null   object
dtypes: int64(1), object(11)
memory usage: 825.8+ KB
```

- The range index has 8807 entries, that is from 0 to 8806.

- **df.shape** is an **attribute** (not a method, so no parentheses ()) of a Pandas DataFrame. It returns a **tuple** representing the dimensions of the DataFrame.
- See in the image below:

```
# Number of rows and columns
print("Shape of the dataset (R x C):", df.shape)
```

- print ("Shape of the dataset (R x C):", df.shape) displays the dimensions of my DataFrame.

- So, when you see output like (200, 10), it means your DataFrame has 200 rows and 10 columns.
- In my case the output is:

```
Shape of the dataset (R x C): (8807, 12)
```

- This means that the DataFrame I was currently working on had 8807 rows and 12 columns.

- The next step is to know the columns in the DataFrame

```python
# List of all column names
print("Columns in the dataset:\n", df.columns.tolist())
```

- This displays the names of all columns in my Pandas DataFrame.

- .columns: This is an **attribute** of a Pandas DataFrame.

- It returns a pandas.Index object (specifically a pd.Index or pd.MultiIndex if you have hierarchical columns) containing the labels (names) of all the columns in the DataFrame.

- .**tolist()**:This is a method called on the df.columns object (which is a pandas.Index).
- The .tolist() method converts the pandas.Index object into a standard Python **list**.
- This is often preferred for printing or further processing because a Python list is a very common and easy-to-work-with data structure.
- The following output is displayed:

```
Columns in the dataset:
 ['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added', 'release_ye
ar', 'rating', 'duration', 'listed_in', 'description']
```

- To know the data types in each column:

```python
# Data types of each column
print("Data types:\n", df.dtypes)
```

- "Data types:\n": This is a string literal that will be printed as a label. The \n is a newline character, ensuring the list of data types starts on the next line for better readability.
- .dtypes: This is an attribute (not a method, so no parentheses ()) of a Pandas DataFrame.
- It returns a Pandas Series where:
  - The index of the Series contains the column names of your DataFrame.
  - The values of the Series are the data types (dtypes) of those respective columns.

- The following Output is displayed:

```
Data types:
 show_id          object
type             object
title            object
director         object
cast             object
country          object
date_added       object
release_year      int64
rating           object
duration         object
listed_in        object
description      object
dtype: object
```

- When you see a data type as object, it often signifies that it can hold any Python object. This includes strings, numbers (integers, floats), lists, dictionaries, custom class instances, and even other complex data structures.

- The next step is to get a precise count of missing values for each column in my Pandas DataFrame.
- This is an absolutely critical step in any data cleaning and preprocessing workflow.

- See the image code below:

```
# Group and Count of missing (null) values in each column
print("Missing values per column:\n", df.isnull().sum())
```

- .isnull(): This is a **method** called on the DataFrame.
    - It performs an element-wise check on every single cell in the DataFrame.
    - It returns a new DataFrame of the **same shape** as your original df, but filled entirely with **Boolean** values (True or False).
    - A True indicates that the corresponding cell in the original df contains a missing value (e.g., NaN, None), and False indicates that the cell has a valid value.

- See the image below as the output.

- The missing values per column are displayed for instance the column with the column head “director” has 2634 missing values.

```
Missing values per column:
 show_id            0
type                0
title               0
director         2634
cast              825
country           831
date_added         10
release_year        0
rating              4
duration            3
listed_in           0
description         0
dtype: int64
```

- The next step is to count the number of duplicate rows in your Pandas DataFrame.
- This is an essential step in data cleaning, as duplicate records can skew analyses and lead to incorrect conclusions.
- See the image below:

```
# Group and Count of duplicate rows
print("Number of duplicate rows:", df.duplicated().sum())
```

- .duplicated(): This is a **method** called on the DataFrame. It performs a row-wise check to identify duplicate rows.
- By default, duplicated() considers a row a duplicate if it's identical to a previously encountered row. It marks the *second and subsequent* occurrences of a duplicate row as True. The *first* occurrence of a row (even if it's duplicated later) is marked as False.
- Run the code

- See the output below after running the cell code.

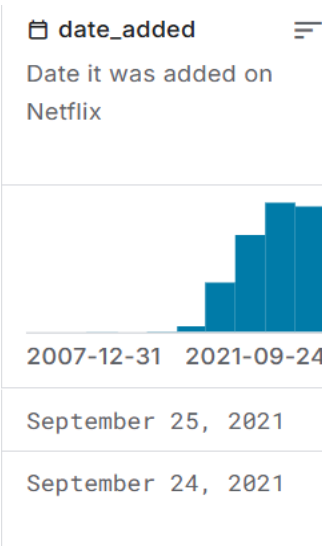
```
Number of duplicate rows: 0
```

STEP 2: STRUCTURING

- This step involves normalizing and standardizing the data format.
- It also involves transforming data and converting from one format to another.
- I included this step in a Head Markdown
- See below:

Step 2:Structuring

- In the original dataset, the “Date_added” column has its rows in the following format:



- I needed to convert that format into datetime format.
- See in the image below:

```
# Convert 'date_added' to datetime
df['date_added'] = pd.to_datetime(df['date_added'],format='mixed')
```

- This is a very common and critical step in data cleaning and preparation, especially when working with date and time data that might be stored as strings.

- .to_datetime(): This is a powerful Pandas function specifically designed for converting various types of input (strings, integers, floats, other datetime objects) into Pandas Timestamp objects (which are essentially datetime objects optimized for Pandas).

- The next step is to separate the ‘duration’ into numeric value and unit

- See the image below:

```
# Separate 'duration' into numeric value and unit (e.g., '90 min' → 90, 'min')
df[['duration_value', 'duration_unit']] = df['duration'].str.extract(r'(\d+)\s*(\w+)')
```

- The above line of code is a brilliant example of using **regular expressions** with Pandas to extract structured information from a string column.

- df['duration']: This selects the 'duration' column from your DataFrame df.

- .str: This is the Pandas StringAccessor.

- When you use .str on a Series, it allows you to apply string methods (like extract, contains, lower, split, etc.) to every string element in that Series.

- It's similar to applying str.extract() in pure Python, but optimized for Pandas Series/DataFrames.

- The next step is to convert the ‘duration_value’ to numeric.
- See in the image below:

```python
# Convert duration_value to numeric
df['duration_value'] = pd.to_numeric(df['duration_value'])
```

- Run the code!
- To view the resulting columns after the changes have been made(i.e. normalizing and standardizing the data formats)

```python
# View Resulting columns
print(df[['duration_value', 'duration_unit']])
```

- The following output is displayed.

```
      duration_value duration_unit
0               90.0           min
1                2.0       Seasons
2                1.0        Season
3                1.0        Season
4                2.0       Seasons
...              ...           ...
8802           158.0           min
8803             2.0       Seasons
8804            88.0           min
8805            88.0           min
8806           111.0           min

[8807 rows x 2 columns]
```

- The data has been restructured.

STEP 3: CLEANING

- Cleaning involves removing duplicates or redundant data, elimination of errors, handling missing values, removing irrelevant information, handling outliers and ensuring high quality data.
- Start off by adding information about this step in a Markdown.
- See in the image below:

```
# Step 3: Cleaning
```

- First, check for any duplicate rows:

```
# Check for duplicate rows
print("Duplicate rows before:", df.duplicated().sum())
```

- Run the code!
- The following output is displayed.

```
Duplicate rows before: 0
```

- If there were any duplicate rows, the following code would apply:

```
# Drop duplicate rows if any
df = df.drop_duplicates()
```

- This would drop them.
- To drop the description column,

```
# Drop description column because it will not be used
df = df.drop(columns=['description'])
```

- **df.drop()**: This is a Pandas DataFrame method used to remove rows or columns.
- **columns=['description']**: This argument tells the drop() method that you want to remove columns, and specifically, the column named 'description'. You pass the column names as a list, even if there's only one.

- To Impute Director values by using relationship between cast and director:

```
# List of Director-Cast pairs and the number of times they appear
df['dir_cast'] = df['director'] + '---' + df['cast']
counts = df['dir_cast'].value_counts() #counts unique values
print(counts)
```

- In order to view the output, I used the “print(counts)
- Run the code!

- The following output is displayed proving that it paired the ‘dir_cast’ to the ‘cast’ for all rows.

```
dir_cast
Rajiv Chilaka---Vatsal Dubey, Julie Tejwani, Rupa Bhimani, Jigna Bhardwaj, Rajesh Kava, Mousam, Swapnil
12
Rathindran R Prasad---Aishwarya Rajesh, Vidhu, Surya Ganapathy, Madhuri, Pavel Navageethan, Avantika Vandanapu
4
S.S. Rajamouli---Prabhas, Rana Daggubati, Anushka Shetty, Tamannaah Bhatia, Sathyaraj, Nassar, Ramya Krishnan, Sudeep
4
Louis C.K.---Louis C.K.
3
Stan Lathan---Dave Chappelle
3
```

- To check if the 'counts' are repeated 3 or more times:
- See the image below:

```
filtered_counts = counts[counts >= 3] #checks if repeated 3 or more times
```

```
filtered_counts = counts[counts >= 3] #checks if repeated 3 or more times
filtered_values = filtered_counts.index #gets the values i.e. names
lst_dir_cast = list(filtered_values) #convert to list
```

- This gets the values of the repeated.

- To fill in missing 'director' values in a DataFrame (df) based on a mapping derived from the list of director-cast combinations (lst_dir_cast), the following code applies:

```
dict_direcast = dict()
for i in lst_dir_cast :
    director,cast = i.split('---')
    dict_direcast[director]=cast
for i in range(len(dict_direcast)):
    df.loc[(df['director'].isna()) & (df['cast'] == list(dict_direcast.items())[i][1]),'director'] = list(dict_direcast.item
```

- To assign "Not Given" to all other director fields:

```
# Assign Not Given to all other director fields
df.loc[df['director'].isna(),'director'] = 'Not Given'
```

- Run the code.

- To fill the missing countries, the following code applies as shown in the image below:

```
#Use directors to fill missing countries
directors = df['director']
countries = df['country']
```

- Run the code!
- To drop the row records that are null, the following code applies as shown:

```python
# dropping other row records that are null
df.drop(df[df['date_added'].isna()].index,axis=0,inplace=True)
df.drop(df[df['rating'].isna()].index,axis=0,inplace=True)
df.drop(df[df['duration'].isna()].index,axis=0,inplace=True)
```

- Run the code.

STEP 4: ENRICHING

- In this step of data wrangling, it is important to decide if there is enough data or if need to seek additional interior or third party sources.
- Repeat the previous steps for any new data.
- I attempted to have another column that just has the 'release_month'
- I applied the following code as shown in the image below:

```python
#Attempting to have a column that just has the 'release_month'
# Assuming df['date_added'] is already in datetime format

df['release_month'] = df['date_added'].dt.month
print(df)
```

- Run the code.
- It does give the release month but in numeric value ie, for the month of September, it portrayed the value 9
- See the output below:

```
      release_month
0                 9
1                 9
2                 9
3                 9
4                 9
...             ...
8802             11
8803              7
8804             11
8805              1
8806              3
```

STEP 5: VALIDATING

- Conducted tests to check data accuracy, quality and consistency
- Checked the completeness of the data.
- Ensured each column has the correct data type e.g. verify that date_added is datetime and duration_value is numeric.
- Ensured no important fields are still missing
- Sample a few rows to check visually
- Reset the Index e.g. df_reset = df.reset_index(drop=True)

STEP 6: PUBLISHING

- Export and distribute.
- Make the wrangled data available by saving it.
- See in the image below:

```
# Save as CSV
df.to_csv('/kaggle/working/cleaned_netflix.csv', index=False)
# Save as Excel
df.to_excel('/kaggle/working/cleaned_netflix.xlsx', index=False)
#Save as JSON
df.to_json('/kaggle/working/cleaned_netflix.json', orient='records',lines=True)
```

- Run the code!
- The csv file is saved in this specific Kaggle notebooks/environments
- In: /kaggle/working/cleaned_netflix.csv

- End of task.

Link to the Kaggle Notebook for the above data wrangling task:
https://www.kaggle.com/code/jedidahwavinya/netflix-figures

CONCLUSION

- In conclusion, working with the Netflix shows and movies dataset has shown just how important data wrangling is in the data analysis process.
- Before any meaningful insights can be drawn, the data must first be cleaned, organized, and structured properly.
- Through this task, I was able to identify and handle missing values, fix inconsistencies, and convert data into appropriate formats using Python tools like Pandas.
- This process not only improved the quality of the dataset but also made it easier to explore trends, patterns, and relationships within the data.
- Whether someone wants to analyze popular genres, track content over the years, or look at country-wise production, having a clean dataset is a crucial first step.
- Though I spent so much time working on it, I have gained new skills on the same and I can say it was all worth it.
- This project was a great hands-on way to understand all the concepts of data wrangling and why each one was important.
  ~Thank you.