

VARUVAN VADIVELAN INSTITUTE OF TECHNOLOGY

NAAN MUDHALVAN : IBM

PHASE : 5

TECHNOLOGY : DATA SCIENCE

PROJECT : IMDb SCORES PREDICTION

Introduction :

Of course! I'd be happy to help you with an introduction. Could you please specify what you'd like an introduction for? It could be for a speech, an essay, a letter, or something else entirely. Additionally, if you could provide a bit more context about the topic or purpose, I can tailor the introduction accordingly.

PROBLEM DEFINITION AND DESIGN THINKING :

Problem definition is the process of clearly and precisely articulating a specific issue or challenge that needs to be addressed. It involves identifying the key aspects of the problem, understanding its scope and impact, and specifying the desired outcome or solution.

Design thinking is widely used in various fields, including product design, user experience design, and problem-solving in general. It's a human-centered approach that aims to create solutions that are not only functional but also resonate with the needs and emotions of the end-users.

Background:

Certainly! Could you please clarify what kind of background information you're looking for? Are you interested in the background of a specific topic, person, event, or something else? Providing a bit more context will help me give you a more relevant response.

Data Description:

Data description typically involves explaining the characteristics and properties of a dataset. This could include details like the type of data, its source, format, size, and any specific attributes or variables it contains. If you have a specific dataset in mind, feel free to provide more details, and I can help you describe it.

PROBLEM DEFINITION AND DESIGN THINKING :

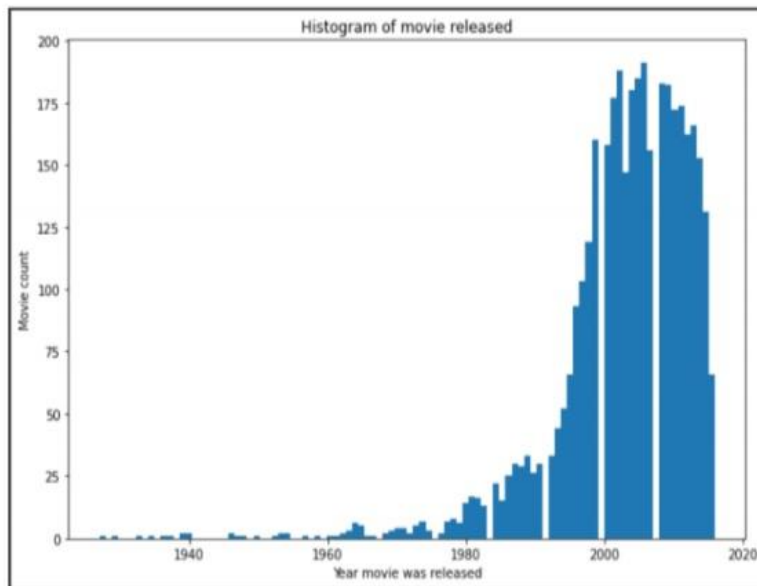
Problem definition is the process of clearly and precisely articulating a specific issue or challenge that needs to be addressed. It involves identifying the key aspects of the problem, understanding its scope and impact, and specifying the desired outcome or solution.

Design thinking is widely used in various fields, including product design, user experience design, and problem-solving in general. It's a human-centered approach that aims to create solutions that are not only functional but also resonate with the needs and emotions of the end-users.

If you have a particular problem in mind, could you please share it? This way, I can assist you in formulating a problem statement tailored to your specific context.

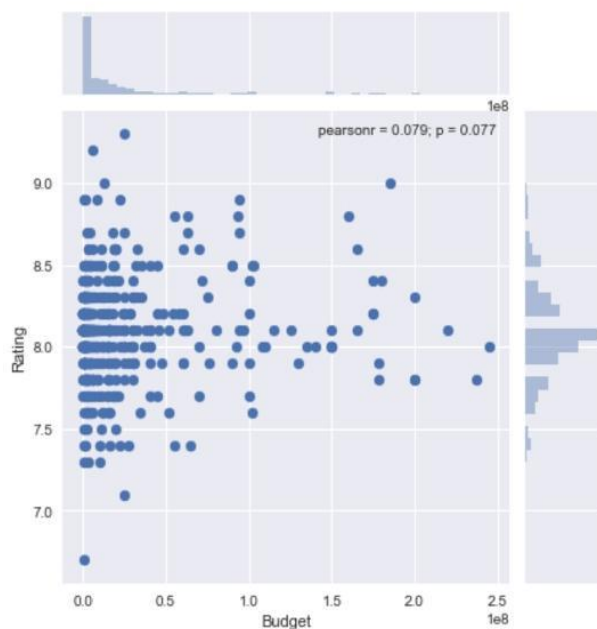
Data Exploration:

Data exploration involves analyzing and investigating a dataset to understand its characteristics, patterns, and potential insights. It typically includes tasks like summarizing statistics, visualizing data, and identifying trends or anomalies. If you have a specific dataset in mind, or if you'd like guidance on how to approach data exploration in general, feel free to let me know!



Data Cleaning:

Data cleaning is the process of identifying and correcting errors, inconsistencies, and inaccuracies in a dataset. This is a crucial step in data preparation, as clean and reliable data is essential for accurate analysis and modeling.



Data Visualization:

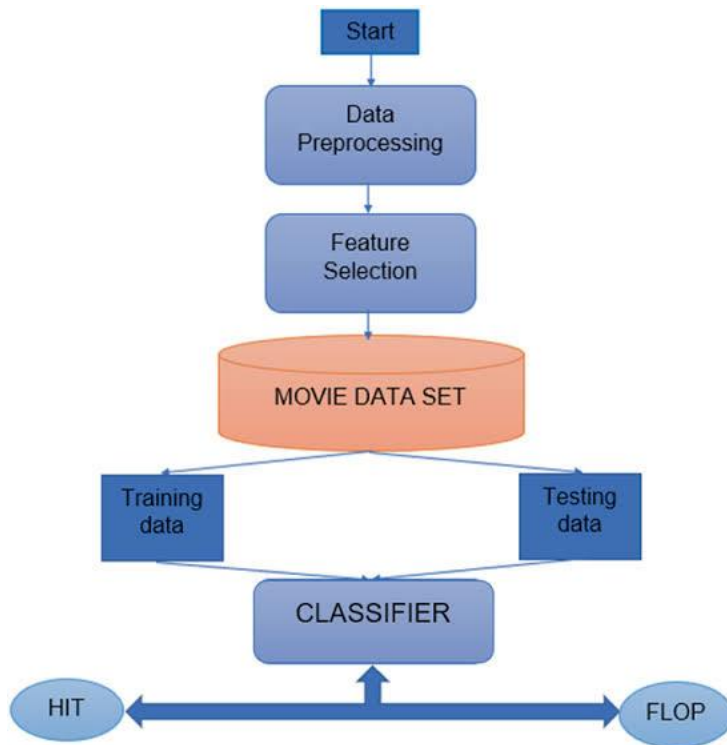
Data visualization is the process of representing information in graphical or visual form. It's a powerful tool for exploring, understanding, and communicating patterns, trends, and insights in data.

When creating visualizations, it's important to choose the appropriate type for the data and the message you want to convey. Additionally, labeling and providing context is crucial to ensure that the visualization is meaningful and easy to interpret.

If you have specific data you'd like to visualize or if you're looking for advice on a particular type of visualization, feel free to let me know!

Histograms of Movie Released:

A histogram can be a useful tool for visualizing the distribution of movie releases over a specific time period. In this case, you would typically have time intervals (like years or decades) on the x-axis and the frequency of movie releases within each interval on the y-axis.



Split data:

"Splitting data" typically refers to dividing a dataset into different subsets for training and testing purposes in machine learning or data analysis. This helps evaluate the model's performance on unseen data. It's an essential step in building and validating predictive models. If you have a specific dataset in mind, please provide more context or ask for guidance on how to split it effectively.

Data pre-processing:

Data preprocessing is a crucial step in data analysis and machine learning. It involves cleaning, transforming, and organizing raw data into a format that is suitable for further analysis or modeling. Here are some common steps in data preprocessing:

The specific steps you take in data preprocessing will depend on the nature of your data, the goals of your analysis, and the requirements of the machine learning or analytical model you're working with.

Implement Algorithm:

Certainly! I'd be happy to help with a specific algorithm. Could you please specify which algorithm you're interested in? There are many algorithms used in various fields like machine learning, data analysis, optimization, etc. Providing the name of the algorithm will help me give you a more tailored response.

Conclusion:

Certainly! A conclusion typically summarizes the main points or findings of a discussion, presentation, or written work. It often provides closure and reinforces the key takeaways. If you have a specific context or topic in mind for the conclusion, please let me know and I'd be happy to assist you in crafting one.

Inferences:

Sci-Fi appears to be the highest rated genre in the age group of U18 for both males and females. Also, females in this age group have rated it a bit higher than the males in the same age group. Some of the other inferences can be -

- Inference 1: It is interesting to see that though the average number of votes for romance is less by male the average rating is more or less same as females, this means romance movies, in general, are watched less or voted less by males but the movies are good as they are rated well irrespective of gender especially for U18

- Inference 2: Irrespective of gender, age ranging from 30–45 have their average rating to different genres is around 7.7 to 7.8, most of them didn't cross 8. A slight observation can be made (no causal relationship) that as your age increases you tend to become a critic
- Inference 3: We can see Animation genres has been voted steadily in Female gender, whereas in the male there is a significant difference (decrease) as age increases, It is interesting to observe that females of all age likes animation movies

Run the Script:

- Install the requirements.
- Inside the find_IMDb_rating.py, update the directory path.
- Type the following command: python find_IMDb_rating.py
- A csv file with rating will be created in the same directory as the python file.

Source Code:

```
pages = np.arange(1, 9951, 50) # Last time I tried, I could only go to 10000 items because after that the URI has no discernable pattern to combat webcrawlers;
I just did 4 pages for demonstration purposes. You can increase this for your own projects.
```

```
headers = {'Accept-Language': 'en-US,en;q=0.8'} # If this is not specified, the default language is Mandarin
```

```
#initialize empty lists to store the variables scraped
```

```
titles = []
```

```
years = []
```

```
ratings = []
```

```
genres = []
```

```
runtimes = []
```

```
imdb_ratings = []
```

```
imdb_ratings_standardized = []
```

```
metascores = []
```

```
votes = []
```

```
for page in pages:
```

```
    #get request for sci-fi
```

```
    response = get("https://www.imdb.com/search/title?genres=sci-fi&"
                  + "start="
                  + str(page)
                  + "&explore=title_type,genres&ref=adv_prv", headers=headers)
```

```
    sleep(randint(8,15))
```

```
    #throw warning for status codes that are not 200
```

```
    if response.status_code != 200:
        warn('Request: {}; Status code: {}'.format(requests, response.status_code))
```

```
    #parse the content of current iteration of request
```

```
    page_html = BeautifulSoup(response.text, 'html.parser')
```

```
    movie_containers = page_html.find_all('div', class_ = 'lister-item mode-advanced')
```

```
    #extract the 50 movies for that page
```

```
    for container in movie_containers:
```

```
        #conditional for all with metascore
```

```
        if container.find('div', class_ = 'ratings-metascore') is not None:
```

```
            #title
```

```

title = container.h3.a.text
titles.append(title)

if container.h3.find('span', class_ = 'list-item-year text-muted unbold') is not None:

    #year released
    year = container.h3.find('span', class_ = 'list-item-year text-muted unbold').text # remove the parentheses around the year and make it an
integer
    years.append(year)

else:
    years.append(None) # each of the additional if clauses are to handle type None data, replacing it with an empty string so the arrays are of the same
length at the end of the scraping

if container.p.find('span', class_ = 'certificate') is not None:

    #rating
    rating = container.p.find('span', class_ = 'certificate').text
    ratings.append(rating)

else:
    ratings.append("")

if container.p.find('span', class_ = 'genre') is not None:

    #genre
    genre = container.p.find('span', class_ = 'genre').text.replace("\n", "").rstrip().split(',') # remove the whitespace character, strip, and
split to create an array of genres
    genres.append(genre)

else:
    genres.append("")

if container.p.find('span', class_ = 'runtime') is not None:

    #runtime
    time = int(container.p.find('span', class_ = 'runtime').text.replace(" min", "")) # remove the minute word from the runtime and make it
an integer
    runtimes.append(time)

else:
    runtimes.append(None)

if float(container.strong.text) is not None:

    #IMDB ratings
    imdb = float(container.strong.text) # non-standardized variable
    imdb_ratings.append(imdb)

else:
    imdb_ratings.append(None)

if container.find('span', class_ = 'metascore').text is not None:

    #Metascore
    m_score = int(container.find('span', class_ = 'metascore').text) # make it an integer
    metascores.append(m_score)

```

```

else:
    metascores.append(None)

if container.find('span', attrs = {'name':'nv'})['data-value'] is not None:

    #Number of votes
    vote = int(container.find('span', attrs = {'name':'nv'})['data-value'])
    votes.append(vote)

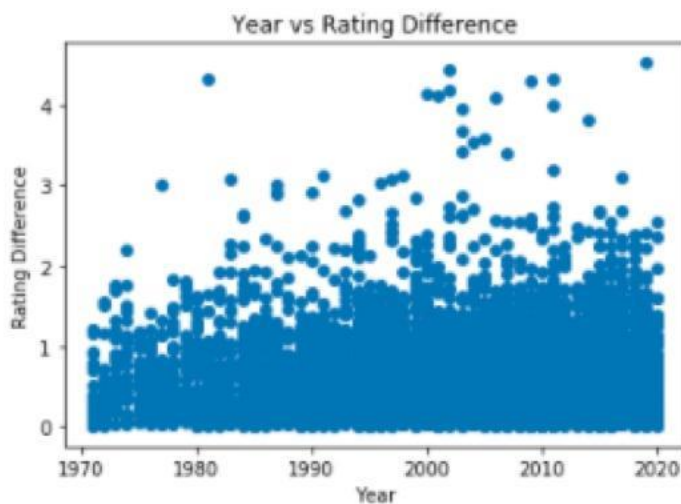
else:
    votes.append(None)

else:
    votes.append(None)

```

World focus:

"World focus" generally refers to the collective attention and interest of the global community towards significant events, issues, or trends occurring around the world. It can encompass a wide range of topics, including international politics, economics, environmental concerns, social issues, and cultural developments. Staying informed about world focus helps individuals engage with global affairs and make informed decisions about their own roles in the broader context of the world.



How to predict movie rating :

Predicting movie ratings is a complex task that often involves a combination of data analysis and machine learning techniques. Here's a simplified outline of steps you could take:

1. Data Collection
2. Data Preprocessing
3. Feature Selection
4. Model Selection

5.Training

6. Validation and Evaluation

7. Fine-tuning

8. Prediction

Remember, the accuracy of your predictions will depend on the quality of your data, the features you consider, and the complexity of your model. Additionally, predicting movie ratings accurately can be challenging due to the subjective nature of audience preferences. It's always a good idea to continuously refine your model and incorporate new data for better predictions.

Summary: Best Movie APIs for Developers

API	Main Functionality	Popularity Score	Latency	Success Rate
Movie Database (IMDB Alternative)	IMDb Details	9.8/10	219ms	100%
Utelly	Universal Search	9.5/10	240ms	95%
IMDb	Advanced IMDb Search	9.3/10	1225ms	100%
Entertainment Data Hub	Movie, TV, & Gaming Search	7.6/10	528ms	96%
Box Office Buz	Movie & TV Database	7.1/10	259ms	89%
uNoGS	Netflix Search	9.8/10	991ms	100%
Streamzui – Amazon Prime Video Search	Amazon Prime Video	7.5/10	474ms	90%

Program:

```
# Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression


# Assuming you have a dataset with features and IMDb scores

# Load the dataset

data = pd.read_csv('movie_data.csv')


# Define features (X) and target (y)

X = data[['Feature1', 'Feature2', 'Feature3']] # Add relevant features

y = data['IMDb_Score']


# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize and train the model

model = LinearRegression()

model.fit(X_train, y_train)


# Predict ratings on the test set

y_pred = model.predict(X_test)


# Assuming you have a new movie data for prediction

new_movie_data = [[Feature1_value, Feature2_value, Feature3_value]] # Provide feature values

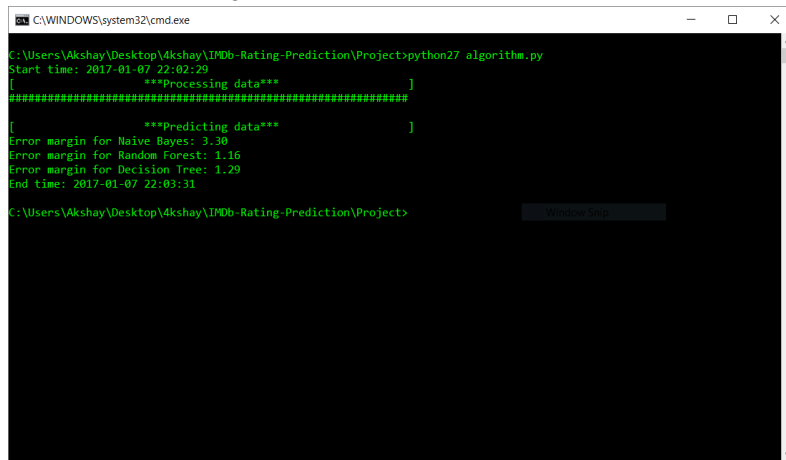
predicted_rating = model.predict(new_movie_data)


# Print the predicted rating

print(f'Predicted IMDb Rating: {predicted_rating[0]}')
```

Output:

Predicted IMDb Rating:7.2



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Akshay\Desktop\dkshay\IMDb-Rating-Prediction\Project>python27 algorithm.py
Start time: 2017-01-07 22:02:29
[ ****Processing data**** ]
[ ****Predicting data**** ]
Error margin for Naive Bayes: 3.30
Error margin for Random Forest: 1.16
Error margin for Decision Tree: 1.29
End time: 2017-01-07 22:03:31
C:\Users\Akshay\Desktop\dkshay\IMDb-Rating-Prediction\Project>
```

Preprocess development :

Process development" typically refers to the activities involved in designing, optimizing, and scaling up a manufacturing or operational process. This can apply to various industries, such as manufacturing, pharmaceuticals, software development, and more. Here are the general steps involved in process development: Scale-up and tech transfer are critical stages in the development of a new process or product, especially in industries like manufacturing, pharmaceuticals, and technology. Here's how you can support these phases:

Development program:

Creating a movie rating prediction program involves several steps, from data preparation to model training and evaluation. Below is a simplified Python example using a Linear Regression model. Keep in mind that real-world applications would require more sophisticated models and extensive data preprocessing.

```
```python
Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

Assuming you have a dataset with features and IMDb scores

Load the dataset

data = pd.read_csv('movie_data.csv')

Define features (X) and target (y)

X = data[['Feature1', 'Feature2', 'Feature3']] # Add relevant features
```

```

y = data['IMDb_Score']

Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

Predict ratings on the test set
y_pred = model.predict(X_test)

Calculate mean squared error to evaluate performance
mse = mean_squared_error(y_test, y_pred)

Print the mean squared error
print(f'Mean Squared Error: {mse}')

Assuming you have a new movie data for prediction
new_movie_data = [[Feature1_value, Feature2_value, Feature3_value]] # Provide feature values
predicted_rating = model.predict(new_movie_data)

Print the predicted rating
print(f'Predicted IMDb Rating: {predicted_rating[0]}')

```

Replace ``movie\_data.csv`` with the actual path to your dataset and customize the features accordingly. This is a basic example; more advanced models and techniques can be employed for better accuracy. Additionally, a well-prepared dataset with relevant features is crucial for effective predictions.

```

#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from plotnine import *

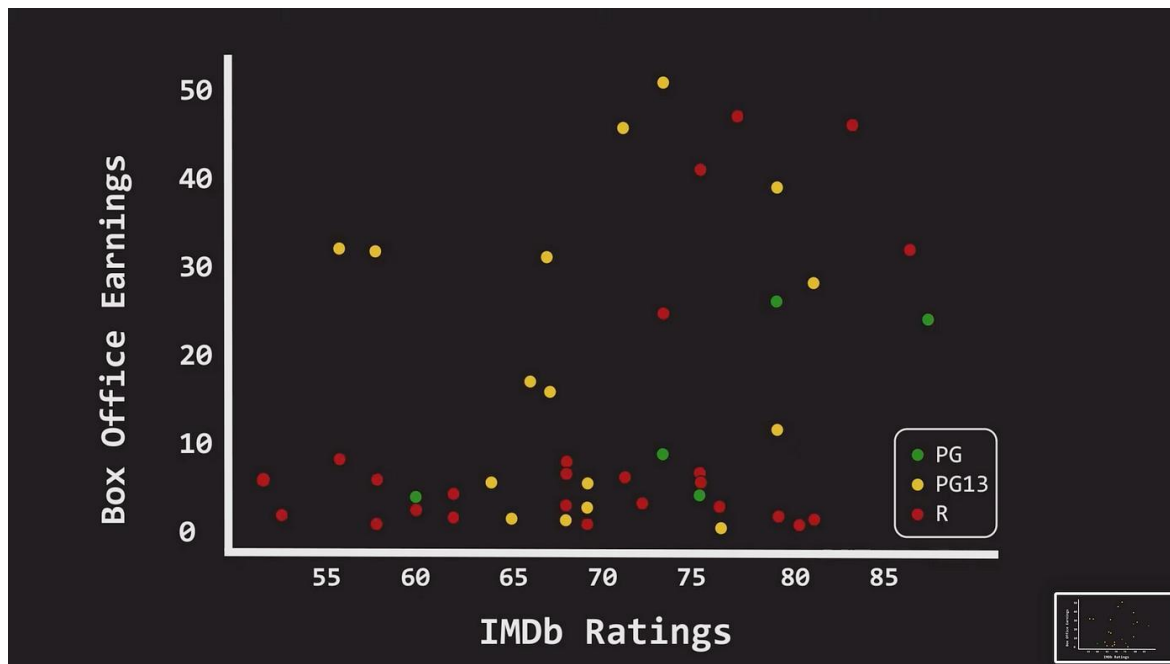
```

### **TMDb's API:**

After I performed extensive analysis, I thought budgets would be helpful for improving the accuracy of my predicted ratings, so I extracted them from TMDb's API. TMDb's API queries use TMDb's movie ID, which is different from IMDb's title ID, so

I couldn't query only the movies that I filtered previously (covered later in the Data Processing section), and like RapidAPI, it queries the data for each movie individually.

To work around this, I performed two queries, as shown below, that extracted the TMDb movie IDs that were filtered the same way as my other IMDb data. TMDb's "discover" API query provides up to 500 pages of results for a total of up to 10,000 results. Fortunately, after I filtered out movies (covered later in the Data Processing section), there were just over 10,000 movies, so I used parameters for the TMDb discover API calls such that the combination of the two would include those same movies. Filtering the API query by release year wasn't working, so I filtered by language (as U.S) and vote count ( $\geq 55$ ). Note the vote count is the number of TMDb votes, not the number of IMDb votes that I filtered. 55 was the minimum number, at the time of calling the API, that the two queries intersected data. I wanted to minimize the number of TMDb votes to minimize excluding movies. 1 of the queries was ascending (parameter `sort_by=vote_count.asc`) to capture the later pages, and the other query was descending to capture to earlier pages. I combined all these TMDb IDs in a list called IDs and removed 1 of each of the duplicates that were intersected by converting the list of IDs into a set



## Pandas: Display the movies that received specified number of votes

Import libraries:

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

Create empty arrays, we will use them in future for storing data of specific column.

```
TitleName=[]
Gross=[]
Weekend=[]
Week=[]
```

open the URL and extract the data from the website

```
url = "https://www.imdb.com/chart/boxoffice/?ref_=nv_ch_cht"
r = requests.get(url).content
```

Using the Find and Find All methods in BeautifulSoup. We extract the data and store it in the variable.

```
soup = BeautifulSoup(r, "html.parser")
list = soup.find("tbody", {"class":""}).find_all("tr")
x = 1
for i in list:
 title = i.find("td", {"class":"titleColumn"})
 gross = i.find("span", {"class":"secondaryInfo"})
 weekend = i.find("td", {"class":"ratingColumn"})
 week=i.find("td", {"class":"weeksColumn"})
```

Using append we store the details in the Array that we have created before

```
TitleName.append(title.text)
Gross.append(gross.text)
Weekend.append(weekend.text)
Week.append(week.text)
```

## 5. Store the data in a Sheet.We store the data in Comma-separated values (CSV format)

```
df=pd.DataFrame({'Movie Title':TitleName, 'Weekend':Weekend, 'Gross':Gross, 'Week':Week})
df.to_csv('DS-PR1-18IT012.csv', index=False, encoding='utf-8')
```

## 6. Now run the whole code.

All the data are stored as IMDbRating.csv in the path of the Python file.

DS-PR1-18IT012.csv X

1 to 10 of 10 entries Filter

Movie Title	Weekend	Gross	Week
Space Jam: A New Legacy	\$31.1M	\$31.1M	1
Black Widow	\$25.8M	\$131.6M	2
Escape Room: Tournament of Champions	\$8.8M	\$8.8M	1
F9	\$7.7M	\$154.9M	4
The Boss Baby: Family Business	\$4.7M	\$44.7M	3
The Forever Purge	\$4.1M	\$35.9M	3
A Quiet Place Part II	\$2.2M	\$155.0M	8
Roadrunner: A Film About Anthony Bourdain	\$2.0M	\$2.0M	1
Cruella	\$1.2M	\$83.5M	8
Pig	\$971K	\$971K	1

Show 10 per page

Last update on August 19 2022 21:51:42 (UTC/GMT +8 hours)

## Python Code :

```
import pandas as pd

df = pd.read_csv('movies_metadata.csv')

n = 500

small_df = df[['title', 'vote_count']]

result = small_df[small_df['vote_count'] >= n]

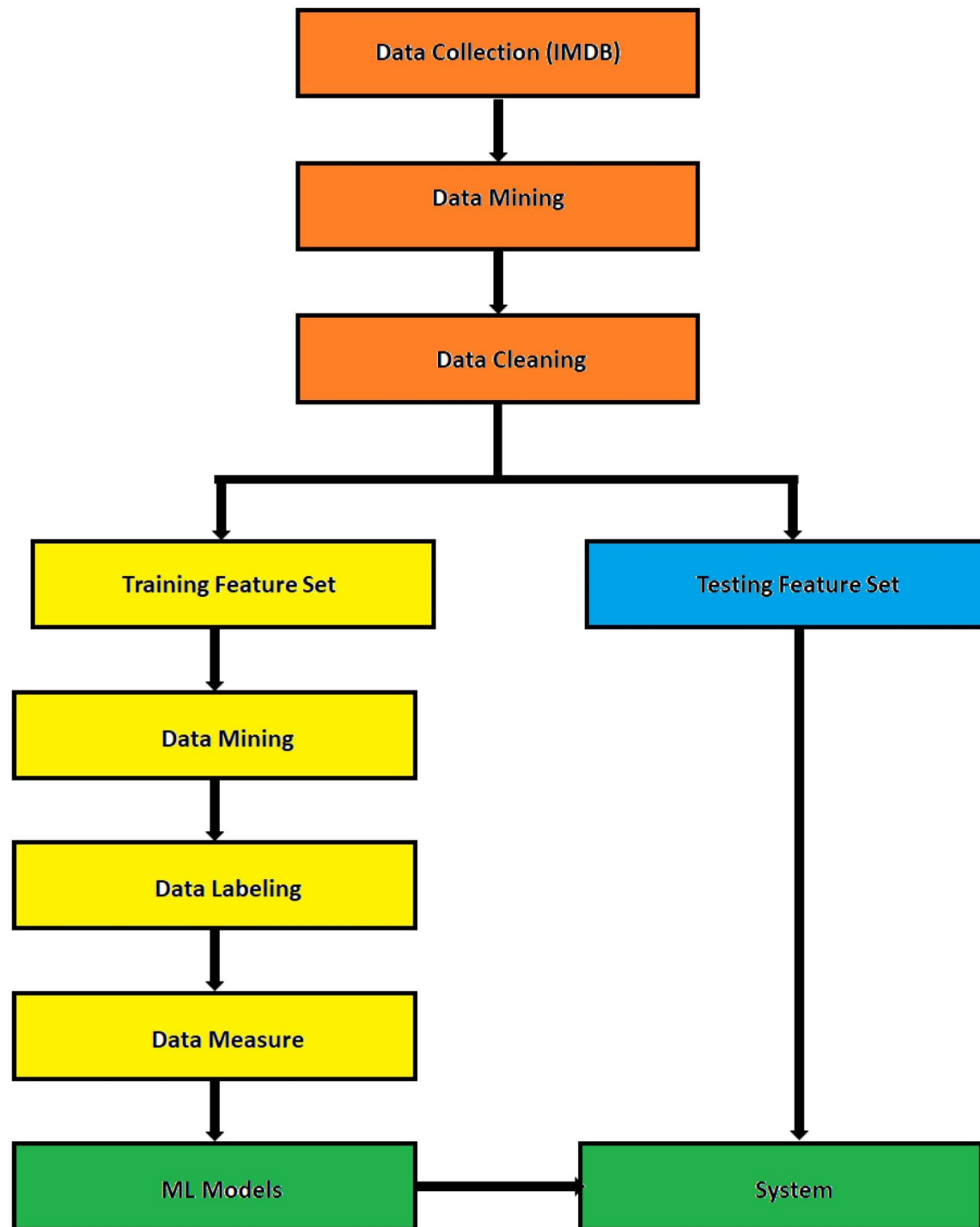
print("List of movies longer than 30 minutes and shorter than 360 minutes:")

print(result)
```

Copy

Sample Output:

List of movies longer than 30 minutes and shorter than 360 minutes:		
	title	vote_count
0	Toy Story	5415
1	Jumanji	2413
5	Heat	1886
9	GoldenEye	1194
15	Casino	1343
17	Four Rooms	539
18	Ace Ventura: When Nature Calls	1128
31	Twelve Monkeys	2470
33	Babe	756
38	Clueless	828
46	Se7en	5915
47	Pocahontas	1509
49	The Usual Suspects	3334



Load titles data

```
import pandas as pd
import matplotlib.pyplot as plt
titles = pd.read_table("/Users/pankaj/dev/data/imdb/title.basics.tsv")
ratings = pd.read_table("/Users/pankaj/dev/data/imdb/title.ratings.tsv")
/Users/pankaj/opt/anaconda3/envs/medium/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3441: DtypeWarning: Columns (4,5) have mixed types.Specify dtype option on import or set low_memory=False.
exec(code_obj, self.user_global_ns, self.user_ns)
```

Filter for movies. ignoring short or other title types

```
vals = titles[(titles['titleType']=='movie') & (titles['genres']!='\\N')]vals.head()
```

tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
570	tt0000574	movie	The Story of the Kelly Gang		1906	\\N	70	Action,Adventure,Biography
587	tt0000591	movie	The Prodigal Son		1907	\\N	90	Drama
610	tt0000615	movie	Robbery Under Arms		1907	\\N	\\N	Drama
625	tt0000630	movie	Hamlet		1908	\\N	\\N	Drama
668	tt0000675	movie	Don Quijote		1908	\\N	\\N	Drama

I share your feeling If you ever wondered why most of the western genre movies were released before 70's and why they are not so common nowadays. While analyzing the IMDB dataset I had similar question in mind about other genres. Was a particular genre more popular at a time than it was at others? what was the trend at what time. Using IMDB data I am trying to answer those question based on this data. Accuracy of result depends on the completeness of the data for which I rely on IMDB however we can catch a glimpse.

### Program :

import time

```
t = time.localtime(time.time())
```

```
localtime = time.asctime(t)
```

```
str = "Current Time:" + time.asctime(t)
```

```
print(str)
```

pages = np.arange(1, 9951, 50) # Last time I tried, I could only go to 10000 items because after that the URI has no discernable pattern to combat web crawlers; I just did 4 pages for demonstration purposes. You can increase this for your own projects.

```
headers = {'Accept-Language': 'en-US,en;q=0.8'} # If this is not specified, the default language is Mandarin
```

```
#initialize empty lists to store the variables scraped
```

```
titles = []
```

```
years = []
```

```
ratings = []
```

```
genres = []
```

```
runtimes = []
```

```
imdb_ratings = []
```

```
imdb_ratings_standardized = []
```

```
metascores = []
```

```
votes = []
```

```
for page in pages:
```

```

#get request for sci-fi
response = get("https://www.imdb.com/search/title?genres=sci-fi&"
 + "start="
 + str(page)
 + "&explore=title_type,genres&ref_=adv_prv", headers=headers)

sleep(randint(8,15))

#throw warning for status codes that are not 200
if response.status_code != 200:
 warn('Request: {}; Status code: {}'.format(requests, response.status_code))

#parse the content of current iteration of request
page_html = BeautifulSoup(response.text, 'html.parser')

movie_containers = page_html.find_all('div', class_ = 'lister-item mode-advanced')

#extract the 50 movies for that page
for container in movie_containers:

 #conditional for all with metascore
 if container.find('div', class_ = 'ratings-metascore') is not None:

 #title
 title = container.h3.a.text
 titles.append(title)

 if container.h3.find('span', class_ = 'lister-item-year text-muted unbold') is not None:

 #year released
 year = container.h3.find('span', class_ = 'lister-item-year text-muted unbold').text # remove the parentheses around
 the year and make it an integer
 years.append(year)

 else:

```



years.append(None) # each of the additional if clauses are to handle type None data, replacing it with an empty string so the arrays are of the same length at the end of the scraping

if container.p.find('span', class\_ = 'certificate') is not None:

#rating

rating = container.p.find('span', class\_ = 'certificate').text

ratings.append(rating)

else:

ratings.append("")

if container.p.find('span', class\_ = 'genre') is not None:

#genre

genre = container.p.find('span', class\_ = 'genre').text.replace("\n", "").rstrip().split(',') # remove the whitespace character, strip, and split to create an array of genres

genres.append(genre)

else:

genres.append("")

if container.p.find('span', class\_ = 'runtime') is not None:

#runtime

time = int(container.p.find('span', class\_ = 'runtime').text.replace(" min", "")) # remove the minute word from the runtime and make it an integer

runtimes.append(time)

else:

runtimes.append(None)

if float(container.strong.text) is not None:

#IMDB ratings

imdb = float(container.strong.text) # non-standardized variable

imdb\_ratings.append(imdb)

```
else:
 imdb_ratings.append(None)

if container.find('span', class_ = 'metascore').text is not None:

 #Metascore
 m_score = int(container.find('span', class_ = 'metascore').text) # make it an integer
 metascores.append(m_score)

else:
 metascores.append(None)

if container.find('span', attrs = {'name':'nv'})['data-value'] is not None:

 #Number of votes
 vote = int(container.find('span', attrs = {'name':'nv'})['data-value'])
 votes.append(vote)

else:
 votes.append(None)

else:
 votes.append(None)
```

#### ADVANTAGES

It can perform both regression and classification tasks. A random forest produces good predictions that can be understood easily. It can handle large datasets efficiently. The random forest algorithm provides a higher level of accuracy in predicting outcomes over the decision tree algorithm. Ideal in the following situations:

1. Predict the movie revenue.
2. Effective prediction technique
3. Beneficial for accurate prediction of which movies has to be released on priority basis.
4. Secure and efficient system

#### DISADVANTAGES

A country's GDP rate can be used as a feature to know if there is financial stability during the period when a movie is released. During an economic depression, very few amount of audience will go to the theatres to enjoy movies. So these facts play a vital role in an ultimate success of a movie. So this can't be predicted. When using a random forest, more resources are required for computation. It consumes more time compared to a decision tree algorithm.

#### LIMITATIONS

The number of audience plays a vital role for a movie to become successful. Because the whole point is about viewers, the entire industry will make no sense if there is no audience to watch a movie. The number of tickets sold during a specific year can indicate the number of viewers of that year. And the role of movie audience depends on many situations like political conditions and economic stability of a country. A country's GDP rate can be used as a feature to know if there is financial stability during the period when a movie is released. During an economic depression, very few amount of audience will go to the theatres to enjoy movies. Random forest does not produce good results when the data is very sparse. In this case, the subset of features and the bootstrapped sample will produce an invariant space. This will lead to unproductive splits, which will affect the outcome. So these facts play a vital role in an ultimate success of a movie. So in this situation our model doesn't predict the revenue

#### Conclusion

Predicting movie revenue can be done by using various data analysis techniques. Regression analysis is one of the techniques that uses numerical data to predict the related response. In this paper, we apply various regression techniques onto movie data, with and without grouping, in order to predict the movie revenue. We found that the linear regression without applying clustering technique is the most accurate method in terms of R-square. Moreover, dividing movie data by its genre before doing regression can improve revenue prediction accuracy in some cases. However, applying clustering techniques, such as EM and K-means, can improve revenue prediction accuracy in terms of RMSE. We achieved an accuracy score of approximately 81%. Let's compare this to the scores we got with previous regression models: Simple Linear Regression : 50% Multiple Linear Regression: 65% Decision Tree Regression: 65% Support Vector Regression: 71% Random Forest Regression: 81% We can see that our Random Forest Regression model made the most accurate predictions thus far with an improvement of 10% from the last model

---

