UNIVERSITY OF MALAYA

EXAMINATION FOR THE DEGREE OF MASTER OF DATA SCIENCE

ACADEMIC SESSION 2019/2020 : SEMESTER II

WQD7005 : Data Mining

June 2020

INSTRUCTIONS TO CANDIDATES :

Answer **ALL** questions (50 marks).

Name : Nurullainy Binti Mat Rashid

ID : 17036591

Link to codes and data:
https://github.com/Nurullainy/Data-Mining-Final-Exam

Link to video :
https://drive.google.com/drive/folders/1hMqLYw5ubSpMRsPvyJwjDG8MuV3GO_Qt?usp=sharing

(This question paper consists of 5 questions on 3 printed pages)

Mini-assignment (50 marks)

Instructions: Work individually, submission via Spectrum.

1. You are required to make a user-agent that will crawl the WWW (your familiar domain) to produce dataset of a particular website.
    - the web site can be as simple as a list of webpages and what other pages they link to
    - the output does not need to be in XHTML (or HTML) form
      a multi-stage approach (e.g. produce the xhtml or html in csv format)

    Topic: Web Crawling Data of TV Shows and Movies at Internet Movie Database (IMDb)

    IMDb is an online database of information related to films, television programs, home videos, video games, and streaming content online – including cast, year released, ratings, production crew, plot summaries, trivia, fan and critical reviews.

    I want to analyze data of TV Show and movie from IMDb. The data can be extracted from this website : https://www.imdb.com/search/title/?year=2017

    1.1 Importing Python Libraries

    First of all, I will do the following step:
    1) Import requests module and BeautifulSoup from bs4
    2) Assign the address of the web page to a variable named 'url'.
    3) Request the server the content of the web page by using get( ), and store the server's response in the variable 'response'.
    4) Print HTTP status code and a small part of response's content by accessing its text attribute (response is now a Response object).

```
In [1]: import requests
        from bs4 import BeautifulSoup as soup
        import re
```

```
In [2]: url = "https://www.imdb.com/search/title/?year=2017"

        response = requests.get(url, headers = {"Accept-Language": "en-US, en;q=0.5"})

        print('HTTP status:',response.status_code) # return response status from the server

        print(response.text[:300]) # Print a small part of response's content by accessing its .text attribute
```

```
HTTP status: 200


<!DOCTYPE html>
<html
    xmlns:og="http://ogp.me/ns#"
    xmlns:fb="http://www.facebook.com/2008/fbml">
    <head>

        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="apple-itunes-app" content="app-id=342792525, app-argument=imd
```

1.2 Using Beautifulsoup Module To Parse The HTML Content

Parsing HTML document and extract the 50 div containers

1) Import the BeautifulSoup class creator from the package bs4.
2) Parse response.text by creating a BeautifulSoup object, and assign this object to page_soup.
3) The 'html.parser' argument indicates that we want to do the parsing using Python's built-in HTML parser.

```
In [3]: page_soup = soup(response.text, "html.parser")  # .text or .content

        type(page_soup)

        # print(page_soup.prettify())

Out[3]: bs4.BeautifulSoup
```

Class attribute has two values; 1) lister-item and 2) mode-advanced.

This combination is unique to these div containers. Use the find_all() method to extract all the div containers that have a class attribute of *lister-item mode-advanced* and assign it to variable *movie_container*. It will return a ResultSet object which is a list containing all the 50 divs

```
In [4]: # To extract all the div containers that have a class attribute of lister-item mode-advanced

        movies_container = page_soup.find_all('div', class_='lister-item mode-advanced')

        # Number of movies in current web page
        len(movies_container)

Out[4]: 50
```

1.3 Extracting The Data For A Single Movie

Now I'm selecting one movie container (let say the first container) to extract 9 attributes that I am interested with for next data mining purposes:

```
In [7]: # Access the first container which contains information about a single movie
        # From a single movie, using this information to extract more data (date release, ratings, etc)

        first_movie = movies_container[0]  # 1st movie in the list

        print(first_movie.text.strip()) # Print a small part of response's content by accessing its .text attribute

        1.
        Dark
        (2017–2020)


        TV-MA
        |
        60 min
        |

        Crime, Drama, Mystery




        8.8




        Rate this

        1
        2
        3
        4
        5
        6
        7
        8
        9
        10

        8.8/10
        X




            A family saga with a supernatural twist, set in a German town, where the disappearance of two young children ex
        poses the relationships among four families.


            Stars:
        Karoline Eichhorn,
        Louis Hofmann,
        Jördis Triebel,
        Stephan Kampwirth


        Votes:
        192,616
```

The 9 following attributes are as follows:

1) The name of the TV show or movie
2) The year of release
3) Runtime of each TV show or movie
4) Genre of TV show or movie
5) Revenues from the movie released
6) The IMDB rating
7) The number of votes from user
8) Stars of the TV show or movie (name of director and main cast)
9) Hyperlink to the TV show or movie

1.4 Extracting Information For All The Tv Shows And Movies In A Single Page

1) Declare list of variables to have something to store the extracted data in.
2) Loop through each container in a web page (the variable which contains all the 50 movie containers).
3) Extract the data points of interest only if the container is True

In [209]:
```
# List to store the scraped data in

names = []
years = []
runtimes = []
genres = []
revenues = []
imdb_ratings = []
votes = []
stars = []
hyperlinks = []
```

In [210]:
```
for container in movies_container:
    if container is not None:

        name = container.h3.a.text
        names.append(name)

        year = container.h3.find('span', class_='lister-item-year text-muted unbold').text
        years.append(year)

        if container.find('span', class_='runtime'):
            runtime = container.find('span', class_='runtime').text
        else:
            runtime = ' '
        runtimes.append(runtime)

        genre = container.p.find('span', class_='genre').text
        genre = genre.replace('\n', '')
        genre = genre.rstrip()
        genres.append(genre)

        if container.findAll('span', attrs = {'name':'nv'})[1:]:
            revenue = container.findAll('span', attrs = {'name':'nv'})[1:]
            revenue = str(revenue)
            revenue = revenue.replace(' ', '').strip()
            revenue = revenue.replace('[<spandata-value="', '').strip()
            revenue = revenue.replace('"name="nv"', '').strip()
            revenue = revenue.replace('</span>]', '').strip()

        else:
            revenue = ''
        revenues.append(revenue)
```

```
        imdb_rating = container.strong.text
        imdb_rating = float(imdb_rating)
        imdb_ratings.append(imdb_rating)

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

        star = container.find('p', class_='').text
        star = str(star)
        star = star.replace('\n', '').strip()
        stars.append(star)

        link = container.h3.find('a')['href']
        link = "https://www.imdb.com/"+link
        hyperlinks.append(link)

    else:
        container = ''
```

Print last 10 movie_container in first web page

In [211]:
```
import pandas as pd

output_df = pd.DataFrame({'movie_name':names,
                          'year_released':years,
                          'runtime': runtimes,
                          'genre' : genres,
                          'revenues' : revenues,
                          'imdb_rating':imdb_ratings,
                          'vote':votes,
                          'Star':stars,
                          'link' : hyperlinks,
                         })

print(output_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 9 columns):
movie_name       50 non-null object
year_released    50 non-null object
runtime          50 non-null object
genre            50 non-null object
revenues         50 non-null object
imdb_rating      50 non-null float64
vote             50 non-null int64
Artist           50 non-null object
link             50 non-null object
dtypes: float64(1), int64(1), object(7)
memory usage: 3.6+ KB
None
```

In [212]:
```
# The last 10 movies in page 1

output_df.tail(10)
```

Out[212]:

	movie_name	year_released	runtime	genre	revenues	imdb_rating	vote	Artist	link
40	Taboo	(2017–)	59 min	Drama, Mystery, Thriller		8.4	110582	Stars:Tom Hardy, David Hayman, Jonathan Pryce,...	https://www.imdb.com//title/tt3647998/
41	Wonder Woman	(2017)	141 min	Action, Adventure, Fantasy	412,563,408>$412.56M	7.4	530070	Director:Patty Jenkins| Stars:Gal Gadot, C...	https://www.imdb.com//title/tt0451279/
42	Baby Driver	(2017)	113 min	Action, Crime, Drama	107,825,862>$107.83M	7.6	414324	Director:Edgar Wright| Stars:Ansel Elgort,...	https://www.imdb.com//title/tt3890160/
43	Pirates of the Caribbean: Dead Men Tell No Tales	(2017)	129 min	Action, Adventure, Fantasy	172,558,876>$172.56M	6.6	251090	Directors:Joachim Rønning, Espen Sandberg| ...	https://www.imdb.com//title/tt1790809/
44	Logan	(2017)	137 min	Action, Drama, Sci-Fi	226,277,068>$226.28M	8.1	619693	Director:James Mangold| Stars:Hugh Jackman...	https://www.imdb.com//title/tt3315342/
45	GLOW	(2017–)	35 min	Comedy, Drama, Sport		8.0	37573	Stars:Alison Brie, Marc Maron, Betty Gilpin, B...	https://www.imdb.com//title/tt5770786/
46	Jumanji: Welcome to the Jungle	(2017)	119 min	Action, Adventure, Comedy	404,515,480>$404.52M	6.9	303322	Director:Jake Kasdan| Stars:Dwayne Johnson...	https://www.imdb.com//title/tt2283362/
47	Young Sheldon	(2017–)	30 min	Comedy		7.4	30359	Stars:Iain Armitage, Zoe Perry, Lance Barber, ...	https://www.imdb.com//title/tt6226232/
48	King Arthur: Legend of the Sword	(2017)	126 min	Action, Adventure, Drama	39,175,066>$39.18M	6.7	184869	Director:Guy Ritchie| Stars:Charlie Hunnam...	https://www.imdb.com//title/tt1972591/
49	Imposters	(2017–)	41 min	Comedy, Crime, Drama		7.8	9287	Stars:Inbar Lavi, Rob Heaps, Parker Young, Mar...	https://www.imdb.com//title/tt5212822/

1.5 Extracting data for all TV shows and movies from multiple pages from year 2000 – 2020

1) Create a list called *pages*, and populate it with the strings corresponding to the first 4 pages.
2) Create a list called *years_url* and populate it with the strings corresponding to the years 2000 - 2020

```
In [214]: pages = [str(i) for i in range(1,5)]
          years_url = [str(i) for i in range(2000,2021)]
```

1.6 Controlling the crawl-rate

Controlling the rate of crawling is important for the website that I will be scraping. If I not controlling the rate of crawling, much less likely to get my IP address banned.

Need to avoid activity disruption of the scraped website by allowing the server to respond to other users' requests too.

Control the loop's rate by using the *sleep()* function from Python's time module. *sleep()* will pause the execution of the loop for a specified amount of seconds. To mimic human behavior, I'll vary the amount of waiting time between requests by using the *randint()* function from the Python's random module. *randint()* randomly generates integers within a specified interval.

```
In [215]: from time import sleep
          from random import randint

          sleep(randint(1,4))
```

Since the web scraping is more than 10 pages, it would be nice to find a way to monitor the scraping process as it's still going. The greater the number of pages, the more helpful the monitoring becomes.

For my script, I'll make use of this feature, and monitor the following parameters:

1) The *frequency (speed) of requests*, to ensure our program is not overloading the server
2) The *number of requests*, so I can halt the loop in case the number of expected requests is exceeded
3) The *status code of our requests*, to make sure the server is sending back the proper responses

To get a frequency value, I divide the number of requests by the time elapsed since the first request.

1) Set a starting time using the *time()* function from the time module, and assign the value to start_time.

2) Assign 0 to the variable requests which to be use to count the number of requests.
3) Start a loop, and then with each iteration:
   - Simulate a request
   - Increment the number of requests by 1
   - Pause the loop for a time interval between 8 and 15 seconds
   - Calculate the elapsed time since the first request, and assign the value to elapsed_time
   - Print the number of requests and the frequency

If I set my request to 100 requests, my output return will look a lengthy and a bit untidy as the output accumulates. To avoid that, I'll clear the output after each iteration, and replace it with information about the most recent request. I use the *clear_output()* function from the IPython's core.display module. Then set the wait parameter of clear_output() to True to wait with replacing the current output until some new output appears.

In [241]:
```
from time import time
from IPython.core.display import clear_output

start_time = time()
requests = 0

for _ in range(3):

    # A request would go here
    requests += 1
    sleep(randint(1,3))
    current_time = time()
    elapsed_time = current_time - start_time
    print('Request: {}; Frequency: {} requests/s'.format(requests, requests/elapsed_time))

    clear_output(wait = True)  # set wait = True, to wait with replacing the current output until some new output ap
```

```
Request: 3; Frequency: 0.7473912696391304 requests/s
```

In [242]:
```
# Redeclaring the lists variables so they become empty again.

names = []
years = []
runtimes = []
genres = []
revenues = []
imdb_ratings = []
votes = []
stars = []
hyperlinks = []
```

In [219]:
```python
from requests import get
from warnings import warn

headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chr


# Preparing the monitoring of the loop
start_time = time()
requests = 0

# For every year in the interval 2000-2020
for year_url in years_url:

    # For every page in the interval 1-4
    for page in pages:

        # Make a get request
        url = "http://www.imdb.com/search/title?release_date="
        response = get(url + year_url + '&sort=num_votes,desc&page=1' + page, headers = headers)

        # Pause the loop interval between 8 and 15 seconds
        sleep(randint(8,15))

        # Monitor the requests
        requests += 1
        elapsed_time = time() - start_time
        print('Request:{}; Frequency: {} requests/s'.format(requests, requests/elapsed_time))
        clear_output(wait = True)

        # Throw a warning for non-200 status codes
        if response.status_code != 200:
            warn('Request: {}; Status code: {}'.format(requests, response.status_code))

        # Break the loop if the number of requests is greater than expected
        if requests > 100:
            warn('Number of requests exceeds than expected.')
            break

        # Parse the content of the request with BeautifulSoup
        page_html = soup(response.text, 'html.parser')

        # Select all the 50 movie containers from a single page
        movies_container = page_html.find_all('div', class_ = 'lister-item mode-advanced')

        for container in movies_container:
            if container is not None:

                name = container.h3.a.text
                names.append(name)

                year = container.h3.find('span', class_='lister-item-year text-muted unbold').text
                years.append(year)

                if container.find('span', class_='runtime'):
                    runtime = container.find('span', class_='runtime').text
                else:
                    runtime = ' '
                runtimes.append(runtime)

                genre = container.p.find('span', class_='genre').text
                genre = genre.replace('\n', '')
                genre = genre.rstrip()
                genres.append(genre)
```

```
            if container.findAll('span', attrs = {'name':'nv'})[1:]:
                revenue = container.findAll('span', attrs = {'name':'nv'})[1:]
                revenue = str(revenue)
                revenue = revenue.replace(' ', '').strip()
                revenue = revenue.replace('[<spandata-value="', '').strip()
                revenue = revenue.replace('"name="nv"', '').strip()
                revenue = revenue.replace('</span>]', '').strip()

            else:
                revenue = ''
            revenues.append(revenue)

            imdb_rating = container.strong.text
            imdb_rating = float(imdb_rating)
            imdb_ratings.append(imdb_rating)

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

            star = container.find('p', class_='').text
            star = str(star)
            star = star.replace('\n', '').strip()
            stars.append(star)

            link = container.h3.find('a')['href']
            link = "https://www.imdb.com/"+link
            hyperlinks.append(link)

        else:
            container = ''
```

```
Request:84; Frequency: 0.06464969451330399 requests/s
```

I set the loop limit to 100 but the requests stop at request number 84. This indicates that I have collected all TV shows and movies data from year 2000 – 2020.

Print 10 movie_container in first web page

In [236]:
```
output_df = pd.DataFrame({'movie_name':names,
                          'year_released':years,
                          'runtime_in_min': runtimes,
                          'genre' : genres,
                          'revenues' : revenues,
                          'imdb_rating':imdb_ratings,
                          'number_of_votes':votes,
                          'Artist':stars,
                          'link' : hyperlinks,
                         })

print(output_df.info())
```

In [237]:
```
output_df.head()
```

Out[237]:

	movie_name	year_released	runtime_in_min	genre	revenues	imdb_rating	number_of_votes	Artist	
0	Gladiator	(2000)	155 min	Action, Adventure, Drama	187,705,427>$187.71M	8.5	1295296	Director:Ridley Scott| Stars:Russell Crowe...	https://www.imdb.com//title/t
1	Memento	(2000)	113 min	Mystery, Thriller	25,544,867>$25.54M	8.4	1088494	Director:Christopher Nolan| Stars:Guy Pear...	https://www.imdb.com//title/t
2	Snatch	(2000)	104 min	Comedy, Crime	30,328,156>$30.33M	8.3	760513	Director:Guy Ritchie| Stars:Jason Statham,...	https://www.imdb.com//title/t
3	Requiem for a Dream	(2000)	102 min	Drama	3,635,482>$3.64M	8.3	742056	Director:Darren Aronofsky| Stars:Ellen Bur...	https://www.imdb.com//title/t
4	X-Men	(2000)	104 min	Action, Adventure, Sci-Fi	157,299,717>$157.30M	7.4	558637	Director:Bryan Singer| Stars:Patrick Stewa...	https://www.imdb.com//title/t

Total number of TV shows and movies collected are 4200 titles.

```
In [51]: output_df.shape
Out[51]: (4200, 9)
```

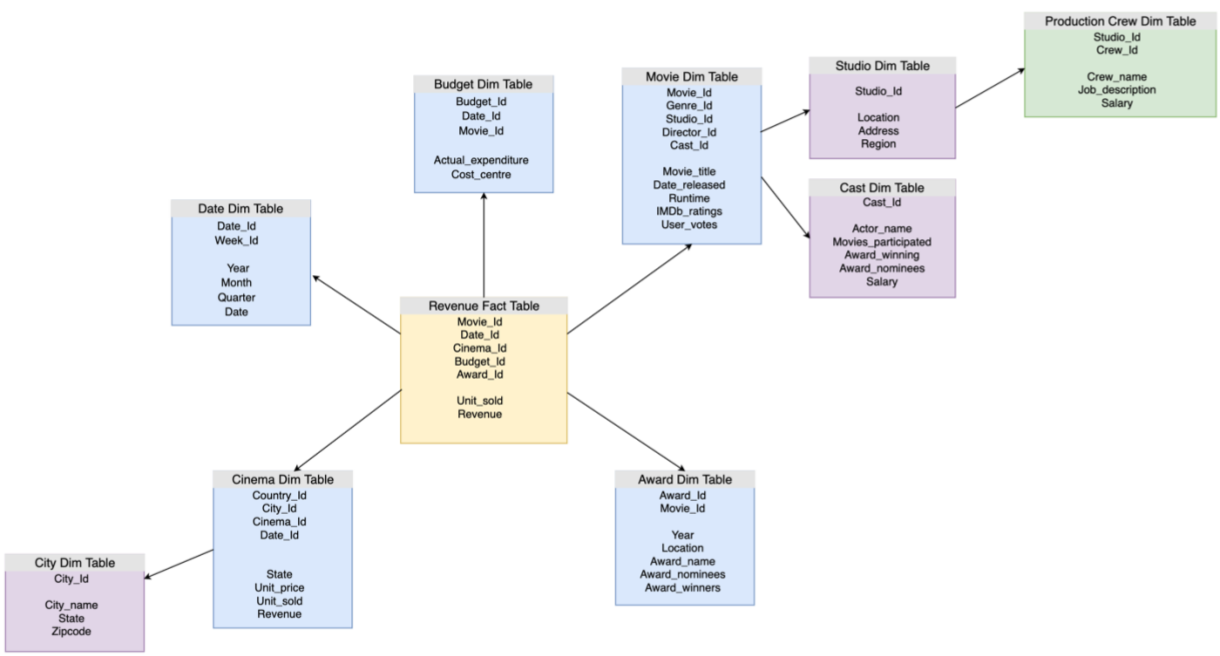
Export dataframe to csv file

Export dataframe to csv file

```
In [239]: import csv
          output_df.to_csv('movies_imdb_with revenue.csv',mode = 'w', index=True, header=True)
```

(10 marks)

2. Draw snowflake schema diagram for the above dataset. Justify your attributes to be selected in the respective dimensions.



Snowflakes Diagram data model for Movie Data Warehouse

Fact Table:

1) Revenue Fact Table
   a. Primary Key: Movie_Id, Date_Id, Cinema_Id, Budget_Id, Award_Id
   b. Foreign Key: Unit_sold and Revenue

Dimension Table:

1) Movie Dimension Table
    a) Primary Key : Movie_Id, Genre_Id, Studio_Id, Director_Id, Cast_Id
    b) Attributes : Movie_title, Date_released, Runtime, IMDb_ratings, User_votes

2) Cast Dimension Table
    a) Primary Key : Cast_Id
    b) Attributes : Actor_name, Movies_participated, Award_winning, Award_nominees, Salary

3) Studio Dimension Table
    a) Primary Key : Studio_Id
    b) Attributes : Location, Address, Region

4) Production Crew Dimension Table
    a) Composite Key : Studio_Id, Crew_Id
    b) Attributes : Crew_name, Job_description, Salary

5) Award Dimension Table
    a) Composite Key : Award_Id, Movie_Id
    b) Attributes : Year, Location, Award_name, Award_nominees, Award_winners

6) Cinema Dimension Table
    a) Composite Key: Cinema_Id, City_Id
    b) Primary Key: Country_Id, Date_Id
    c) Foreign Key: State, Unit_price, Unit_sold, Revenue

7) City Dimension Table
    a) Primary Key : City_Id
    b) Foreign Key : State
    c) Attributes : City_name, Zipcode

8) Budget Dimension Table
    a) Primary Key : Budget_Id, Date_Id, Movie_Id
    b) Attributes : Actual_expenditure, Cost_centre

9) Date Dimension Table
    a) Primary Key : Date_Id, Week_Id
    b) Attributes : Year, Month, Quarter

Description:

a) In snowflakes schema, **Primary Key** is that column of the table whose every row data is uniquely identified. Every row in the table must have a primary key and no two rows can have the same primary key. Primary key value can never be null nor can be modified or updated. The **Composite Key** is a form of the candidate key where a set of columns will uniquely identify every row in the table. For example, Cinema_Id and City_Id in Cinema dimension table. **Foreign Key** are the columns of a table that points to the key of another table. They act as a cross-reference between tables.

b) OLAP operations consist of 5 types of operation which are
   - Drill down - converting less detailed data into highly detailed data
   - Roll Up - climbing up in the concept hierarchy
   - Dice - select a sub-cube from the OLAP by selecting two or more dimensions
   - Slice - select a sub-cube from the OLAP which results in a new sub-cube creation.
   - Pivot - rotates the current view to get a new view of the representation

c) For my data warehouse, I am using typical OLAP operation which are **Roll Up** and **Drill Down** based on 1 fact table (Revenue) and 9 dimension tables (Movie, Studio, Cast, Production Crew, Award, Cinema, City, Budget and Date)

d) **Drill Down** is to examine my summary data which break out by dimension attributes. From higher level to lower level summary; introducing new dimension. Based on my Movie_Id in Revenue fact table to Movie, Budget and Award dimension table. Another example is Studio_Id in Movie dimension table to Studio and Production Crew dimension table

e) **Roll Up** is operation by climbing up hierarchy or by dimension reduction. For instance, my attribute of City_Id in City dimension table to Cinema dimension table and Studio_Id in Production Crew dimension table to Movie dimension table.

(10 marks)

3. You are required to write code to create a decision tree (DT) model using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:

- Importing required libraries

```
In [2]: import pandas as pd
        import numpy as np

        import seaborn as sns
        import matplotlib.pyplot as plt
        import plotly.express as px

        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import classification_report, accuracy_score
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeRegressor

        from sklearn.tree import export_graphviz
        from sklearn.model_selection import GridSearchCV
        import pydot
        from io import StringIO
        from sklearn.tree import export_graphviz
```

- Loading Data

```
In [3]: # Load dataset

        df = pd.read_csv('movies_imdb_preprocessed.csv')
        df.head()
```

Out[3]:

	movie_name	year_released	runtime_in_min	genre	revenues	imdb_rating	user_votes	director	actor
0	Gladiator	2000	155	Action, Adventure, Drama	187705427.0	8.5	1295546	Ridley Scott	Russell Crowe, Joaquin Phoenix, Connie Nielsen...
1	Memento	2000	113	Mystery, Thriller	25544867.0	8.4	1088700	Christopher Nolan	Guy Pearce, Carrie-Anne Moss, Joe Pantoliano, ...
2	Snatch	2000	104	Comedy, Crime	30328156.0	8.3	760646	Guy Ritchie	Jason Statham, Brad Pitt, Benicio Del Toro, De...
3	Requiem for a Dream	2000	102	Drama	3635482.0	8.3	742193	Darren Aronofsky	Ellen Burstyn, Jared Leto, Jennifer Connelly, ...
4	X-Men	2000	104	Action, Adventure, Sci-Fi	157299717.0	7.4	558716	Bryan Singer	Patrick Stewart, Hugh Jackman, Ian McKellen, F...

```
In [4]: # Remove NAs and unused attribute

        df.dropna(how='any', inplace=True)
        df = df.drop(['movie_name'], axis=1)
```

```
In [7]: # Preprocessing revenues column

        df_revenue = df['revenues'].div(1000000).to_frame('col') # Change to Million notation
        df_revenue.shape

        df['revenues'] = df_revenue['col']

        df['revenues'] = df['revenues'].round(0).astype(int)
        #df.columns = ['revenues in mil']  # Rename the columns name
```

```
In [8]: # Change data to categorical variables

        df['year_released'] = df['year_released'].astype('str')
        df['genre'] = df['genre'].astype('str')
        df['director'] = df['director'].astype('str')
        df['actor'] = df['actor'].astype('str')
```

```
In [9]: # one hot encoding all categorical variables
        # all numerical variables are automatically excluded
        # number of columns after the conversion should explode
        print("Before:", len(df.columns))

        # one hot encoding
        df = pd.get_dummies(df)

        print("After:", len(df.columns))

        Before: 8
        After: 1351
```

- Feature Selection

```
In [11]: # Find average revenue number to estimate a threshold

         df['revenues'].mean()

Out[11]: 122.46446078431373
```

Based on average revenue, I set USD 100 million as threshold to binarize the target variable

```
In [12]: # Change revenues column to binary variables

         threshold, upper, lower = 100, 1, 0

         df['revenues'] = np.where(df['revenues']>threshold, upper, lower)

In [13]: df['revenues'].unique()

Out[13]: array([1, 0])

In [31]: df.head()

Out[31]:
```

	runtime_in_min	revenues	imdb_rating	user_votes	year_released_2000	year_released_2001	year_released_2002	year_released_2003	year_released_2004	ye
0	155	1	8.5	1295546	1	0	0	0	0	
1	113	0	8.4	1088700	1	0	0	0	0	
2	104	0	8.3	760646	1	0	0	0	0	
3	102	0	8.3	742193	1	0	0	0	0	
4	104	1	7.4	558716	1	0	0	0	0	

5 rows × 1351 columns

```
In [14]: # Assigning X and y variables. y variable is revenues in mil while the rest of the variables are X variables

         X = df.drop(['revenues'], axis=1)
         y = df['revenues']
```

- Splitting Data

```
In [15]: # Setting random state = 0
         rs = 0

         # Training set = 70%
         # Test Set = 30%
         # Stratify = Yes
         X_mat = np.asmatrix(X)
         X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size=0.3, stratify=y, random_state=rs)
```

As train_test_split shuffles the dataset before splitting it, it is important to set a consistent random state, which is the seed number used to generate the shuffle. I am using 0 for random state number

Convert X (DataFrame object) into a numpy matrix that can be consumed by sklearn. Next, use the train_test_split function to split dataset into 70% training and 30% test data. This is to ensure there is enough representation of the minority class in the training set. In this case, I need larger training set, which is 70/30.

Stratification method ensures the same ratio of positive and negative targets in both train and test data set.

- Building Decision Tree Model

Initialise a model and training it using .fit function.

```
In [16]: # Decision tree training

         model = DecisionTreeClassifier()
         model.fit(X_train, y_train)

Out[16]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False,
                                random_state=None, splitter='best')
```

- Evaluating Model

```
In [17]: print("Train accuracy:", model.score(X_train, y_train))

         Train accuracy: 1.0
```

It seems that the model has managed to learn all of the patterns in training data and is able to predict with 100% accuracy. However, need to check whether it can replicate the performance on similar data that it is not trained on (test data).

```
In [18]: print("Test accuracy:", model.score(X_test, y_test))

         Test accuracy: 0.7224489795918367
```

This is a clear indication of overfitting of the model. This model will fail to make accurate predictions with new data because it learned the training data too well. Need to make the model generalize better on training dataset.

I'm using classification_report() function to assess the model's prediction on test data. classification_report() outputs a number of statistics for each target class:

1. Precision: Proportion of all positive predictions that are correct. Precision is a measure of how many positive predictions were actual positive observations.

2. Recall: Proportion of all real positive observations that are correct. Precision is a measure of how many actual positive observations were predicted correctly.
3. F1: The harmonic mean of precision and recall. F1 score is an 'average' of both precision and recall.
4. Support: Number of instances in each class.

```
In [19]: y_pred = model.predict(X_test)

         print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.75      0.74       132
           1       0.70      0.69      0.70       113

    accuracy                           0.72       245
   macro avg       0.72      0.72      0.72       245
weighted avg       0.72      0.72      0.72       245
```

To understand the Decision Tree model built, lets view the feature importance and visualize the tree using sklearn module.

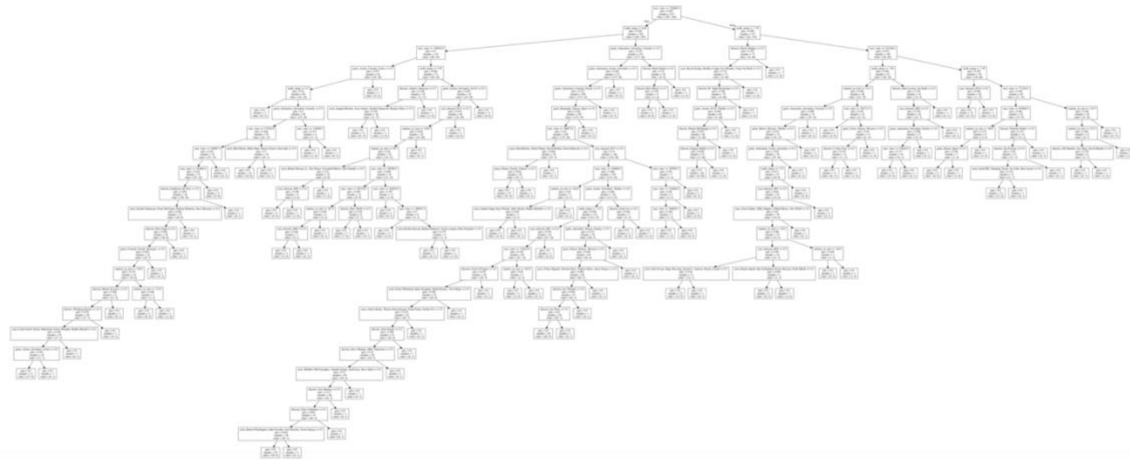
- Visualizing Decision Trees

```
In [20]: # grab feature importances from the model and feature name from the original X
         importances = model.feature_importances_
         feature_names = X.columns

         # sort them out in descending order
         indices = np.argsort(importances)
         indices = np.flip(indices, axis=0)

         # Print 10 most important features
         indices = indices[:10]

         for i in indices:
             print(feature_names[i], ':', importances[i])
```

```
user_votes : 0.33192163958645887
imdb_rating : 0.14176438634151728
runtime_in_min : 0.08031531119929572
genre_Animation, Adventure, Comedy : 0.03344948847789565
year_released_2013 : 0.017474449105946074
genre_Animation, Action, Adventure : 0.01494659694261953
genre_Action, Adventure, Sci-Fi : 0.013854789878016046
genre_Animation, Comedy, Family : 0.01016894017695399
genre_Biography, Drama, Sport : 0.009785232690288675
year_released_2001 : 0.008648437889323644
```

We can't really gain insights of the decision tree with feature importance only. Need to perform feature importance and visualization to understand the decision tree model. Use export_graphviz function and pydot module and save .png file to view the decision tree.

```
In [21]: # Visualize the 1st Decision Tree

         dotfile = StringIO()
         export_graphviz(model, out_file=dotfile, feature_names=X.columns)

         graph = pydot.graph_from_dot_data(dotfile.getvalue())
         graph[0].write_png("dt1.png")
```



The above tree shows that the model is very complex, incomprehensible, and deep, which is a typical characteristic of an overfitting model. I want to limit the complexity of the model by setting the max_depth that the model can go. max_depth in a decision tree is a hyperparameter for structuring the depth of the tree (model).

```
In [22]: # Retrain model with a smaller max_depth limit = 9

         model = DecisionTreeClassifier(max_depth=9, random_state=rs)
         model.fit(X_train, y_train)

         print("Train accuracy:", model.score(X_train, y_train),"\nTest accuracy:", model.score(X_test, y_test))
         print()

         y_pred = model.predict(X_test)
         print(classification_report(y_test, y_pred))
```

```
Train accuracy: 0.9124343257443083
Test accuracy: 0.7020408163265306

              precision    recall  f1-score   support

           0       0.70      0.78      0.74       132
           1       0.70      0.61      0.65       113

    accuracy                           0.70       245
   macro avg       0.70      0.70      0.70       245
weighted avg       0.70      0.70      0.70       245
```

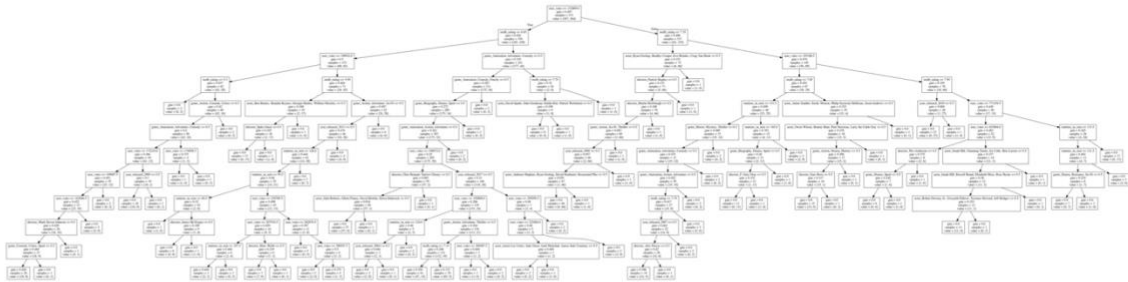
The simpler model (smaller max_depth) resulting the accuracy of the model on training data reduce to 91.2%. This means that model notes there is a trend in the data but not learning the training data too well.

View new feature importance and visualize this new decision tree.

```
In [41]: importances = model.feature_importances_
         feature_names = X.columns

         # sort them out in descending order
         indices = np.argsort(importances)
         indices = np.flip(indices, axis=0)

         # Print 10 variable X
         indices = indices[:10]

         for i in indices:
             print(feature_names[i], ':', importances[i])
```

```
user_votes : 0.3768873337944813
imdb_rating : 0.16509594141580766
runtime_in_min : 0.07063236145916083
genre_Animation, Adventure, Comedy : 0.04790613932512415
genre_Animation, Action, Adventure : 0.020414692785475655
genre_Biography, Drama, Sport : 0.018912354024285566
genre_Action, Adventure, Sci-Fi : 0.016971881080745224
year_released_2000 : 0.01642744187308721
year_released_2013 : 0.014008536765059568
genre_Animation, Comedy, Family : 0.01198674299857236
```

```
In [42]: # Visualize the new tree

         dotfile = StringIO()
         export_graphviz(model, out_file=dotfile, feature_names=X.columns)
         graph = pydot.graph_from_dot_data(dotfile.getvalue())
         graph[0].write_png("dt2.png") # saved in the following file
```



This looks better from the first model. However, the tree has more 20 leaf nodes here. Furthermore, there are a number of samples and value splits in each node.

Next is to find the optimal combination of parameters for the model.

Finding optimal parameters with GridSearchCV

A common method to find the optimal set of parameters for a model is to run a exhaustive search over all possible values of each parameter. To choose the best model among all candidates, k-fold cross validation is typically used.

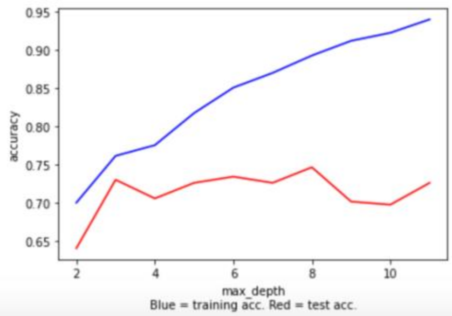
Grid-search builds a model for every combination of hyperparameters specified and evaluates each model.

In sklearn, the grid-search and k-fold validation is implemented in *GridSearchCV*.

Begin with plotting max_depth values vs training and test accuracy score to a give an idea of the optimal max_depth.

```
In [43]: test_score = []
         train_score = []

         # check the model performance for max depth from 2-12
         for max_depth in range(2, 12):
             model = DecisionTreeClassifier(max_depth=max_depth, random_state=rs)
             model.fit(X_train, y_train)

             test_score.append(model.score(X_test, y_test))
             train_score.append(model.score(X_train, y_train))
```

```
In [44]: # plot max depth hyperparameter values vs training and test accuracy score

         plt.plot(range(2, 12), train_score, 'b', range(2,12), test_score, 'r')
         plt.xlabel('max_depth\nBlue = training acc. Red = test acc.')
         plt.ylabel('accuracy')
         plt.show()
```



To perform a GridSearchCV, we first have to determine the hyperparameters and possible values of parameters that we want to use. A model hyperparameter is a characteristic of a model that is external to the model and whose value cannot be estimated from data.

For decision tree model, I will search on 3 hyperparameters:

- Criterion: The function to measure the quality of a split. There are two criterias we will use, “gini” for the Gini impurity and “entropy” for the information gain.
- Max depth: The maximum depth of the tree. Let's start with range of 2-11.
- Min samples leaf: The minimum number of samples required to be at a leaf node, allowing us to limit the minimum size of a leaf node. Let's start with range of 20-60 with step of 10.

In [27]:
```
# GridsearchCV #1

params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(1, 10),
          'min_samples_leaf': range(20, 60, 10)}

cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=10)
cv.fit(X_train, y_train)

print("Train accuracy:", cv.score(X_train, y_train))
print("Test accuracy:", cv.score(X_test, y_test))

# test the best model
y_pred = cv.predict(X_test)
print(classification_report(y_test, y_pred))

# print parameters of the best model
print(cv.best_params_)
```

```
Train accuracy: 0.7495621716287215
Test accuracy: 0.7183673469387755
              precision    recall  f1-score   support

           0       0.71      0.81      0.76       132
           1       0.73      0.61      0.67       113

    accuracy                           0.72       245
   macro avg       0.72      0.71      0.71       245
weighted avg       0.72      0.72      0.71       245

{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 40}
```

The accuracy score for the 3rd model is 72% which improved from the 2nd model, 70%. At the same time, the Precision is increased by 2% from the previous model. I want to fine tune and further optimise on the parameters if possible.

At this moment, the metric Recall (sensitivity) is acceptable at 81% and 61%.

Let's do another grid search, now being more specific based on the 1st grid search result.

In [28]:
```
# GridsearchCV #2

params = {'criterion': ['gini'],
          'max_depth': range(1, 6),
          'min_samples_leaf': range(40, 60, 5)}

cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=10)
cv.fit(X_train, y_train)

print("Train accuracy:", cv.score(X_train, y_train))
print("Test accuracy:", cv.score(X_test, y_test))

# test the best model
y_pred = cv.predict(X_test)
print(classification_report(y_test, y_pred))

# print parameters of the best model
print(cv.best_params_)
```

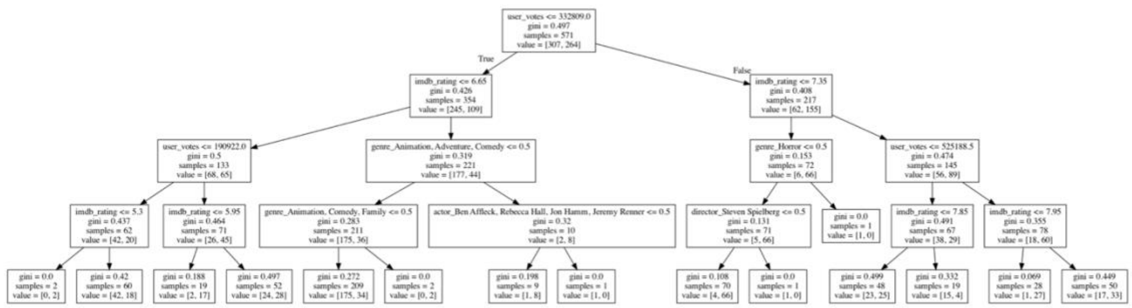
```
Train accuracy: 0.7495621716287215
Test accuracy: 0.7183673469387755
              precision    recall  f1-score   support

           0       0.71      0.81      0.76       132
           1       0.73      0.61      0.67       113

    accuracy                           0.72       245
   macro avg       0.72      0.71      0.71       245
weighted avg       0.72      0.72      0.71       245

{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 45}
```

The classification report of 4th model return not much different from the 3rd model. Both models have same weighted average score for Accuracy, Precision and Recall. In this case, I will retrain a new model based simpler hyperparameter, *being only 4 levels deep.*

In [69]:
```
# Retrain final model with optimal parameter, max_depth limit = 4

model = DecisionTreeClassifier(max_depth=4, random_state=rs)
model.fit(X_train, y_train)

print("Train accuracy:", model.score(X_train, y_train),"\nTest accuracy:", model.score(X_test, y_test))
print()
```

```
Train accuracy: 0.7758318739054291
Test accuracy: 0.7061224489795919
```

In [70]:
```
# Visualize the optimal tree

dotfile = StringIO()
export_graphviz(model, out_file=dotfile, feature_names=X.columns)

graph = pydot.graph_from_dot_data(dotfile.getvalue())
graph[0].write_png("optimal_tree.png") # saved in the following file
```



(10 marks)

4. You are required to write code to find frequent itemsets using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:

- Importing required libraries

In [182]:
```
import pandas as pd
import numpy as np

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules

import csv
```

In [183]:
```
# Load dataset

df = pd.read_csv('movies_imdb_preprocessed.csv')
df.head()
```

Out[183]:

	movie_name	year_released	runtime_in_min	genre	revenues	imdb_rating	user_votes	director	actor
0	Gladiator	2000	155	Action, Adventure, Drama	187705427	8.5	1295546	Ridley Scott	Russell Crowe, Joaquin Phoenix, Connie Nielsen...
1	Memento	2000	113	Mystery, Thriller	25544867	8.4	1088700	Christopher Nolan	Guy Pearce, Carrie-Anne Moss, Joe Pantoliano, ...
2	Snatch	2000	104	Comedy, Crime	30328156	8.3	760646	Guy Ritchie	Jason Statham, Brad Pitt, Benicio Del Toro, De...
3	Requiem for a Dream	2000	102	Drama	3635482	8.3	742193	Darren Aronofsky	Ellen Burstyn, Jared Leto, Jennifer Connelly, ...
4	X-Men	2000	104	Action, Adventure, Sci-Fi	157299717	7.4	558716	Bryan Singer	Patrick Stewart, Hugh Jackman, Ian McKellen, F...

- Creating a list from dataset (Question 1)

In [9]:
```
# Create new subset of dataset

revenue_actor = df[['revenues', 'genre']]

revenue_actor.head(10)
```

Out[9]:

	revenues	genre
0	187705427	Action, Adventure, Drama
1	25544867	Mystery, Thriller
2	30328156	Comedy, Crime
3	3635482	Drama
4	157299717	Action, Adventure, Sci-Fi
5	233632142	Adventure, Drama, Romance
6	15070285	Comedy, Crime, Drama
7	95011339	Drama, Mystery, Sci-Fi
8	215409889	Action, Adventure, Thriller
9	166244045	Comedy, Romance

Consolidate the items into each revenue number, in this case revenues in million

In [50]:
```
# Group actor by revenue generated (298 unique revenue number)

basket = revenue_actor.groupby(['revenues in mil'])['genre'].apply(list)

print("\n", basket[:6])
```

```
 revenues in mil
0    [Action, Crime, Thriller, Crime, Drama, Myster...
1    [Drama, Mystery, Sci-Fi, Action, Drama, Sci-Fi...
2    [Crime, Drama, Crime, Drama, Comedy, Romance, ...
3                                    [Drama, Romance]
4    [Drama, Crime, Drama, Musical, Comedy, Drama, ...
5    [Drama, Thriller, Animation, Adventure, Family...
Name: genre, dtype: object
```

```
In [51]: # List all the actor in list format (for model preparation)

basket_list = list(basket)

print("\n", basket_list[:10])
```

```
 [['Action, Crime, Thriller', 'Crime, Drama, Mystery', 'Action, Crime, Drama', 'Crime, Drama, Sport', 'Adventure, C
omedy, Sci-Fi', 'Drama', 'Drama, Fantasy, Romance', 'Comedy, Horror', 'Action, Adventure, Drama'], ['Drama, Mystery
, Sci-Fi', 'Action, Drama, Sci-Fi', 'Action, Drama, Mystery', 'Drama, Thriller', 'Drama, Thriller', 'Drama'], ['Cri
me, Drama', 'Crime, Drama', 'Comedy, Romance, Sport', 'Drama, Horror, Romance', 'Biography, Crime, Drama'], ['Drama
, Romance'], ['Drama', 'Crime, Drama, Musical', 'Comedy, Drama, Romance', 'Action, Comedy, Crime', 'Sci-Fi, Thrille
r'], ['Drama, Thriller', 'Animation, Adventure, Family', 'Drama, Mystery, Sci-Fi', 'Action, Drama, Sci-Fi', 'Animat
ion, Drama, Fantasy'], ['Comedy, Drama', 'Biography, Drama, History', 'Drama, Romance', 'Action, Crime, Thriller',
'Drama, Mystery, Romance', 'Action, Adventure, Comedy', 'Comedy, Drama'], ['Drama, Mystery, Thriller', 'Comedy, Dra
ma', 'Drama'], ['Crime, Drama', 'Comedy, Crime, Drama'], ['Comedy, Drama, Romance']]
```

- Convert list to dataframe with Boolean values

```
In [52]: # Convert list to dataframe with Boolean values

te = TransactionEncoder()
te_ary = te.fit(basket_list).transform(basket_list)

df2 = pd.DataFrame(te_ary, columns=te.columns_)
df2.head(10)
```

Out[52]:

	Action, Adventure	Action, Adventure, Biography	Action, Adventure, Comedy	Action, Adventure, Crime	Action, Adventure, Drama	Action, Adventure, Family	Action, Adventure, Fantasy	Action, Adventure, History	Action, Adventure, Horror	Action, Adventure, Mystery	...	Horror	Horror, Mystery	Horror, Mystery, Thriller	Hor Sc
0	False	False	False	False	True	False	False	False	False	False	...	False	False	False	Fa
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
5	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
6	False	False	True	False	False	False	False	False	False	False	...	False	False	False	Fa
7	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
8	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa
9	False	False	False	False	False	False	False	False	False	False	...	False	False	False	Fa

10 rows × 157 columns

The above table shows the distribution of each movie genre(s) in one revenue number. *False* indicates no genre(s) by the specific revenue number whereas *True* indicates that the movie genre(s) falls under that specific revenue number.

- Find frequently occurring itemsets using Apriori Algorithm

  1) Pros: Easy to code up
  2) Cons: May be slow on large datasets
  3) Works with: Numeric values, nominal values

General approach to the Apriori algorithm:

1) Preparation: Any data type will work because we storing sets.
2) Train: Use the Apriori algorithm to find frequent itemsets.

3) Test: Doesn't apply.
4) Application: This will be used to find frequent itemsets and association rules between items.

`Apriori` is an algorithm for frequent itemset mining and Association Rule learning over relational databases. The algorithm identify the frequent individual items in the database and extending them to larger and larger itemsets as long as those itemsets appear sufficiently often in the database.

The frequent itemsets determined by `Apriori` can be used to generate `Association Rules` which highlight general trends in the database. This application is widely used in market basket analysis.

The Support and Confidence are measures to measure how interesting a rule is. These parameters are used to exclude rules in the result that have a Support or a Confidence lower than the minimum support and minimum confidence respectively. *I have experimented a number trial of minimum support number and 0.01 is the best for this dataset.*

In [59]:
```
# Frequently occurring itemsets using Apriori Algorithm

frequent_itemsets_apriori = apriori(df2, min_support=0.01,
                                    use_colnames=True).sort_values(by='support', ascending=0)
frequent_itemsets_apriori.head(10)
```

Out[59]:

	support	itemsets
7	0.204698	(Action, Adventure, Sci-Fi)
31	0.117450	(Animation, Adventure, Comedy)
5	0.097315	(Action, Adventure, Fantasy)
39	0.083893	(Comedy)
45	0.080537	(Comedy, Drama, Romance)
3	0.070470	(Action, Adventure, Drama)
1	0.070470	(Action, Adventure, Comedy)
51	0.057047	(Crime, Drama, Thriller)
10	0.057047	(Action, Comedy, Crime)
48	0.057047	(Comedy, Romance)

The 1st row shows that (Action, Adventure, Sci-Fi) has support value of 0.204698 which means it occurred 167 times in the dataset. Let's view all itemset frequency in dataframe

```
In [60]: # Frequently occurring itemsets using Apriori Algorithm
         # Adding new column frequency (number of occurence) of each itemset

         frequent_itemsets_apriori['frequency'] = frequent_itemsets_apriori['support'].mul(816)  # 816 is total of transactio
         frequent_itemsets_apriori['frequency'] = frequent_itemsets_apriori['frequency'].round(0).astype(int)
         frequent_itemsets_apriori = frequent_itemsets_apriori[frequent_itemsets_apriori.columns[[1,2,0]]]

         frequent_itemsets_apriori
```

Out[60]:

	itemsets	frequency	support
7	(Action, Adventure, Sci-Fi)	167	0.204698
31	(Animation, Adventure, Comedy)	96	0.117450
5	(Action, Adventure, Fantasy)	79	0.097315
39	(Comedy)	68	0.083893
45	(Comedy, Drama, Romance)	66	0.080537
3	(Action, Adventure, Drama)	58	0.070470
1	(Action, Adventure, Comedy)	58	0.070470
51	(Crime, Drama, Thriller)	47	0.057047
10	(Action, Comedy, Crime)	47	0.057047
48	(Comedy, Romance)	47	0.057047
13	(Action, Crime, Thriller)	47	0.057047
42	(Comedy, Drama)	44	0.053691
61	(Drama, Romance)	44	0.053691
8	(Action, Adventure, Thriller)	44	0.053691
12	(Action, Crime, Sci-Fi)	8	0.010067
18	(Action, Fantasy, Horror)	8	0.010067
19	(Action, Fantasy, Thriller)	8	0.010067
25	(Adventure, Comedy, Drama)	8	0.010067
27	(Adventure, Comedy, Sci-Fi)	8	0.010067
34	(Biography, Comedy, Drama)	8	0.010067
44	(Comedy, Drama, Music)	8	0.010067
46	(Comedy, Family, Fantasy)	8	0.010067
58	(Drama, Mystery, Romance)	8	0.010067
86	(Action, Adventure, Sci-Fi, Comedy, Romance)	8	0.010067
62	(Drama, Romance, Sci-Fi)	8	0.010067
72	(Action, Adventure, Comedy, Action, Crime, Thr...	8	0.010067
75	(Action, Adventure, Sci-Fi, Action, Adventure,...	8	0.010067
76	(Action, Adventure, Drama, Action, Crime, Drama)	8	0.010067
77	(Comedy, Drama, Romance, Action, Adventure, Fa...	8	0.010067
79	(Action, Adventure, Sci-Fi, Action, Crime, Thr...	8	0.010067
83	(Action, Adventure, Sci-Fi, Biography, Drama)	8	0.010067
84	(Comedy, Action, Adventure, Sci-Fi)	8	0.010067
85	(Comedy, Drama, Action, Adventure, Sci-Fi)	8	0.010067
112	(Action, Adventure, Comedy, Comedy, Comedy, Dr...	8	0.010067

113 rows × 3 columns

- Find frequently occurring itemsets using F-P Growth

  The FP-Growth Algorithm is an alternative way to find frequent itemsets. It uses a divide-and-conquer strategy where the core of this method is the usage of pattern fragment growth named frequent-pattern tree (FP-tree). This method retains the itemset association information using an extended prefix-tree structure for storing information about frequent patterns.

This method is proven to be more efficient and scalable for mining the complete set of frequent patterns over other algorithm such as Apriori Algorithm.

1) Pros: Usually faster than Apriori.
2) Cons: Not possible to hold the FP-tree in the main memory. Partition the database into a set of smaller databases and then construct an FP-tree from each of these smaller databases.
3) Works with Nominal values.

General approach to FP-growth algorithm:

1) Preparation: Discrete data is needed because we're storing sets. For continuous data, it will need to be quantized into discrete values.
2) Train: Build an FP-tree and mine the tree.
3) Test: Doesn't apply.
4) Application: This can be used to identify commonly occurring items that can be used to make decisions, suggest items, make forecasts, and so on.

I have experimented a number trial of minimum support number and 0.01 is the best for this dataset.

In [62]:
```
# Frequently occurring itemsets using F-P Growth (Frequent Pattern Growth)

frequent_itemsets_fpgrowth = fpgrowth(df2, min_support=0.01,
                                      use_colnames=True).sort_values(by='support', ascending=0)
frequent_itemsets_fpgrowth.head(10)
```

Out[62]:

	support	itemsets
39	0.204698	(Action, Adventure, Sci-Fi)
37	0.117450	(Animation, Adventure, Comedy)
45	0.097315	(Action, Adventure, Fantasy)
33	0.083893	(Comedy)
15	0.080537	(Comedy, Drama, Romance)
0	0.070470	(Action, Adventure, Drama)
18	0.070470	(Action, Adventure, Comedy)
1	0.057047	(Action, Crime, Thriller)
32	0.057047	(Crime, Drama, Thriller)
16	0.057047	(Action, Comedy, Crime)

There are 113 number of itemsets found by F-P Growth algorithm in this dataset.

The 1st row shows that (Action, Adventure, Sci-Fi) has support value of 0.204698 which means it occurred 167 times in the dataset. Let's view all itemsets frequency in dataframe

```
In [65]: # Frequently occurring itemsets using F-P Growth (Frequent Pattern Growth)
         # Adding new column number of occurence (frequency) of each itemset

         frequent_itemsets_fpgrowth['frequency'] = frequent_itemsets_fpgrowth['support'].mul(816)  # 816 is total of transact
         frequent_itemsets_fpgrowth['frequency'] = frequent_itemsets_fpgrowth['frequency'].round(0).astype(int)
         frequent_itemsets_fpgrowth = frequent_itemsets_fpgrowth[frequent_itemsets_fpgrowth.columns[[1,2,0]]]

         frequent_itemsets_fpgrowth
```

Out[65]:

	itemsets	frequency	support
39	(Action, Adventure, Sci-Fi)	167	0.204698
37	(Animation, Adventure, Comedy)	96	0.117450
45	(Action, Adventure, Fantasy)	79	0.097315
33	(Comedy)	68	0.083893
15	(Comedy, Drama, Romance)	66	0.080537
0	(Action, Adventure, Drama)	58	0.070470
18	(Action, Adventure, Comedy)	58	0.070470
1	(Action, Crime, Thriller)	47	0.057047
32	(Crime, Drama, Thriller)	47	0.057047
16	(Action, Comedy, Crime)	47	0.057047
44	(Comedy, Romance)	47	0.057047
14	(Drama, Romance)	44	0.053691
63	(Action, Adventure, Thriller)	44	0.053691
19	(Comedy, Drama)	44	0.053691
7	(Adventure, Comedy, Sci-Fi)	8	0.010067
21	(Drama, Mystery, Romance)	8	0.010067
24	(Adventure, Comedy, Drama)	8	0.010067
25	(Drama, Music)	8	0.010067
26	(Action, Crime, Sci-Fi)	8	0.010067
27	(Biography, Comedy, Drama)	8	0.010067
35	(Drama, Horror, Mystery)	8	0.010067
41	(Drama, Romance, Sci-Fi)	8	0.010067
43	(Comedy, Drama, Music)	8	0.010067
59	(Action, Biography, Drama)	8	0.010067
82	(Drama, Romance, Action, Crime, Thriller)	8	0.010067
64	(Comedy, Family, Fantasy)	8	0.010067
70	(Action, Adventure)	8	0.010067
71	(Action, Adventure, Sci-Fi, Action, Adventure,...	8	0.010067
72	(Action, Adventure, Comedy, Action, Crime, Thr...	8	0.010067
73	(Comedy, Romance, Action, Crime, Thriller)	8	0.010067
74	(Action, Adventure, Sci-Fi, Action, Crime, Thr...	8	0.010067
75	(Action, Adventure, Drama, Action, Crime, Drama)	8	0.010067
78	(Comedy, Romance, Crime, Drama, Mystery)	8	0.010067
80	(Comedy, Drama, Drama, Mystery, Sci-Fi)	8	0.010067
112	(Comedy, Drama, Romance, Action, Adventure, Fa...	8	0.010067

113 rows × 3 columns

- Mine the Association Rules

  Association rules analysis is a technique to uncover how items are associated to each other. There are 3 common ways to measure association:

  1) Measure 1: **Support** - This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. If we

discover that genre of certain movie beyond a certain proportion, we might consider using that proportion as our support threshold. Thus, we identify itemsets with support values above this threshold as significant itemsets.

2) Measure 2: **Confidence** - This says how likely genre B is chosen when genre A is chosen, expressed as {A -> B}. This is measured by the proportion of transactions with genre A, in which genre B also appears. One drawback of the confidence measure is that it might misrepresent the importance of an association. This is because it only accounts for how popular A are, but not B. If B are also very popular in general, there will be a higher chance that a transaction containing A will also contain B, thus inflating the confidence measure. To account for the base popularity of both items, we use a third measure called Lift.
3) Measure 3: `Lift`. This says how likely item B is purchased when item A is purchased, while controlling for how popular item B is. This measurement take the account of probability of having B in the basket with knowledge of A being present over the probability of having B in the basket without any knowledge about present of A.
    i. Lift of {A -> B} = 1, means no association between items.
    ii. Lift of {A -> B} > 1, means that item B is likely to be bought if item A is bought,
    iii. Lift of {A -> B} < 1, means that item B is unlikely to be bought if item A is bought

a) Mine the Association Rules using Apriori Algorithm

In [41]:
```
# Generate the Association Rules using Apriori Algorithm with their corresponding support, confidence and lift.

rules_apriori = association_rules(frequent_itemsets_apriori, metric="lift", min_threshold=1)  # min_threshold = mini
rules_apriori = rules_apriori.sort_values(by='lift', ascending=0)
```

In [42]:
```
# View top 5 rules

rules_apriori.head()
```

Out[42]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
71	(Comedy, Drama)	(Comedy, Action, Adventure, Comedy)	0.053691	0.016779	0.010067	0.187500	11.175000	0.009166	1.210119
70	(Comedy, Action, Adventure, Comedy)	(Comedy, Drama)	0.016779	0.053691	0.010067	0.600000	11.175000	0.009166	2.365772
68	(Comedy, Drama, Comedy)	(Action, Adventure, Comedy)	0.013423	0.070470	0.010067	0.750000	10.642857	0.009121	3.718121
73	(Action, Adventure, Comedy)	(Comedy, Drama, Comedy)	0.070470	0.013423	0.010067	0.142857	10.642857	0.009121	1.151007
47	(Comedy, Crime, Drama)	(Comedy, Drama, Romance)	0.020134	0.080537	0.010067	0.500000	6.208333	0.008446	1.838926

Antecedent and a Consequent, both of which are a list of genres. Note that implication here is co-occurrence and not causality.

The maximum value of Lift is 11.1 and maximum value for Confidence is 0.75 found in Association Rules using Apriori Algorithm. I want to view for a large value of Lift and Confidence with range value of more than 1 and more than 0.4 respectively. This means that genre B is likely to be chosen if genre A is chosen

```
In [64]: # Filter the dataframe for Lift > 1 and high confidence >= 0.5

rules_apriori[(rules_apriori['lift'] > 1) &
              (rules_apriori['confidence'] >= 0.5)].sort_values(by='lift', ascending=0)
```

Out[64]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
69	(Comedy, Action, Adventure, Comedy)	(Comedy, Drama)	0.016779	0.053691	0.010067	0.60	11.175000	0.009166	2.365772
68	(Comedy, Comedy, Drama)	(Action, Adventure, Comedy)	0.013423	0.070470	0.010067	0.75	10.642857	0.009121	3.718121
47	(Comedy, Crime, Drama)	(Comedy, Drama, Romance)	0.020134	0.080537	0.010067	0.50	6.208333	0.008446	1.838926
70	(Comedy, Drama, Action, Adventure, Comedy)	(Comedy)	0.020134	0.083893	0.010067	0.50	5.960000	0.008378	1.832215
44	(Comedy, Crime)	(Comedy)	0.020134	0.083893	0.010067	0.50	5.960000	0.008378	1.832215
59	(Drama, Sport)	(Action, Adventure, Sci-Fi)	0.013423	0.204698	0.010067	0.75	3.663934	0.007319	3.181208

Now I want to view small value of Lift and Confidence with range value of less than 1, equal and more than 0 respectively. This means that genre B is unlikely to be chosen if genre A is chosen

```
In [68]: # Filter the dataframe for Lift < 1 and high confidence >= 0

rules_apriori[(rules_apriori['lift'] < 1) &
              (rules_apriori['confidence'] >= 0)].sort_values(by='lift', ascending=1)
```

Out[68]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
79	(Action, Adventure, Sci-Fi)	(Comedy)	0.204698	0.083893	0.010067	0.049180	0.586230	-0.007106	0.963492
78	(Comedy)	(Action, Adventure, Sci-Fi)	0.083893	0.204698	0.010067	0.120000	0.586230	-0.007106	0.903752
16	(Action, Adventure, Sci-Fi)	(Action, Adventure, Fantasy)	0.204698	0.097315	0.013423	0.065574	0.673827	-0.006497	0.966031
17	(Action, Adventure, Fantasy)	(Action, Adventure, Sci-Fi)	0.097315	0.204698	0.013423	0.137931	0.673827	-0.006497	0.922550
68	(Action, Adventure, Drama)	(Action, Adventure, Sci-Fi)	0.070470	0.204698	0.010067	0.142857	0.697892	-0.004358	0.927852
69	(Action, Adventure, Sci-Fi)	(Action, Adventure, Drama)	0.204698	0.070470	0.010067	0.049180	0.697892	-0.004358	0.977609
75	(Action, Adventure, Sci-Fi)	(Action, Crime, Thriller)	0.204698	0.057047	0.010067	0.049180	0.862102	-0.001610	0.991726
65	(Action, Adventure, Sci-Fi)	(Comedy, Romance)	0.204698	0.057047	0.010067	0.049180	0.862102	-0.001610	0.991726
64	(Comedy, Romance)	(Action, Adventure, Sci-Fi)	0.057047	0.204698	0.010067	0.176471	0.862102	-0.001610	0.965724
74	(Action, Crime, Thriller)	(Action, Adventure, Sci-Fi)	0.057047	0.204698	0.010067	0.176471	0.862102	-0.001610	0.965724
81	(Action, Adventure, Sci-Fi)	(Comedy, Drama)	0.204698	0.053691	0.010067	0.049180	0.915984	-0.000923	0.995256
80	(Comedy, Drama)	(Action, Adventure, Sci-Fi)	0.053691	0.204698	0.010067	0.187500	0.915984	-0.000923	0.978833
2	(Animation, Adventure, Comedy)	(Action, Adventure, Sci-Fi)	0.117450	0.204698	0.023490	0.200000	0.977049	-0.000552	0.994128
3	(Action, Adventure, Sci-Fi)	(Animation, Adventure, Comedy)	0.204698	0.117450	0.023490	0.114754	0.977049	-0.000552	0.996955

Rules with Lift value equal to 1

```
In [89]: # Filter the dataframe for Lift == 1 and high confidence >= 0

rules_apriori[(rules_apriori['lift'] == 1) &
              (rules_apriori['confidence'] >= 0)].sort_values(by='lift', ascending=1)
```

Out[89]:

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

Findings 1:

- About 74 rules have a high Lift value (more than 1), which means that it increase the chances of occurence of movie genre in Consequents in spite high Confidence value.
- A value of Lift which greater than 1 indicate for high association between Antecedents and Consequents. The greater the value of Lift, the greater are the chances of preference to choose genre in Consequents. Here, if the viewer has already watched movie with genre of (Comedy, Action, Adventure, Comedy), viewer will likely watch (Comedy, Drama) movie genre.
- Comedy genre has high Lift value
- Lift is the measure that will help movie producer to decide what kind of movie genre to produce next based on revenue generated from an individual movie.
- These 74 rules also have wide range of Confidence number, range between 0.04 to 0.75.
- 14 rules showed that genre fall in antecedent does not increase the chances of viewer to watch genre in consequent. Eg: Viewer who likes to watch (Action, Adventure, Sci-Fi) movie most likely does not watch (Comedy) movie genre.
- There is no Association Rule for Lift value equal to 1

b) Mine the Association Rules using F-P Growth

6b) Mine the Association Rules using F-P Growth

In [74]:
```
rules_fpgrowth = association_rules(frequent_itemsets_fpgrowth, metric="lift", min_threshold=1)
rules_fpgrowth = rules_fpgrowth.sort_values(by='lift', ascending=0)
```

In [75]:
```
rules_fpgrowth
```

Out[75]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
31	(Comedy, Action, Adventure, Comedy)	(Comedy, Drama)	0.016779	0.053691	0.010067	0.600000	11.175000	0.009166	2.365772
34	(Comedy, Drama)	(Comedy, Action, Adventure, Comedy)	0.053691	0.016779	0.010067	0.187500	11.175000	0.009166	1.210119
30	(Comedy, Comedy, Drama)	(Action, Adventure, Comedy)	0.013423	0.070470	0.010067	0.750000	10.642857	0.009121	3.718121
35	(Action, Adventure, Comedy)	(Comedy, Comedy, Drama)	0.070470	0.013423	0.010067	0.142857	10.642857	0.009121	1.151007
57	(Comedy, Crime, Drama)	(Comedy, Drama, Romance)	0.020134	0.080537	0.010067	0.500000	6.208333	0.008446	1.838926
56	(Comedy, Drama, Romance)	(Comedy, Crime, Drama)	0.080537	0.020134	0.010067	0.125000	6.208333	0.008446	1.119847
71	(Drama, Mystery, Sci-Fi)	(Comedy, Drama)	0.030201	0.053691	0.010067	0.333333	6.208333	0.008446	1.419463
70	(Comedy, Drama)	(Drama, Mystery, Sci-Fi)	0.053691	0.030201	0.010067	0.187500	6.208333	0.008446	1.193598
48	(Comedy, Crime)	(Comedy)	0.020134	0.083893	0.010067	0.500000	5.960000	0.008378	1.832215
32	(Comedy, Drama, Action,	(Comedy)	0.020134	0.083893	0.010067	0.500000	5.960000	0.008378	1.832215

The maximum value of Lift is 11.1 and maximum value for Confidence is 1.0 found in Association Rules using F-P Growth Algorithm same like the Apriori Algorithm.

I want to view for a large value of Lift and Confidence with range value of more than 1 and more than 0.8 respectively. This means that genre B is likely to be chosen if genre A is chosen

In [151]:
```
# Filter the dataframe for Lift > 1 and high confidence >= 0.5

rules_fpgrowth[(rules_fpgrowth['lift'] > 1) &
               (rules_fpgrowth['confidence'] >= 0.5)].sort_values(by='lift', ascending=0)
```

Out[151]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
37	(Comedy, Action, Adventure, Comedy)	(Comedy, Drama)	0.016779	0.053691	0.010067	0.60	11.175000	0.009166	2.365772
36	(Comedy, Comedy, Drama)	(Action, Adventure, Comedy)	0.013423	0.070470	0.010067	0.75	10.642857	0.009121	3.718121
66	(Comedy, Crime, Drama)	(Comedy, Drama, Romance)	0.020134	0.080537	0.010067	0.50	6.208333	0.008446	1.838926
38	(Comedy, Drama, Action, Adventure, Comedy)	(Comedy)	0.020134	0.083893	0.010067	0.50	5.960000	0.008378	1.832215
57	(Comedy, Crime)	(Comedy)	0.020134	0.083893	0.010067	0.50	5.960000	0.008378	1.832215
50	(Drama, Sport)	(Action, Adventure, Sci-Fi)	0.013423	0.204698	0.010067	0.75	3.663934	0.007319	3.181208

For rules with Lift value less than 1 and Confidence is equal and more than 0

In [36]:
```
# Filter the dataframe for Lift < 1 and high confidence >= 0

rules_fpgrowth[(rules_fpgrowth['lift'] < 1) &
              (rules_fpgrowth['confidence'] >= 0)].sort_values(by='lift', ascending=1)
```

Out[36]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
63	(Comedy)	(Action, Adventure, Sci-Fi)	0.083893	0.204698	0.010067	0.120000	0.586230	-0.007106	0.903752
62	(Action, Adventure, Sci-Fi)	(Comedy)	0.204698	0.083893	0.010067	0.049180	0.586230	-0.007106	0.963492
16	(Action, Adventure, Sci-Fi)	(Action, Adventure, Fantasy)	0.204698	0.097315	0.013423	0.065574	0.673827	-0.006497	0.966031
17	(Action, Adventure, Fantasy)	(Action, Adventure, Sci-Fi)	0.097315	0.204698	0.013423	0.137931	0.673827	-0.006497	0.922550
72	(Action, Adventure, Sci-Fi)	(Action, Adventure, Drama)	0.204698	0.070470	0.010067	0.049180	0.697892	-0.004358	0.977609
73	(Action, Adventure, Drama)	(Action, Adventure, Sci-Fi)	0.070470	0.204698	0.010067	0.142857	0.697892	-0.004358	0.927852
46	(Action, Adventure, Sci-Fi)	(Comedy, Romance)	0.204698	0.057047	0.010067	0.049180	0.862102	-0.001610	0.991726
78	(Action, Adventure, Sci-Fi)	(Action, Crime, Thriller)	0.204698	0.057047	0.010067	0.049180	0.862102	-0.001610	0.991726
47	(Comedy, Romance)	(Action, Adventure, Sci-Fi)	0.057047	0.204698	0.010067	0.176471	0.862102	-0.001610	0.965724
79	(Action, Crime, Thriller)	(Action, Adventure, Sci-Fi)	0.057047	0.204698	0.010067	0.176471	0.862102	-0.001610	0.965724
32	(Action, Adventure, Sci-Fi)	(Comedy, Drama)	0.204698	0.053691	0.010067	0.049180	0.915984	-0.000923	0.995256
33	(Comedy, Drama)	(Action, Adventure, Sci-Fi)	0.053691	0.204698	0.010067	0.187500	0.915984	-0.000923	0.978833
1	(Animation, Adventure, Comedy)	(Action, Adventure, Sci-Fi)	0.117450	0.204698	0.023490	0.200000	0.977049	-0.000552	0.994128
0	(Action, Adventure, Sci-Fi)	(Animation, Adventure, Comedy)	0.204698	0.117450	0.023490	0.114754	0.977049	-0.000552	0.996955

For rules with Lift value equal to 1

```
In [37]: # Filter the dataframe for Lift == 1 and high confidence >= 0

         rules_fpgrowth[(rules_fpgrowth['lift'] == 1) &
                   (rules_fpgrowth['confidence'] >= 0)].sort_values(by='lift', ascending=1)
Out[37]:
```

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

Findings 2:

- The output from F-P Growth algorithm is same like Apriori algorithm for this case study. About 74 rule have a high Lift value (more than 1), which means that it increase the chances of occurrence of movie genre in `Consequents` in spite high `Confidence` value.
- These 74 rules of high Lift value also have wide range of Confidence number, range between 0.04 to 0.75.
- F-P Growth algorithm showed faster in processing the data than Apriori algorithm because it scan the database twice to generate the itemsets unlike Apriori scans multiple times over database to generate itemsets.
- There are about 4 rules with Lift value less than 1
- There is no Association Rule for Lift value equal to 1

(10 marks)

5. It will be appeared in week 14.

(10 marks)

Submissions:

The student is expected to submit answers to each question individually, and submit the document in PDF format. The student can include online materials, screenshots, videos and/or codes (ipynb format) to support your answer