Jana Chatterjee, Rohini Mutur, Maria A. Zamudio

Disney Movies ETL and Analysis

Objective

On November 8th, 2018 Disney announced the launch of its own streaming service “Disney+” to be available in North America by Nov-2019.For that reason we have decided to scrape some Disney movies’ data that would enables us to Extract, Transform and Load data into clean and usable datasets, in order to analyze it.

Datasets

Below are the steps followed after we decided the information needed to ingest in order to achieve analysis: Revenue, Year, Ratings, Voice Actors:

1. Extract: Source research
    a. IMDB
    b. Wikipedia - Disney Movies
    c. Wikipedia - Movie Gross Sales
    d. Disneymovieslist.com
    e. Kaggle - Disney Characters & Voice Actors (CSV)
2. Transformation: Cleansing
3. Load: SQL Workbench Data Load

Extraction:

1. IMDB Web Scraping

    1. Using Splinter we accessed a Disney Movies predetermined list from IMDB: 'https://www.imdb.com/list/ls026785255/'
    2. Beautiful Soup was used in order to scrape the website

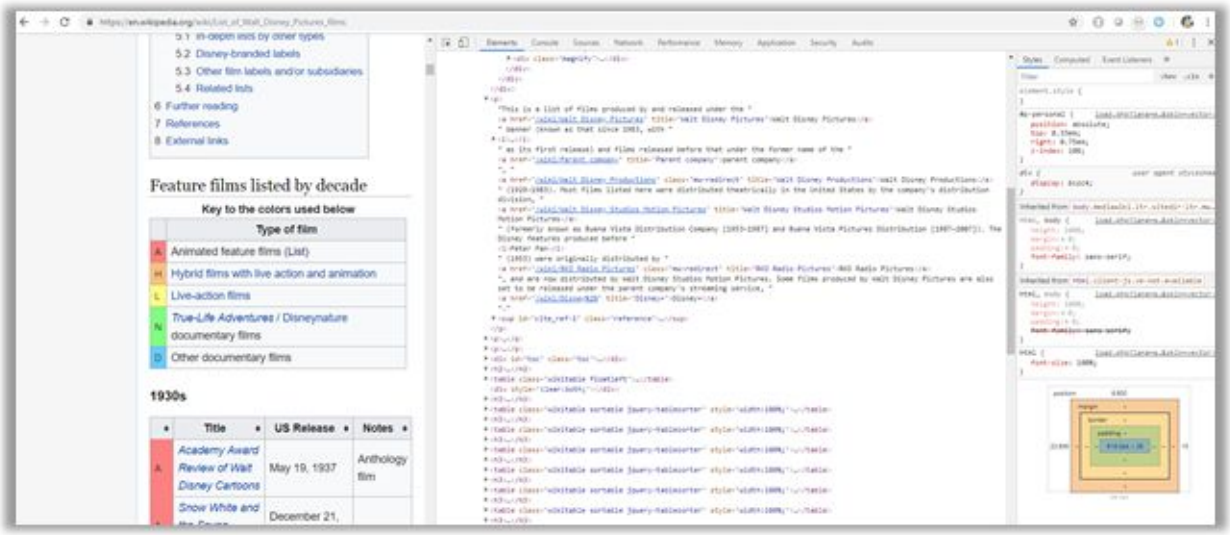
```
from splinter import Browser
from bs4 import BeautifulSoup as bs
import pandas as pd
import time

!which chromedriver
executable_path = {'executable_path': '/usr/local/bin/chromedriver'}
browser = Browser('chrome', **executable_path, headless=False)
```

```
/usr/local/bin/chromedriver
```

```
#url_disney = 'https://www.imdb.com/search/title-text?trivia=Disney+movie+'
url_disney = 'https://www.imdb.com/list/ls026785255/'
browser.visit(url_disney)


for _ in range(0,2):

    html = browser.html
    soup = bs(html,"html.parser")
```

2. Wikipedia Data Scraping

Inspect HTML



Imports & URL Inspection with Splinter & Beautiful Soup

```
In [1]: # Dependencies
        import pandas as pd
        from bs4 import BeautifulSoup
        from splinter import Browser
        import urllib.request
        import requests
        import time

In [2]: # Outline the path for chromedriver so splinter can run and read through the URL pages
        executable_path = {"executable_path": "chromedriver"}
        browser = Browser("chrome", **executable_path, headless=False)

In [4]: # First, Let's go to the Nasa News URL
        url = "https://en.wikipedia.org/wiki/List_of_Walt_Disney_Pictures_films"
        browser.visit(url)

In [5]: # Scrape Time
        html = browser.html
        soup = BeautifulSoup(html, 'html.parser')

In [9]: # Check out the soup 🥣
        print(soup.prettify())

Out[9]: <!DOCTYPE html>
        <html class="client-js ve-not-available" dir="ltr" lang="en" xmlns="http://www.w3.org/1999/xhtml"><head>
        <meta charset="utf-8"/>
        <title>List of Walt Disney Pictures films - Wikipedia</title>
        <script>document.documentElement.className = document.documentElement.className.replace( /(^|\s)client-nojs(\s|$)/, "$1client
        -js$2" );</script>
        <script>(window.RLQ=window.RLQ||[]).push(function(){mw.config.set({"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":fal
        se,"wgNamespaceNumber":0,"wgPageName":"List_of_Walt_Disney_Pictures_films","wgTitle":"List of Walt Disney Pictures films","wg
        CurRevisionId":892071592,"wgRevisionId":892071592,"wgArticleId":1970335,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"v
        iew","wgUserName":null,"wgUserGroups":["*"],"wgCategories":["Wikipedia indefinitely semi-protected pages","Use American Engli
        sh from November 2018","All Wikipedia articles written in American English","Use mdy dates from December 2018","Pages contain
        ing links to subscription-only content","Walt Disney Pictures films","Lists of films by studio","Disney-related lists","Walt
        Disney Pictures","American films by studio"],"wgBreakFrames":false,"wgPageContentLanguage":"en","wgPageContentModel":"wikitex
        t","wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","Januar
        y","February","March","April","May","June","July","August","September","October","November","December"],"wgMonthNamesShort":
        ["","Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"],"wgRelevantPageName":"List_of_Walt_Disney_Pictur
        es_films","wgRelevantArticleId":1970335,"wgRequestId":"XLAbsApAMFwAAKtN6HwAAABF","wgCSPNonce":false,"wgIsProbablyEditable":fa
        lse,"wgRelevantPageIsProbablyEditable":false,"wgRestrictionEdit":["autoconfirmed"],"wgRestrictionMove":["autoconfirmed"],"wgF
        laggedRevsParams":{"tags":{}},"wgStableRevisionId":null,"wgBetaFeaturesFeatures":[],"wgMediaViewerOnClick":true,"wgMediaViewe
```

Tables with Pandas

```
In [16]: # Tables with Pandas
         tables = pd.read_html(url)
         tables

Out[16]: [     0                                                  1
          0  NaN                                         Type of film
          1    A                          Animated feature films (List)
          2    H          Hybrid films with live action and animation
          3    L                                     Live-action films
          4    N  True-Life Adventures / Disneynature documentar...
          5    D                                Other documentary films,
                 0                                                  1                 2  \
          0  NaN                                              Title         US Release
          1    A  Academy Award Review of Walt Disney Cartoons       May 19, 1937
          2    A                 Snow White and the Seven Dwarfs  December 21, 1937

                        3
          0           Notes
          1  Anthology film
          2             NaN  ,
                 0                                  1                 2  \
          0  NaN                                Title         US Release
          1    A                            Pinocchio    February 7, 1940
```

DataFrame & Slicing

```
In [34]: #Slice and Dice, Make it Nice
         df = tables[3]
         df.columns = ['Type of Film','Title', 'US Release', 'Notes']
         df.head(100)
```

Out[34]:

	Type of Film	Title	US Release	Notes
0	NaN	Title	US Release	Notes
1	A	Cinderella	February 15, 1950	NaN
2	L	Treasure Island	July 29, 1950	NaN
3	A	Alice in Wonderland	July 28, 1951	NaN
4	L	The Story of Robin Hood and His Merrie Men	June 26, 1952	NaN
5	A	Peter Pan	February 5, 1953	NaN
6	L	The Sword and the Rose	July 23, 1953	NaN
7	N	The Living Desert	November 10, 1953	NaN
8	L	Rob Roy, the Highland Rogue	February 27, 1954	NaN
9	N	The Vanishing Prairie	August 16, 1954	NaN
10	L	20,000 Leagues Under the Sea	December 23, 1954	NaN
11	L	Davy Crockett, King of the Wild Frontier	May 25, 1955	Compilation film mostly made up from pre-exist...
12	A	Lady and the Tramp	June 22, 1955	NaN
13	N	The African Lion	September 14, 1955	NaN
14	L	The Littlest Outlaw	December 22, 1955	NaN
15	L	The Great Locomotive Chase	June 8, 1956	NaN
16	L	Davy Crockett and the River Pirates	July 18, 1956	Compilation film mostly made up from pre-exist...
17	N	Secrets of Life	November 6, 1956	NaN
18	L	Westward Ho the Wagons!	December 20, 1956	NaN

Drop NaN's & Unneeded Columns

```
In [48]: #-----------------------------------------------#
         # It's Cleaning Time, Coderella ***
         #-----------------------------------------------#

         # Make a DataFrame and only include the columns you need to use.

         disney_pd = disney_pd[['Type of Film',
                                'Title',
                                'US Release',
                                ]]

         disney_pd = pd.DataFrame(disney_pd)

         # Check it out
         disney_pd.isnull().sum()

Out[48]: Type of Film    1
         Title           0
         US Release      0
         dtype: int64

In [50]: # K, now exit all the NaNs and preview
         disney_pd['Type of Film'].dropna()
         disney_pd = disney_pd.dropna()
         disney_pd.head(100)
```

Out[50]:

	Type of Film	Title	US Release
1	L	Toby Tyler	January 21, 1960
2	L	Kidnapped	February 24, 1960
3	L	Pollyanna	May 19, 1960
4	L	The Sign of Zorro	June 11, 1960
5	N	Jungle Cat	August 10, 1960
6	L	Ten Who Dared	November 1, 1960
7	L	Swiss Family Robinson	December 21, 1960
8	A	One Hundred and One Dalmatians	January 25, 1961
9	L	The Absent-Minded Professor	March 16, 1961

3. Wikipedia Gross Revenue Data Scraping :

   > We scraped the gross revenue information of Disney movies over the years from Wikipedia

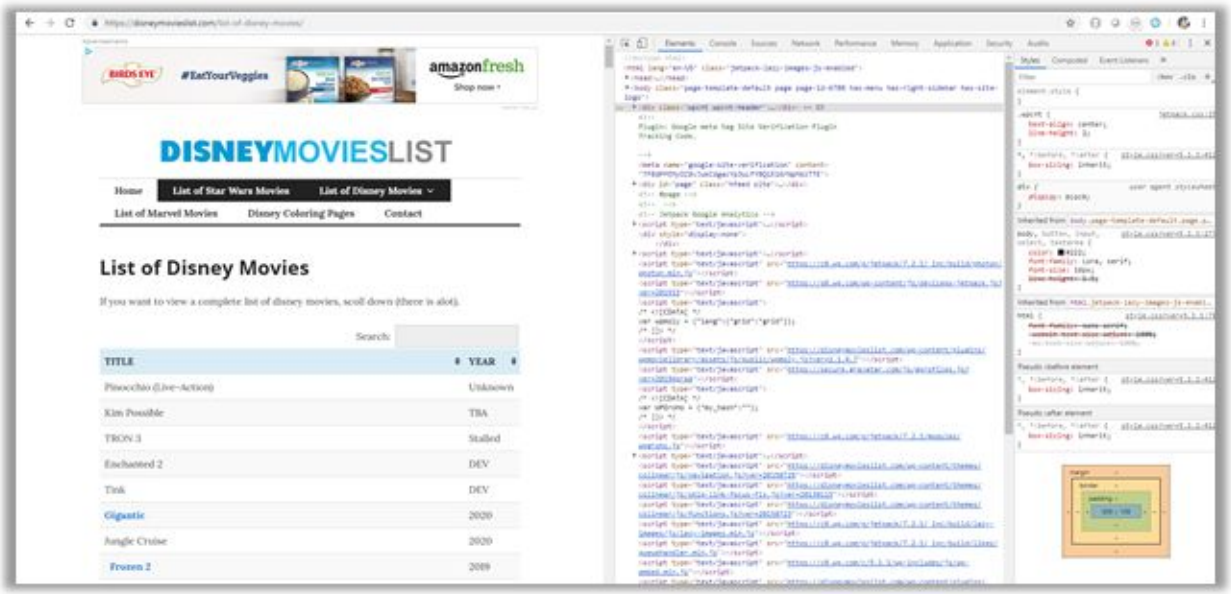
```
import pandas as pd
import re
from sqlalchemy import create_engine
import pymysql
pymysql.install_as_MySQLdb()

url='https://en.wikipedia.org/wiki/List_of_highest-grossing_animated_films'

tables=pd.read_html(url)
movies_df=tables[4]
```

Out[41]:

	Year	Title	Worldwide gross	Budget	Ref(s)
1	1937	Snow White and the Seven Dwarfs	418, 200, 000+(8,500,000)R	$1,488,423	[# 125][# 170][# 171]
2	1938	NaN	NaN	NaN	NaN
3	1939	Gulliver's Travels	$3,200,000*	$700,000	[# 172][# 173]
4	1940	Pinocchio	87, 000, 862∗(3,500,000)R	$2,600,000	[# 158][# 171][# 174]
5	1941	Dumbo	$1,600,000*	$950,000	[# 175][# 176]

4. List Disney Scraping

Inspect HTML



Imports & URL Inspection with Beautiful Soup

```
In [1]: # Dependencies
        import pandas as pd
        from bs4 import BeautifulSoup
        from splinter import Browser
        import urllib.request
        import requests
        import time

In [2]: # URL of page to be scraped
        url = 'https://disneymovieslist.com/list-of-disney-movies/'

In [3]: # Retrieve page with the requests module
        response = requests.get(url)

In [4]: # Check the resposne - 200 means it worked
        response

Out[4]: <Response [200]>

In [8]: # Scrape Time
        soup = BeautifulSoup(response.text, 'html.parser')

In [9]: # Check out the soup
        print(soup.prettify())

        <!DOCTYPE html>
        <html lang="en-US">
         <head>
          <meta charset="utf-8"/>
          <meta content="width=device-width, initial-scale=1" name="viewport"/>
          <link href="http://gmpg.org/xfn/11" rel="profile"/>
          <link href="https://disneymovieslist.com/xmlrpc.php" rel="pingback"/>
          <title>
           List of Disney Movies - Disney Movies List
          </title>
          <link href="//s0.wp.com" rel="dns-prefetch">
           <link href="//secure.gravatar.com" rel="dns-prefetch">
            <link href="//fonts.googleapis.com" rel="dns-prefetch"/>
            <link href="//s.w.org" rel="dns-prefetch"/>
            <link href="https://disneymovieslist.com/feed/" rel="alternate" title="Disney Movies List » Feed" type="application/rss+x
        ml"/>
            <link href="https://disneymovieslist.com/comments/feed/" rel="alternate" title="Disney Movies List » Comments Feed" type
        ="application/rss+xml"/>
            <script type="text/javascript">
```

Tables with Pandas

```
In [14]: # K, now put into tables with pandas
         tables = pd.read_html(url)
         tables

Out[14]: [
                                        TITLE     YEAR
          0                 Pinocchio (Live-Action)  Unknown
          1                            Kim Possible      TBA
          2                                  TRON 3  Stalled
          3                           Enchanted 2      DEV
          4                                    Tink      DEV
          5                               Gigantic     2020
          6                           Jungle Cruise    2020
          7                                 Frozen 2   2019
          8                     Dumbo (Live-Action)    2019
          9                           Descendants 3    2019
          10                    Mulan (Live-Action)    2019
          11                            Toy Story 4    2019
          12                           The Lion King   2019
          13                                 Aladdin   2019
          14                     Mary Poppins Returns  2018
          15                                Magic Camp 2018
          16                    Cruella (Live-Action)  2018
          17       Wreck-It Ralph 2: Ralph Breaks the Internet 2018
          18                            Freaky Friday  2018
          19                        A Wrinkle in Time  2018
          20                                  Zombies  2018
          21                           Incredibles 2   2018
          22                                  Cars 3   2017
          23                     Beauty and the Beast  2017
          24                                    Coco   2017
          25                           Descendants 2   2017
          26                 Tangled: Before Ever After 2017
          27  Pirates of the Caribbean: Dead Men Tell No Tales 2017
          28                                Zootopia   2016
          29                              Jungle Book  2016
```

DataFrame & Slicing

```
In [15]: #Slice and Dice, Make it Nice
         disney_pd = tables[0]
         disney_pd.columns = ['TITLE', 'YEAR']
         disney_pd.head(100)

Out[15]:
```

	TITLE	YEAR
0	Pinocchio (Live-Action)	Unknown
1	Kim Possible	TBA
2	TRON 3	Stalled
3	Enchanted 2	DEV
4	Tink	DEV
5	Gigantic	2020
6	Jungle Cruise	2020
7	Frozen 2	2019
8	Dumbo (Live-Action)	2019
9	Descendants 3	2019
10	Mulan (Live-Action)	2019
11	Toy Story 4	2019
12	The Lion King	2019
13	Aladdin	2019
14	Mary Poppins Returns	2018
15	Magic Camp	2018
16	Cruella (Live-Action)	2018
17	Wreck-It Ralph 2: Ralph Breaks the Internet	2018
18	Freaky Friday	2018
19	A Wrinkle in Time	2018

Check for NaN's

```
In [16]: # Count Stuff
         disney_pd.count()

Out[16]: TITLE    562
         YEAR     562
         dtype: int64

In [17]: # Checkout the Nans
         disney_pd.isnull().sum()

Out[17]: TITLE    0
         YEAR     0
         dtype: int64
```

Transformation:

1. IMDB Web Scraping

1. Created a Loop that scrapes the 2 pages available for the IMDB data.
2. Created an additional Loop that scrapes the Titles, IMDB Ratings, Metascore Ratings, Years and Links:
   a. Title: By inspecting the website we were able to determine the tags of the titles ("h3",class_="lister-item-header") and found out that the titles from each movie has a "Year" appended.
      i. Removed all the leading and trailing spaces from the string (STRIP)
      ii. Replaced '\n' with blank spaces.
      iii. Splitted the blank spaces previously created
      iv. Extracted the year from the titles and removed it

```
movie_title = movie.find("h3",class_="lister-item-header")
if movie_title is not None:
    movie_title = movie_title.text.strip().replace('\n', ' ') #replace characters
    movie_split = movie_title.split(' ')
    len_movie_split = len(movie_split)
    movie_title = ' '.join(movie_split[1:-1]) #extract the year and remove from title
    year = movie_split[-1] #grab the year from the movie
```

   b. Metascore: Refers to the Metascore Rating. The tag is as follows ("div", class_="inline-block ratings-metascore")
      i. Removed all the leading and trailing spaces from the string (STRIP)

ii. The string is “7.3 Metascore”. We only needed “7.3” therefore we splitted the string and bring the first sub-string ([0])

```
met ceevee.zip movie.find("div", class_="inline-block ratings-metascore")
if metascore is not None:
    metascore = metascore.text.strip().split(' ')[0]
```

c. IMDB: Refers to the IMDB Rating. The tag is as follows ("span", class_="ipl-rating-star__rating") and no transformation was needed.

```
imdb = movie.find("span", class_="ipl-rating-star__rating")
if imdb is not None:
    imdb = imdb.text.strip()
#content_div = movie.find("div",class_="lister-item-content")
url_extract = movie.find('a')['href']
url_extract

# append the link to the original website
original_url = 'https://www.imdb.com'
movie_url2 = original_url + url_extract
```

d. URL_Extract: Refers to the IMDB link of each movie. The tag is as follows ('a')['href'], and it was appended to the original link: 'https://www.imdb.com'

3. Created Dataframe with the fields mentioned above.

```
data = {'Title': movie_title, 'year': year, 'Metascore': metascore, 'IMDB':imdb, 'link': movie_url2}
x = pd.DataFrame(data,index = [0])
df = pd.concat((df, x), axis=0)
```

4. Finish the loop by adding the “Next” button, which basically says that once data has been appended from the first page it should move to the next one.

```
browser.click_link_by_partial_text("Next")
```

5. Outcome:

Title	year	Metascore	IMDB	link
Snow White and the Seven Dwarfs	(1937)	95	7.6	https://www.imdb.com/title/tt0029583/?ref_=ttl...
Pinocchio	(1940)	99	7.5	https://www.imdb.com/title/tt0032910/?ref_=ttl...
Fantasia	(1940)	96	7.8	https://www.imdb.com/title/tt0032455/?ref_=ttl...
Dumbo	(1941)	96	7.3	https://www.imdb.com/title/tt0033563/?ref_=ttl...
Bambi	(1942)	91	7.3	https://www.imdb.com/title/tt0034492/?ref_=ttl...
Victory Through Air Power	(1943)	None	6.7	https://www.imdb.com/title/tt0036497/?ref_=ttl...
The Three Caballeros	(1944)	85	6.5	https://www.imdb.com/title/tt0038166/?ref_=ttl...
Make Mine Music	(1946)	60	6.4	https://www.imdb.com/title/tt0038718/?ref_=ttl...
Song of the South	(1946)	54	7.3	https://www.imdb.com/title/tt0038969/?ref_=ttl...

2. Transform & Load Template (All CSV's)

- Created Transform & Load Template for all group CSV's
- See Challenges & Limitations

In [1]:
```
import pandas as pd
from sqlalchemy import create_engine
import pymysql
pymysql.install_as_MySQLdb()
```

In [2]:
```
wikidiz_file = "WikiDisney.csv"
wikidiz_df = pd.read_csv(wikidiz_file)
wikidiz_df.head(25)
```

Out[2]:

	Type of Film	Title	US Release
0	L	Toby Tyler	January 21, 1960
1	L	Kidnapped	February 24, 1960
2	L	Pollyanna	May 19, 1960
3	L	The Sign of Zorro	June 11, 1960
4	N	Jungle Cat	August 10, 1960
5	L	Ten Who Dared	November 1, 1960
6	L	Swiss Family Robinson	December 21, 1960
7	A	One Hundred and One Dalmatians	January 25, 1961

```
In [3]: listdiz_file = "disneylist.csv"
        listdiz_df = pd.read_csv(listdiz_file)
        listdiz_df.head(25)
```

Out[3]:

	TITLE	YEAR
0	Pinocchio (Live-Action)	Unknown
1	Kim Possible	TBA
2	TRON 3	Stalled
3	Enchanted 2	DEV
4	Tink	DEV
5	Gigantic	2020
6	Jungle Cruise	2020
7	Frozen 2	2019
8	Dumbo (Live-Action)	2019

```
In [4]: imdbdiz_file = "IMDBDisney.csv"
        imdbdiz_df = pd.read_csv(imdbdiz_file,encoding="UTF-8")
        imdbdiz_df.head()
```

Out[4]:

	Title	year	Metascore	IMDB	link
0	The Jungle Book	-2016	77	7.4	https://www.imdb.com/title/tt3040964/?ref_=tx_...
1	The Black Cauldron	-1985	59	6.5	https://www.imdb.com/title/tt0088814/?ref_=tx_...
2	Frozen (I)	-2013	74	7.5	https://www.imdb.com/title/tt2294629/?ref_=tx_...
3	Christopher Robin	-2018	60	7.3	https://www.imdb.com/title/tt4575576/?ref_=tx_...
4	Beauty and the Beast	-2017	65	7.2	https://www.imdb.com/title/tt2771200/?ref_=tx_...

In [5]:
```
grossdiz_file = "grossmovie.csv"
grossdiz_df = pd.read_csv(grossdiz_file,encoding="UTF-8")
grossdiz_df.head()
```

Out[5]:

	Year	Title	Worldwide gross	Budget	Total Sales
0	1937	Snow White and the Seven Dwarfs	418,200,000+(8,500,000)R	$1,488,423	426700000
1	1939	Gulliver's Travels	$3,200,000*	$700,000	3200000
2	1940	Pinocchio	87,000,862+(3,500,000)R	$2,600,000	90500862
3	1941	Dumbo	$1,600,000*	$950,000	1600000
4	1942	Bambi	267,447,150(3,449,353)R	$1,700,000–2,000,000	270896503

In [6]:
```
chardiz_file = "dischar.csv"
chardiz_df = pd.read_csv(chardiz_file,encoding="UTF-8")
chardiz_df.head()
```

Out[6]:

	movie_title	release_date	hero	villian	song
0	\nSnow White and the Seven Dwarfs	21-Dec-37	Snow White	Evil Queen	Some Day My Prince Will Come
1	\nPinocchio	7-Feb-40	Pinocchio	Stromboli	When You Wish upon a Star
2	\nFantasia	13-Nov-40	NaN	Chernabog	NaN
3	Dumbo	23-Oct-41	Dumbo	Ringmaster	Baby Mine
4	\nBambi	13-Aug-42	Bambi	Hunter	Love Is a Song

In [7]:
```
voiceactdiz_file = "disney_voice_actors.csv"
voiceactdiz_df = pd.read_csv(voiceactdiz_file)
voiceactdiz_df.head(25)
```

Out[7]:

	character	voice_actor	movie
0	Abby Mallard	Joan Cusack	Chicken Little
1	Abigail Gabble	Monica Evans	The Aristocats
2	Abis Mal	Jason Alexander	The Return of Jafar
3	Abu	Frank Welker	Aladdin
4	Achilles	None	The Hunchback of Notre Dame

```
In [9]: connection_string = "root:password1@127.0.0.1/disney"
        engine = create_engine(f'mysql:///{connection_string}')

In [10]: engine.table_names()

Out[10]: ['dis_characters',
          'dis_voices',
          'disneygrosssales',
          'disneylist',
          'imdbdisney',
          'wikidisney']

In [11]: # Send the Wikipedia Disney data to SQL
         wikidiz_df.to_sql(name='wikidisney', con=engine, if_exists='append', index=False)

In [12]: # Send the Disney List data to SQL
         listdiz_df.to_sql(name='disneylist', con=engine, if_exists='append', index=False)

In [13]: # Send the IMDB Disney data to SQL
         imdbdiz_df.drop('year',axis=1).to_sql(name='imdbdisney', con=engine, if_exists='append', index=False)

In [14]: imdbdiz_df.head(27).tail(4)

Out[14]:
```

	Title	year	Metascore	IMDB	link
5	Cinderella (I)	-2015	67	6.9	https://www.imdb.com/title/tt1661199/?ref_=tx_...
6	Robin Hood	-1973	57	7.6	https://www.imdb.com/title/tt0070608/?ref_=tx_...
7	Tarzan	-1999	79	7.3	https://www.imdb.com/title/tt0120855/?ref_=tx_...
8	Winnie the Pooh	-2011	74	7.2	https://www.imdb.com/title/tt1449283/?ref_=tx_...

```
In [15]: # Send the Gross Movie Sales data to SQL
         grossdiz_df.to_sql(name='disneygrosssales', con=engine, if_exists='append', index=False)

         ---------------------------------------------------------------------------
         UnicodeEncodeError                        Traceback (most recent call last)
         <ipython-input-15-06898427b4c7> in <module>
               1 # Send the Gross Movie Sales data to SQL
         ----> 2 grossdiz_df.to_sql(name='disneygrosssales', con=engine, if_exists='append', index=False)
```

```
In [16]: # # Send the Disney Character data to SQL
         chardiz_df.to_sql(name='dis_characters', con=engine, if_exists='append', index=False)

         ---------------------------------------------------------------------------
         InternalError                             Traceback (most recent call last)
         ~\Anaconda3\lib\site-packages\sqlalchemy\engine\base.py in _execute_context(self, dialect, constructor, statement, parameter
         s, *args)
            1223                    self.dialect.do_executemany(
         -> 1224                        cursor, statement, parameters, context
            1225                    )

         ~\Anaconda3\lib\site-packages\sqlalchemy\dialects\mysql\mysqldb.py in do_executemany(self, cursor, statement, parameters, con
         text)
             131     def do_executemany(self, cursor, statement, parameters, context=None):
         --> 132         rowcount = cursor.executemany(statement, parameters)
             133         if context is not None:

         ~\Anaconda3\lib\site-packages\pymysql\cursors.py in executemany(self, query, args)
             196                                         self.max_stmt_length,
         --> 197                                         self._get_db().encoding)
             198

In [17]: # # Send the Disney Voice Actor data to SQL
         voiceactdiz_df.to_sql(name='dis_voices', con=engine, if_exists='append', index=False)
```

3. Wikipedia Data Scraping for Gross Sales:

1. We started by removing some unnecessary columns from the dataset like Ref[s] Column
2. Dropped rows with Title as ‘TBD’.
3. Removed column ‘Distributor Rental’ and added a new column ‘Total Sales’

```
movies_df['Worldwide gross'].head(10)

movies_df = movies_df.dropna(subset=['Title'])
movies_df.head()

movies_df.drop(movies_df.loc[movies_df['Title']=='TBD'].index,inplace=True)
movies_df=movies_df.reset_index(drop=True)
movies_df.head()
movies_df=movies_df.drop(['Ref(s)'],axis=1)
movies_df=movies_df.rename(columns={'Distributor Rental':'Total Sales'})
movies_df.to_csv('Movies_New.csv')

movies_df.head()
```

4. We had to do some major transformations with the “Worldwide Gross” column as it had sales values in different currencies and with various conditionals:

```
1                                $3,200,000*
2                 $87,000,862* ($3,500,000)R
3                                $1,600,000*
4                $267,447,150 ($3,449,353)R
5                                  $799,000*
6                                $3,355,000R
7                  ES€5,595,283ES (~$90,000)
8                                $3,275,000R
9                                $3,165,000R
10                               $2,560,000R
11                               $1,625,000R
12    $263,591,415($20,000,000/$7,800,000)*R
13                               $2,400,000*
14                $145,000,000 ($7,000,000)
15               $187,000,000 ($6,500,000)*R
16                $51,600,000* ($5,300,000)R
17                              $215,880,212
18             $22,182,353*R ($13,050,777)*R
19       $1,940,903*-2,438,233*($1,130,000)R
20                               $2,764,684*
21                $378,000,000($23,800,000)R
22               SEK1,270,971SW (~$245,000)H
23                              $12,000,000*
24               $191,000,000 ($26,462,000)R
25               SEK1,202,319 SW (~$253,000)
26                               $90,000,000
27               $32,056,467* ($17,160,000)R
28          SEK5,813,000SW (~$2,675,205.50)H
```

5. We did the cleaning using python functions and some pattern matching as below:

```
#movies_df.loc[df['Title']=='TBD'].index, inplace=True

total_gross=[]
rental_distribution=[]
temp_array=[]
for g in gross_list:
    g_i=g.split()
        #print(g_i)
    n0=re.sub("\D", "",g_i[0])


    if len(g_i)>1:
        n1=re.sub("\D", "",g_i[1])
    else:
        n1=0
    temp_array.append(f" {n0}+{n1}")

movies_df['temp']=temp_array
inter_list=movies_df['temp'].str.split('+')[0]

inter_list = list(map(int, inter_list))
inter_list

movies_df.head()
```

```
new_sum_list=[]
for t in temp_array:
    int_list=t.split('+')
    try:
        int_list = list(map(int, int_list))
        new_sum_list.append(int_list[0]+int_list[1])
        #print(f"{t}....{int_list[0]+int_list[1]}")
    except:
        #print(f"original {t}")
        new_sum_list.append(t)
movies_df['Total Sales']=new_sum_list
#movies_df.to_csv('Movies_new_again.csv')

movies_df.drop(['temp'],axis=1)
movies_df.head()
```

6. Then we finally had a clean extract of data with Total Sales column being populated with the summated gross sales values from the 'Worldwide Gross' column as below:

	Year	Title	Worldwide gross	Budget	Total Sales
0	1937	Snow White and the Seven Dwarfs	418, 200, 000+(8,500,000)R	$1,488,423	426700000
1	1939	Gulliver's Travels	$3,200,000*	$700,000	3200000
2	1940	Pinocchio	87, 000, 862∗(3,500,000)R	$2,600,000	90500862
3	1941	Dumbo	$1,600,000*	$950,000	1600000
4	1942	Bambi	267, 447, 150(3,449,353)R	$1,700,000–2,000,000	270896503
5	1943	Victory Through Air Power	$799,000*	~$789,000	799000
6	1944	The Three Caballeros	$3,355,000R	TBD	3355000
7	1945	The Enchanted Sword	ES€5,595,283ES (~$90,000)	TBD	5685283
8	1946	Make Mine Music	$3,275,000R	$1,370,000	3275000
9	1947	Fun and Fancy Free	$3,165,000R	TBD	3165000
10	1948	Melody Time	$2,560,000R	$1,500,000	2560000
11	1949	The Adventures of Ichabod and Mr. Toad	$1,625,000R	TBD	1625000
12	1950	Cinderella	263, 591, 415(20,000,000/$7,800,000)*R	$2,200,000	263591415
13	1951	Alice in Wonderland	$2,400,000*	$3,000,000	2400000
14	1953	Peter Pan	145, 000, 000(7,000,000)	$3,000,000–4,000,000	152000000
15	1955	Lady and the Tramp	187, 000, 000(6,500,000)*R	$4,000,000	193500000
16	1959	Sleeping Beauty	51, 600, 000∗(5,300,000)R	$6,000,000	56900000

Load:

> To create the required database tables we ran the following scripts on the SQL database.

```
CREATE TABLE `dis_characters` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `movie_title` text,
 `release_date` text,
 `hero` text,
 `villain` text,
 `song` text,
 PRIMARY KEY (`id`)
);

CREATE TABLE `dis_voices` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `character` text,
 `voice_actor` text,
 `movie` text,
 PRIMARY KEY (`id`)
) ;
```

```
CREATE TABLE `disneylist` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `TITLE` text,
 `YEAR` text,
 PRIMARY KEY (`id`)
) ;


CREATE TABLE `imdbdisney` (
  id int NOT NULL AUTO_INCREMENT,
 `Title` text,
 `year` text,
 `Metascore` int(11) DEFAULT NULL,
 `IMDB` double DEFAULT NULL,
 `link` text,
  PRIMARY KEY (`id`)
) ;


CREATE TABLE `disneygrosssales` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `Year` int(11) NOT NULL,
 `title` varchar(100) DEFAULT NULL,
 `Worldwide gross` varchar(100) ,
 `Budget` varchar(20) DEFAULT NULL,
 `Total Sales` int(11) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ;


CREATE TABLE `wikidisney` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `Type of Film` text,
 `Title` text,
 `US Release` text,
 PRIMARY KEY (`id`)
) ;
```

We loaded our clean data scraped from the web into our database tables using to_sql from the pymysql library as below:

```
engine = create_engine("mysql://root:[redacted]@localhost/disney")
conn = engine.connect()


engine.table_names()
```

```
# Send the Wikipedia Disney data to SQL
wikidiz_df.to_sql(name='wikidisney', con=engine, if_exists='append', index=False)
```

```
# Send the Disney List data to SQL
listdiz_df.to_sql(name='disneylist', con=engine, if_exists='append', index=False)
```

```
# Send the IMDB Disney data to SQL
imdbdiz_df.to_sql(name='imdbdisney', con=engine, if_exists='append', index=False)
```

```
# Send the Gross Movie Sales data to SQL
grossdiz_df.to_sql(name='disneygrosssales', con=engine, if_exists='append', index=False)
```

Results and Analysis

Analysis 1: Gross vs. Ratings

> Merged the data from IMDB and Wikipedia to create a SQL view of the Top 10 highly grossed movies along with their IMDB scores and gross sales details

```
CREATE VIEW TOP_10_GROSSERS AS
select a.Title,a.year,Metascore,IMDB,link,`Worldwide gross`,Budget,`Total Sales` from imdbdisney a
join disneygrosssales b
on a.Title=b.title
Order by b.`Total Sales` DESC
Limit 10;

select * from TOP_10_GROSSERS;
```

Results:

The movies that performed well at the box-office aligned with the assumption of having a good/decent IMDB and metascore rating.

	IMDB	Total_Sales
0	8.3	$1,802,482,472.00
1	8.5	$1,735,447,909.00
2	8.0	$1,412,761,314.00
3	7.3	$1,294,557,007.00
4	7.6	$1,149,615,809.00

Analysis 2: Vocal Actors Participation

> Queried the database to understand the involvement of the vocal actors on a movie.

```
actor_voice_movie  = pd.read_sql('select voice_actor, count(distinct movie) as count_of_movies, \
        count(distinct `character`) as count_of_characters \
        from dis_voices group by voice_actor order by count(distinct movie) desc', conn)
actor_voice_movie.head(10)
```

Results:

When analyzing the involvement of voice actors per movie, each voice actor generally played multiple roles in a movie. As an example Frank Welker who played 24 characters on 18 movies.

```
actor_voice_new = pd.read_sql('select voice_actor,movie,`character` \
        from dis_voices where voice_actor = "Frank Welker" order by movie', conn)
actor_voice_new.head(5)
```

	voice_actor	count_of_movies	count_of_characters
0	None	34	53
1	Frank Welker	18	24
2	Jim Cummings	13	17
3	Jeff Bennett	8	8
4	Bill Thompson	6	7
5	Corey Burton	6	8
6	Pinto Colvig	6	7
7	Rob Paulsen	6	6
8	David Ogden Stiers	5	6
9	J. Pat O'Malley	5	8

Analysis 3: IMDB vs Metascore ratings

> Standardized the IMDB and Metascore Ratings to understand the differences that exists between each other

```
standardized_ratings = imdbdiz_df[['Metascore', 'IMDB']].rank(pct=True)
standardized_ratings.mean()
```

Results:

Metascore and IMDB has similar means of ratings. As a conclusion, there is no difference in between the ratings given by one of the other.

```
Metascore    0.505682
IMDB         0.505000
dtype: float64
```

Analysis 4: Movie Performance in the last 10 years

> Queried the database to analyse the performance of movies in the last 10 years:

```
select Title,Budget,Year,`Total Sales`
from disneygrosssales
where Year between 2010 and 2019
group by year
order by `Total Sales` desc ;
```

Challenges and Limitations

- **Data Cleaning**
  To clean the data it was difficult to apply a single pattern match or regex to the entire column because of the complex nature of the scraped raw data. Hence we had to apply multiple pattern match conditions and do some manual editing to extract a clean column of data.

- **Encoding for special characters varies from operating systems**
  As shown above in the Transform & Load Template section, we had different outcomes with encoding. One Windows user had no issue whatsoever, while the other Windows user experienced major encoding issues with two of the CSV's. The MAC user also had no issues with encoding.

- **SQL engine connection can vary from different sql servers**
  Similarly, we had SQL engine connection challenges resulting in an uninstall/re-install of SQL to remedy the issue.
  These challenges impacted the Transform and Load process from Python to SQL, and also provided a lesson learned in expectation management. Regardless of scope, this process requires ample time, patience, communication, and persistence.

ETL Benefits

- Bulk loading of large datasets.
- Creates an easily adaptable and scalable data load template.

How might ETL change when moving to the cloud?

- When the ETL is moved to a cloud server there is a possibility that multiple users could be trying to load data into the same table and this could lead to concurrency issues. It also has advantages where everyone can easily and quickly access the newly loaded data leading to better data governance.