In [1]:

```
#  This cell obtains United States Consumer Price Index (CPI) data from an online server.
#  A function (defined in the present cell) that incorporates CPI data
#  will later be used to make inflation adjustments on movie-related sales and budget figures.
#
#  MICHAEL COLLINS, 2020-09-15_0729_MDT

import pandas as pd
import numpy as np
from IPython.display import display

#  Obtain monthly United States consumer price indices from "inflationdata dot com"
#  and store them in pandas dataframe cpi_df.
source_url = 'https://inflationdata.com/Inflation/Consumer_Price_Index/HistoricalCPI.aspx?reloaded=true'
cpi_df = pd.read_html(source_url)[0].drop(columns='Ave.')
cpi_df.set_index('Year', drop=True, inplace=True, verify_integrity=False)

year_to_cpi = np.round(cpi_df.mean(axis=1, skipna=True), decimals=6)

def inflation_multiplier(from_year, into_year):
    cpi_ratio = None
    bomb = False
    if not bomb:
        from_cpi = None
        try:
            from_cpi = year_to_cpi[from_year]
        except:
            pass
        if not isinstance(from_cpi, float):
            bomb = True
    if not bomb:
        into_cpi = None
        try:
            into_cpi = year_to_cpi[into_year]
        except:
            pass
        if not isinstance(into_cpi, float):
            bomb = True
    if not bomb:
        cpi_ratio = np.round((into_cpi/from_cpi), decimals=6)
    return cpi_ratio

check_inflation_multiplier = True
if not check_inflation_multiplier:
    print("SKIPPING quality checks of inflation_multiplier function...")
    print()
else:
    print("PERFORMING quality checks of inflation_multiplier function...")
    print()

    print("Annual values of the United States Consumer Price Index are")
    print("contained in the pandas series 'year_to_cpi', as follows:")
    display(year_to_cpi)

    print("Here are the inflation adjustment factors one would apply to convert ")
    print("the value of [Year YYYY dollars] into the value of [Year 2020 dollars]:")
    print()
    from_years = list(range(1900,2031))
    for j in [2020]:
        for i in from_years:
            factor_ij = inflation_multiplier(from_year=i, into_year=j)
            print("inflation_multiplier(" + str(i) + ", " + str(j) + ") = " + repr(factor_ij))
        print()
    print()
```

```
PERFORMING quality checks of inflation_multiplier function...

Annual values of the United States Consumer Price Index are
contained in the pandas series 'year_to_cpi', as follows:
```

```
Year
2020    258.045375
2019    255.657417
2018    251.106833
2017    245.119583
2016    240.007167
           ...
1917     12.825000
1916     10.883333
1915     10.108333
1914     10.016667
```

In [2]:

```
#  This cell obtains film industry data directly from the website "the-numbers.com"
#  and stores it in a dictionary called "movieHandle_to_movieDoss".
#
#  A cleaned-up version of the above data is stored in another dictionary called
#  "eventHandle_to_eventDoss".  That dictionary forms the basis of a pandas dataframe
#  in a subsequent cell of this Notebook.
#
#  MICHAEL COLLINS, 2020-09-11_2133_MDT

import datetime
from datetime import date
import requests
from bs4 import BeautifulSoup
import random
import string
from dateutil.parser import parse as dateparse
import os
import pathlib
import numpy as np
import pandas as pd
import winsound
from IPython.display import display
import matplotlib.pyplot as plt
import seaborn as sns

# Note: TN is an abbreviation for "The Numbers"; shorthand for 'the-numbers dot com'
# Note: BOY is an abbreviation for "Box-Office Year"
# Note: bgt is an abbreviation for the "budget" table
# Note: tgy is an abbreviation for the "top-grossing yearly" table


# Constants related to processing of generic calendar dates
YEAR_PLAUSIBLE_FIRST = 1900
YEAR_PLAUSIBLE_LAST = 2100
DATE_PLAUSIBLE_FIRST = datetime.date(YEAR_PLAUSIBLE_FIRST,1,1)
DATE_PLAUSIBLE_LAST = datetime.date(YEAR_PLAUSIBLE_LAST,12,31)
WEEKDAY_MONDAY = 0
WEEKDAY_TUESDAY = 1
WEEKDAY_WEDNESDAY = 2
WEEKDAY_THURSDAY = 3
WEEKDAY_FRIDAY = 4
WEEKDAY_SATURDAY = 5
WEEKDAY_SUNDAY = 6

# Constants related to "the-numbers.com" website
TN_SALES_OMIT = [",", "$"]
TN_SEATS_OMIT = [","]
TN_MAIN_URL = "https://the-numbers.com"
TN_PARENT_FOLDER = "./dayduh"
TN_MAIN_FOLDER = "./dayduh/the-numbers"
TN_TOPGROSS_SUBFOLDER = "/top-gross"
TN_TOPGROSS_SUBURL = "/market/"
TN_TOPGROSS_URL_SUFFIX = "/top-grossing-movies"
TN_BUDGETS_SUBFOLDER = "/budgets"
TN_BUDGETS_SUBURL = "/movie/budgets/"
TN_MOVIE_SUBFOLDER = "/movie"
TN_MOVIE_SUBURL = "/movie/"
TN_MOVIE_URL_SUFFIX = "#tab=summary"
TN_DISTRIB_SUBURL = "/market/distributor/"
TN_GENRE_SUBURL = "/market/genre/"
TN_YEAR_CONSIDER_FIRST = 1977
TN_YEAR_CONSIDER_LAST = 2020
TN_GENRE_HANDLES = ["_unknown_genre_", "Action", "Adventure", "Black-Comedy", "Comedy", "Concert-or-Performance",
                    "Documentary", "Drama", "Horror", "Multiple-Genres", "Musical", "Reality",
                    "Romantic-Comedy", "Thriller-or-Suspense", "Western"]
TN_GENRE_ABBRS = ["(--none--)", "ACTION", "ADVENTURE", "BLK-COMEDY", "COMEDY", "CONCERT",
                  "DOCUMENTARY", "DRAMA", "HORROR", "MULTI-GENRE", "MUSICAL", "REALITY",
                  "ROM-COMEDY", "THRILLER", "WESTERN"]

# dictionaries that relate genre handles to genre abbreviations and vice versa
TN_GENRE_INDICES = list(range(len(TN_GENRE_ABBRS)))
gHandle_to_gAbbr = dict(zip(TN_GENRE_HANDLES, TN_GENRE_ABBRS))
gAbbr_to_gHandle = dict(zip(TN_GENRE_ABBRS, TN_GENRE_HANDLES))
gAbbr_to_gIndex = dict(zip(TN_GENRE_ABBRS, TN_GENRE_INDICES))
gIndex_to_gAbbr = dict(zip(TN_GENRE_INDICES, TN_GENRE_ABBRS))
```

```
76
77  # functions that allow commentary to be printed,
78  # depending on value of COMMENTARY_LEVEL
79  comm1 = lambda s: print(s) if COMMENTARY_LEVEL >= 1 else None
80  comm2 = lambda s: print(s) if COMMENTARY_LEVEL >= 2 else None
81  comm3 = lambda s: print(s) if COMMENTARY_LEVEL >= 3 else None
82  comm4 = lambda s: print(s) if COMMENTARY_LEVEL >= 4 else None
83  COMMENTARY_LEVEL = 3
84
85  # This function returns the first calendar day of a given Box Office Year
86  def boxOfficeYear_firstDay(boxOfficeYear):
87      # The first day of Box Office Year YYYY is
88      #     [the day AFTER the first Sunday in Calendar Year YYYY]
89      d = None
90      bomb = False
91      bomb = bomb or (not isinstance(boxOfficeYear, int))
92      bomb = bomb or (not boxOfficeYear >= YEAR_PLAUSIBLE_FIRST)
93      bomb = bomb or (not boxOfficeYear < YEAR_PLAUSIBLE_LAST)
94      if not bomb:
95          d = datetime.date(boxOfficeYear, 1, 1)
96          while not d.weekday() == WEEKDAY_SUNDAY:
97              d += datetime.timedelta(days=1)
98          while not d.weekday() == WEEKDAY_MONDAY:
99              d += datetime.timedelta(days=1)
100     return d
101
102 # This function returns the last calendar day of a given Box Office Year
103 def boxOfficeYear_lastDay(boxOfficeYear):
104     # The last day of Box Office Year YYYY is
105     #     [the first Sunday in Calendar Year (YYYY + 1)]
106     d = None
107     bomb = False
108     bomb = bomb or (not isinstance(boxOfficeYear, int))
109     bomb = bomb or (not boxOfficeYear >= YEAR_PLAUSIBLE_FIRST)
110     bomb = bomb or (not boxOfficeYear < YEAR_PLAUSIBLE_LAST)
111     if not bomb:
112         d = datetime.date(boxOfficeYear + 1, 1, 1)
113         while not d.weekday() == WEEKDAY_SUNDAY:
114             d += datetime.timedelta(days=1)
115     return d
116
117 # This function returns the Box Office Year associated with a given calendar day
118 def date_to_boxOfficeYear(d):
119     # The last day of Box Office Year YYYY is
120     #     [the first Sunday in Calendar Year (YYYY + 1)]
121     boxOfficeYear = None
122     bomb = False
123     bomb = bomb or (not isinstance(d, datetime.date))
124     bomb = bomb or (not d >= DATE_PLAUSIBLE_FIRST)
125     bomb = bomb or (not d <= DATE_PLAUSIBLE_LAST)
126     if not bomb:
127         d_calendar_year = d.year
128         mainBOY_firstDay = boxOfficeYear_firstDay(d_calendar_year)
129         bomb = not isinstance(mainBOY_firstDay, datetime.date)
130     if not bomb:
131         if d < mainBOY_firstDay:
132             boxOfficeYear = d_calendar_year - 1
133         else:
134             boxOfficeYear = d_calendar_year
135     return boxOfficeYear
136
137
138 # This function makes two audible beeps.
139 # An audible signal, suitably implemented, can alert the user
140 # when (for example) a long-running computational task is completed.
141 def make_beeps():
142     Freq = 440 # Set Frequency To 440 Hertz
143     Dur = 500 # Set Duration To 500 ms == 0.5 second
144     winsound.Beep(Freq,Dur)
145     Freq = 880 # Set Frequency To 880 Hertz
146     Dur = 500 # Set Duration To 500 ms == 0.5 second
147     winsound.Beep(Freq,Dur)
148     return
149
150 # extract movie handle from movie href
151 def get_movie_handle(href):
```

```
    def remove_mh_prefix(v):
        return v[len(TN_MOVIE_SUBURL):] if v.startswith(TN_MOVIE_SUBURL) else v
    def remove_mh_suffix(v):
        return v[:-len(TN_MOVIE_URL_SUFFIX)] if v.endswith(TN_MOVIE_URL_SUFFIX) else v
    return remove_mh_suffix(remove_mh_prefix(href))

# extract distributor handle from distributor href
def get_distrib_handle(v):
    return v[len(TN_DISTRIB_SUBURL):] if v.startswith(TN_DISTRIB_SUBURL) else v

# extract genre handle from genre href
def get_genre_handle(v):
    return v[len(TN_GENRE_SUBURL):] if v.startswith(TN_GENRE_SUBURL) else v

def tnURL_movieWebsite(movie_handle, as_path=False):
    bomb = False
    bomb = bomb or (not isinstance(movie_handle, str))
    bomb = bomb or (not len(movie_handle) >= 1)
    f_result = None
    if not bomb:
        path_ref = "/" + movie_handle + ".html"
        url_ref = movie_handle + TN_MOVIE_URL_SUFFIX
        if as_path:
            f_result = TN_MAIN_FOLDER + TN_MOVIE_SUBFOLDER + path_ref
        else:
            f_result = TN_MAIN_URL + TN_MOVIE_SUBURL + url_ref
    return f_result

def tnURL_budgets_glob(start_rank=1, as_path=False):
    bomb = False
    bomb = bomb or (not isinstance(start_rank, int))
    bomb = bomb or (not start_rank >= 1)
    f_result = None
    if not bomb:
        page, j = divmod(start_rank, 100)
        if page > 0:
            i_rank = (100 * page) + 1
            s_rank = str(i_rank).zfill(4)
            path_ref = "/all_" + s_rank + ".html"
            url_ref = "all/" + s_rank
        else:
            path_ref = "/all_0001.html"
            url_ref = "all/1"
        if as_path:
            f_result = TN_MAIN_FOLDER + TN_BUDGETS_SUBFOLDER + path_ref
        else:
            f_result = TN_MAIN_URL + TN_BUDGETS_SUBURL + url_ref
    return f_result

def tnURL_topGross_byYear(box_office_year, as_path=False):
    bomb = False
    bomb = bomb or (not isinstance(box_office_year, int))
    bomb = bomb or (not box_office_year >= TN_YEAR_CONSIDER_FIRST)
    bomb = bomb or (not box_office_year <= TN_YEAR_CONSIDER_LAST)
    f_result = None
    if not bomb:
        path_ref = "/top-grossing-movies_" + str(box_office_year) + ".html"
        url_ref = str(box_office_year) + "/top-grossing-movies"
        if as_path:
            f_result = TN_MAIN_FOLDER + TN_TOPGROSS_SUBFOLDER + path_ref
        else:
            f_result = TN_MAIN_URL + TN_TOPGROSS_SUBURL + url_ref
    return f_result


def local_curated_folder_exists(desired_folder):
    result = False

    still_going = True
    while still_going:
        # Check whether a locally-curated version of the desired webContent already exists.
        try:
            old_folder = pathlib.Path(desired_folder)
            old_folder_exists = old_folder.exists()
        except:
            print("Issue: could not determine whether local folder {" + desired_folder + "} already exists.")
```

```
            still_going = False
            break
        if old_folder_exists:
            # This is the most-preferred outcome.  The desired folder already exists.
            still_going = False
            result = True
            break

        # At this point, the desired local folder does NOT already exist.

        # Try to create the desired local folder.
        try:
            new_folder = pathlib.Path(desired_folder).mkdir(mode=0o777, parents=True, exist_ok=False)
        except:
            print("Issue: could not create new local folder {" + desired_folder + "}.")
            still_going = False
            break

        # At this point, we THINK a new instance of the desired local folder was just created.
        # Make sure it is actually there.
        try:
            new_folder = pathlib.Path(desired_folder)
            new_folder_exists = new_folder.exists()
        except:
            print("Issue: could not determine whether **newly-created** local folder {" + desired_folder + "} already exists.")
            still_going = False
            break
        if new_folder_exists:
            # This is the second-most-preferred outcome.  The desired folder was created.
            still_going = False
            result = True
            break

    return result


def local_curated_file_exists(webContent_local_path, webContent_url):
    result = False

    still_going = True
    while still_going:

        # Check whether a locally-curated version of the desired webContent already exists.
        try:
            old_file = pathlib.Path(webContent_local_path)
            old_file_exists = old_file.exists()
        except:
            print("Issue: could not determine whether local file {" + webContent_local_path + "} already exists.")
            still_going = False
            break
        if old_file_exists:
            # This is the most favorable outcome.  A locally curated version of the webContent was found.
            still_going = False
            result = True
            break

        # Download the webContent via webContent_url
        try:
            r = requests.get(webContent_url)
            webContent = r.content
        except:
            print("Issue: webContent was not obtained from URL {" + webContent_url + "}.")
            still_going = False
            break

        # Save the webContent as a locally-curated file.
        try:
            with open(webContent_local_path, 'wb') as f:
                f.write(webContent)
        except:
            print("Issue: webContent was not written to file {" + webContent_local_path + "}.")
            still_going = False
            break

        # Confirm that the (brand-new) locally-curated file exists
        new_file_exists = False
```

```
        try:
            new_file = pathlib.Path(webContent_local_path)
            new_file_exists = new_file.exists()
        except:
            print("Issue: after webContent was stored locally, could not determine " +
                  "whether the local copy {" + webContent_local_path + "} exists.")
            still_going = False
            break
        if not new_file_exists:
            print("Issue: after webContent was stored locally, the local copy {" +
                  webContent_local_path + "} does not exist.")
            still_going = False
            break

        # Confirm that the (brand-new) locally-curated file can be read
        try:
            with open(webContent_local_path, 'rb') as g:
                localContent = g.read()
        except:
            print("Issue: after webContent was stored locally, the local copy {" +
                  webContent_local_path + "} exists, but could not be read.")
            still_going = False
            break

        # Confirm that the archived version of the webContent
        # is exactly the same as the downloaded version of the webContent
        if not localContent == webContent:
            print("Issue: after webContent was stored locally, the local version " +
                  "found at {" + webContent_local_path + "} is NOT an exact replica of the " +
                  "webContent that was downloaded from {" + webContent_url + "}.")
            still_going = False
            break

        # The archived webContent is an exact replica of the downloaded webContent
        still_going = False
        result = True
        print("An exact replica of the webContent at {" + webContent_url +
              "} was stored in local file {" + webContent_local_path + "}.")
        print()

    return result




# ============================================================================
#   IN THIS SECTION,
#       QUALITY CHECKS ON THE FUNCTIONS DEFINED ABOVE
#       MAY OPTIONALLY BE PERFORMED
#       AT THE DISCRETION OF THE USER
#       BY SETTING THE "check_????" FLAGS TO True or False, ACCORDINGLY
# ============================================================================

check_genre_dicts = True
if not check_genre_dicts:
    print("SKIPPING quality checks of genre-related dictionaries...")
    print()
else:
    print("PERFORMING quality checks of genre-related dictionaries...")
    print()

    print("Quality check of gHandle_to_gAbbr...")
    for i, k in enumerate(TN_GENRE_HANDLES):
        def check_same(v1, v2):
            return "PASS" if (v1 == v2) else "FAIL"
        v_expected = TN_GENRE_ABBRS[i]
        v_lookup = "--absent--"
        try:
            v_lookup = gHandle_to_gAbbr[k]
        except:
            v_lookup = None
        print("i=" + str(i) + ", key=" + repr(k) + ", v_expected=" + repr(v_expected) +
              ", v_lookup=" + repr(v_lookup) + ", status=" + check_same(v_expected, v_lookup))
    print()

    print("Quality check of gAbbr_to_gHandle...")
```

```
    for i, k in enumerate(TN_GENRE_ABBRS):
        def check_same(v1, v2):
            return "PASS" if (v1 == v2) else "FAIL"
        v_expected = TN_GENRE_HANDLES[i]
        v_lookup = "--absent--"
        try:
            v_lookup = gAbbr_to_gHandle[k]
        except:
            v_lookup = None
        print("i=" + str(i) + ", key=" + repr(k) + ", v_expected=" + repr(v_expected) +
              ", v_lookup=" + repr(v_lookup) + ", status=" + check_same(v_expected, v_lookup))
    print()

check_boxOfficeYear_date_functions = True
if not check_boxOfficeYear_date_functions:
    print("SKIPPING quality checks of boxOfficeYear-related date functions...")
    print()
else:
    print("PERFORMING quality checks of boxOfficeYear-related date functions...")
    print()

    print("Calculating the beginning and ending dates of various Box Office Years...")
    print()
    BOYs = [y for y in range(TN_YEAR_CONSIDER_FIRST, TN_YEAR_CONSIDER_LAST + 1)]
    for BOY in BOYs:
        day_first = boxOfficeYear_firstDay(BOY)
        day_last = boxOfficeYear_lastDay(BOY)

        print("BOY = {" + str(BOY) + "}, starts " +
              day_first.strftime("%a") + " {" +
              repr(day_first) + "}, ends " +
              day_last.strftime("%a") + " {" +
              repr(day_last) + "}")
    print()

    print("Verifying that the calculated boxOfficeYear for each individual day")
    print("that occurs DURING that particular boxOfficeYear is in fact the")
    print("known (prescribed) boxOfficeYear...")
    print()
    BOYs = [y for y in range(1915,2032)]
    for BOY in BOYs:
        day_first = boxOfficeYear_firstDay(BOY)
        day_last = boxOfficeYear_lastDay(BOY)
        d = day_first
        BOY_num_days_agree = 0
        BOY_num_days_disagree = 0
        BOY_list_days_disagree = []
        while d <= day_last:
            calc_BOY = date_to_boxOfficeYear(d)
            if calc_BOY == BOY:
                BOY_num_days_agree += 1
            else:
                BOY_num_days_disagree += 1
                BOY_list_days_disagree.append(d)

            d += datetime.timedelta(days=1)
        if BOY_num_days_disagree > 0:
            print("BOX OFFICE YEAR " + str(BOY) + ":")
            print("     " + str(BOY_num_days_disagree) + " days DISAGREE, as follow:")
            for d in BOY_list_days_disagree:
                print("     " + repr(d))
    print()

check_tnURL_budgets_glob = False
if not check_tnURL_budgets_glob:
    print("SKIPPING quality checks of function tnURL_budgets_glob...")
    print()
else:
    print("PERFORMING quality checks of function tnURL_budgets_glob...")
    print()
    start_nums = sorted([random.randint(-100, 3001) for k in range(100)])
    for i in start_nums:
        print("i = {" + str(i) + "}: ")
        print("     budgets_glob_path = " + repr(tnURL_budgets_glob(i, as_path=True)))
        print("     budgets_glob_url  = " + repr(tnURL_budgets_glob(i)))
    print()
```

```
456
457 check_tnURL_topGross_byYear = False
458 if not check_tnURL_topGross_byYear:
459     print("SKIPPING quality checks of function tnURL_topGross_byYear...")
460     print()
461 else:
462     print("PERFORMING quality checks of function tnURL_topGross_byYear...")
463     print()
464     box_office_years = [iBOY for iBOY in range(1960,2031)]
465     for iBOY in box_office_years:
466         print("boxOfficeYear = {" + str(iBOY) + "}: ")
467         print("     topGross_byYear_path = " + repr(tnURL_topGross_byYear(iBOY, as_path=True)))
468         print("     topGross_byYear_url  = " + repr(tnURL_topGross_byYear(iBOY)))
469     print()
470
471 check_tnURL_movieWebsite = False
472 if not check_tnURL_movieWebsite:
473     print("SKIPPING quality checks of function tnURL_movieWebsite...")
474     print()
475 else:
476     print("PERFORMING quality checks of function tnURL_movieWebsite...")
477     print()
478     # Here are some example movie handles.  This is for testing the
479     # operation of the tnURL_movieWebsite function
480     movie_handles = ['(Untitled)', '10', '10-000-B-C', '10-Cloverfield-Lane', '10-Days-in-a-Madhouse',
481                      '10-Questions-for-the-Dalai-Lama', '10-Things-I-Hate-About-You', '10-to-Midnight',
482                      '10-Years', '100-Acres-of-Hell', '100-Arabica', '100-Bloody-Acres', '1001-Grams',
483                      '101-Dalmatians-(1961)', '101-Dalmatians-(1996)', '101-ReykjavA-k', '102-Dalmatians',
484                      '102-Not-Out-(India)', '10E', '10th-and-Wolf', '11-09-01-September-11', '11-11-11',
485                      '11-14', '11th-Hour', '12-(2009-Russian-Federation)', '12-Angry-Men', '12-in-a-Box',
486                      '12-jours-(France-2017)', '12-Monkeys', '12-O-Clock-Boys', '12-Rounds', '12-Strong',
487                      '12-Years-a-Slave', '120-battements-par-minute-(France)-(BPM-Beats-Per-Minute)',
488                      '127-Hours', '127-Hours-(2010)', '13-Going-On-30',
489                      '13-Hours-The-Secret-Soldiers-of-Benghazi', '13-Months-of-Sunshine', '13-Sins',
490                      '13-Tzameti', '13B', '13th-Warrior-The', '1408', '1492-Conquest-of-Paradise',
491                      '15', '15-17-to-Paris-The', '15-fevrier-1839', '15-Minutes', '16-Blocks',
492                      '16-to-Life', '1612', '17-Again', '17-filles-(France)', '1776', '18-Again',
493                      '18-ans-apres', '180-South', '1898-Los-ultimos-de-Filipinas-(Spain)', '1915',
494                      '1917-(2019)', '1941', '1945-(Hungary)', '1969', '1981', '1982',
495                      '1999-Cannes-Intl-Adv-Festival', '2-13', '2-22-(2017)', '2-automnes-3-hivers',
496                      '2-Days-in-New-York', '2-Days-In-The-Valley', '2-Fast-2-Furious', '2-For-the-Money',
497                      '2-Guns', '2-Manner-2-Frauen-4-Probleme', '2-ou-3-choses-que-je-sais-d-elle',
498                      '2-States', '20-centimetros', '20-Dates', '20-Feet-From-Stardom', '200-Cartas',
499                      '200-Cigarettes', '20000-Days-on-Earth', '20000-Leagues-Under-the-Sea-(1916)',
500                      '20000-Leagues-Under-the-Sea-(1954)', '2001-A-Space-Odyssey',
501                      '2005-Academy-Award-Nominated-Short-Films-The',
502                      '2006-Academy-Award-Nominated-Short-Films', '2009-Oscar-Shorts', '2010',
503                      '2010-Oscar-Shorts', '2011-Oscar-Shorts', '2012', '2012-Oscar-Shorts',
504                      '2012-Time-for-Change', '2013-Oscar-Shorts', '2014-Oscar-Shorts', '2015-Oscar-Shorts',
505                      '2016-Obama-s-America']
506     for mh in movie_handles[0:10]:
507         movieWebsite_path = tnURL_movieWebsite(mh, as_path=True)
508         movieWebsite_url  = tnURL_movieWebsite(mh)
509
510         print("movie_handle = {" + mh + "}: ")
511         print("     movieWebsite_path = " + repr(movieWebsite_path))
512         print("     movieWebsite_url  = " + repr(movieWebsite_url))
513         print()
514
515         # Now store a local version of the movie's web content
516
517         if not local_curated_file_exists(movieWebsite_path, movieWebsite_url):
518             print("Issue: Attempt to store webContent at URL {" +
519                   movieWebsite_url + "} as local file {" +
520                   movieWebsite_path + "} has failed.")
521             print()
522
523
524
525 # ============================================================================
526 #   IN THIS SECTION,
527 #       LOCAL FOLDERS (FOR CACHED LOCAL VERSIONS OF WEB CONTENT) ARE CREATED, AS NEEDED.
528 #
529 # ============================================================================
530
531 issues_found = True
```

```
if local_curated_folder_exists(TN_PARENT_FOLDER):
    if local_curated_folder_exists(TN_MAIN_FOLDER):
        if local_curated_folder_exists(TN_MAIN_FOLDER + TN_MOVIE_SUBFOLDER):
            if local_curated_folder_exists(TN_MAIN_FOLDER + TN_BUDGETS_SUBFOLDER):
                if local_curated_folder_exists(TN_MAIN_FOLDER + TN_TOPGROSS_SUBFOLDER):
                    issues_found = False
                else:
                    print("Issue: Attempt to create local folder {" +
                          TN_MAIN_FOLDER + TN_TOPGROSS_SUBFOLDER + "} has failed.")
            else:
                print("Issue: Attempt to create local folder {" +
                      TN_MAIN_FOLDER + TN_BUDGETS_SUBFOLDER + "} has failed.")
        else:
            print("Issue: Attempt to create local folder {" +
                  TN_MAIN_FOLDER + TN_MOVIE_SUBFOLDER + "} has failed.")
    else:
        print("Issue: Attempt to create local folder {" +
              TN_MAIN_FOLDER + "} has failed.")
else:
    print("Issue: Attempt to create local folder {" +
          TN_PARENT_FOLDER + "} has failed.")
if issues_found:
    # Pitch a fit.
    print("Unable to cache webContent in local data curation folders.")
    print()
    print("Purposely Terminating program execution, now.")
    assert False
print("All desired folders for local curation of webContent were verified to be present.")
print()



# ================================================================================
#   IN THIS SECTION,
#       HARVESTING OF DATA FROM "the-numbers dot com" IS PERFORMED.
#
#       EACH MODE OF DATA HARVEST OPERATION
#       MAY OPTIONALLY BE PERFORMED AT THE DISCRETION OF THE USER
#       BY SETTING "harvest_????" FLAGS TO True OR False, ACCORDINGLY
# ================================================================================

movieHandle_to_movieDoss = dict()
distribHandle_to_distribDoss = dict()
genreHandle_to_genreDoss = dict()

bgt_rows_cumulative = 0
tgy_rows_cumulative = 0


harvest_bgt = True
if not harvest_bgt:
    print("SKIPPING harvest of [movies sorted by production budget]...")
    print()
else:
    print("PERFORMING harvest of [movies sorted by production budget]...")
    print()

    COMMENTARY_LEVEL = 2
    bgt_rank_nextup = 1
    bgt_rank_max = 9999

    comm1("=========================================")
    comm1("   HARVESTING DATA FROM [MULTI-PAGE TABLE")
    comm1("   OF MOVIES RANKED BY PRODUCTION BUDGET]")
    comm1("=========================================")
    comm1("")

    quite_done = False
    issues_found = True
    while not quite_done:

        bgt_glob_path = tnURL_budgets_glob(bgt_rank_nextup, as_path=True)
        bgt_glob_url = tnURL_budgets_glob(bgt_rank_nextup)

        comm4(bgt_glob_path)
        comm4("")
```

```
        comm4(bgt_glob_url)
        comm4("")

        if not local_curated_file_exists(bgt_glob_path, bgt_glob_url):
            print("Issue: Attempt to store webContent at URL {" +
                  bgt_glob_url + "} as local file {" +
                  bgt_glob_path + "} has failed.")
            quite_done = True
            break

        comm3("Getting HTML content from local file {" + bgt_glob_path + "} as soup...")
        comm3("")
        try:
            with open(bgt_glob_path, 'rb') as g:
                localContent = g.read()
                soup = BeautifulSoup(
                    localContent, "html.parser")
        except:
            print("Issue: Attempt retrieve curated webContent from local file {" +
                  bgt_glob_path + "} has failed.")
            quite_done = True
            break

        comm4("Identifying tables in soup...")
        comm4("")

        soup_tables = soup.find_all("table")

        num_soup_tables = len(soup_tables)
        comm4("num_soup_tables = " + repr(num_soup_tables))
        comm4("")

        if num_soup_tables == 0:
            print("ISSUE: No tables were identified in soup.")
            print("Did not obtain tabular data from file {" + bgt_glob_path + "}.")
            quite_done = True
            break

        if num_soup_tables > 1:
            comm4("A total of " + str(num_soup_tables) + " tables were identified in soup.")
            comm4("Only the first table will be examined.")
        table = soup_tables[0]

        # Look for the header row, then extract headers from it, if found.
        comm4("Now processing the HEADER ROW...")
        comm4("")
        contains_th = lambda tag: len(tag.find_all("th")) > 0

        tr_list = [tr for tr in table.find_all("tr", recursive=False) if contains_th(tr)]

        num_header_rows = len(tr_list)
        comm4("num_header_rows = " + repr(num_header_rows))
        comm4("")
        if num_header_rows == 0:
            print("ISSUE: No header row was identified in the table.")
            quite_done = True
            break
        if num_header_rows > 1:
            comm4("A total of " + str(num_header_rows) + " header rows were identified in the table.")
            comm4("Only the first header row will be examined.")
            comm4("")

        original_headers = [" ".join(th.strings) for th in tr_list[0].find_all("th")]

        table_num_columns = len(original_headers)
        comm4("table_num_columns = " + repr(table_num_columns))
        comm4("")
        if table_num_columns == 0:
            print("ISSUE: No column headers were found in the header row.")
            quite_done = True
            break

        comm4("The following column headers were identified:")
        comm4(repr(original_headers))
        comm4("")

```

```
        # correlate original headers with column indices
        colHeader_to_j = dict(zip(original_headers, [j for j in range(table_num_columns)]))
        j_MOVIE = colHeader_to_j['Movie']
        j_RELEASE_DATE = colHeader_to_j['Release Date']
        j_PRODUCTION_BUDGET = colHeader_to_j['Production Budget']
        j_DOMESTIC_GROSS = colHeader_to_j['Domestic Gross']
        j_WORLDWIDE_GROSS = colHeader_to_j['Worldwide Gross']

        comm4("Now processing the DATA ROWS...")
        comm4("")

        contains_a = lambda tag: len(tag.find_all("a")) > 0
        contains_td = lambda tag: len(tag.find_all("td")) > 0
        tr_list = [tr for tr in table.find_all("tr", recursive=False) if contains_a(tr) & contains_td(tr)]

        num_data_rows = len(tr_list)
        comm4("num_data_rows = " + repr(num_data_rows))
        comm4("")
        if num_data_rows == 0:
            print("ISSUE: No data rows were identified in the table.")
            quite_done = True
            break

        for i, tr in enumerate(tr_list):
            td_list = tr.find_all("td")

            num_td = len(td_list)
            if num_td < table_num_columns:
                print("ISSUE: Only "+ str(num_td) + " data elements were identified " +
                      "in the data row whose index is {" + str(i) + "} .")
                print("Each data row in the table was expected to have {" + str(table_num_columns) + "} columns .")
                quite_done = True
                break

            for tda in [td_list[j_MOVIE].a]:
                if tda is None:
                    film_handle = None
                else:
                    film_handle = get_movie_handle(str(tda.get('href')))

            for tda in [td_list[j_RELEASE_DATE].a]:
                found_date = None
                if not tda is None:
                    try:
                        found_string = tda.string
                    except:
                        found_string = "unknown"

                    if found_string.lower() != "unknown":
                        try:
                            found_date = dateparse(found_string).date()
                        except:
                            print("MINOR ISSUE: unable to dateparse this: {" + repr(tda) + "}")

                if found_date is None:
                    print("MINOR_ISSUE: film {" + film_handle + "} is missing a release date in {" + bgt_glob_url + "} glob.")

                bgt_release_date = found_date

            for td in [td_list[j_PRODUCTION_BUDGET]]:
                if td is None:
                    bgt_productionBudget_dollars = -1
                else:
                    bgt_productionBudget_dollars = int("".join([r for r in td.string if not r in TN_SALES_OMIT]))

            for td in [td_list[j_DOMESTIC_GROSS]]:
                if td is None:
                    bgt_domUS_gross_dollars = -1
                else:
                    bgt_domUS_gross_dollars = int("".join([r for r in td.string if not r in TN_SALES_OMIT]))

            for td in [td_list[j_WORLDWIDE_GROSS]]:
                if td is None:
                    bgt_world_gross_dollars = -1
                else:
                    bgt_world_gross_dollars = int("".join([r for r in td.string if not r in TN_SALES_OMIT]))
```

```
760
761
762             bgt_doss = dict()
763             bgt_doss['01_film_handle'] = film_handle
764             bgt_doss['bgt_release_date'] = bgt_release_date
765             bgt_doss['bgt_productionBudget_dollars'] = bgt_productionBudget_dollars
766             bgt_doss['bgt_domUS_gross_dollars'] = bgt_domUS_gross_dollars
767             bgt_doss['bgt_world_gross_dollars'] = bgt_world_gross_dollars
768
769             # Check whether a dossier already exists for this movieHandle.
770             # If a dossier does not already exist, create an empty dossier for it.
771             try:
772                 old_doss = movieHandle_to_movieDoss[film_handle]
773             except:
774                 movieHandle_to_movieDoss[film_handle] = dict()
775             # Assign values for top-level keys
776             # Top-level key-value pairs contain static (non-time-dependent) information about the film
777             for k in bgt_doss.keys():
778                 # Check whether key k is already defined in the dossier
779                 new_val = bgt_doss[k]
780                 try:
781                     old_val = movieHandle_to_movieDoss[film_handle][k]
782                 except:
783                     movieHandle_to_movieDoss[film_handle][k] = new_val
784
785         bgt_rows_cumulative += num_data_rows
786         comm4("All {" + str(num_data_rows)+ "} data rows in the table were processed as expected.")
787         comm4("")
788
789         if num_data_rows < 100:
790             comm2("The last table has been harvested.")
791             comm2("")
792             quite_done = True
793             issues_found = False
794             break
795
796         # This tells us the rank of the first movie in the next glob to be harvested
797         bgt_rank_nextup += 100
798
799         if bgt_rank_nextup > bgt_rank_max:
800             comm2("ISSUE: The desired maximum number of rows {" + str(bgt_rank_max)+ "} has been reached.")
801             comm2("")
802             quite_done = True
803             break
804
805
806     if issues_found:
807         comm4("ISSUES were encountered while processing file {" + bgt_glob_path + "}.")
808         comm4("")
809     else:
810         comm2("Data from file {" + bgt_glob_path + "} was processed without issues.")
811         comm4("A total of {" + str(bgt_rows_cumulative)+ "} data rows were processed, so far.")
812         comm4("")
813
814     comm4("bgt_rows_cumulative = " + repr(bgt_rows_cumulative))
815     comm4("")
816     print("DONE PERFORMING harvest of [movies sorted by production budget].")
817     print()
818
819 harvest_tgy = True
820 if not harvest_tgy:
821     print("SKIPPING harvest of top-grossing movies by year...")
822     print()
823 else:
824     print("PERFORMING harvest of top-grossing movies by year...")
825     print()
826
827     COMMENTARY_LEVEL = 3
828
829     tgy_desired_year_first = 1977
830     tgy_desired_year_last = 2020
831     tgy_desired_years = [i for i in range(tgy_desired_year_first,
832                                           (tgy_desired_year_last + 1))]
833
834     comm1("========================================")
835     comm1("   HARVESTING DATA FROM [ANNUAL TABLES OF")
```

```
    comm1("   OF MOVIES RANKED BY DOMESTIC GROSS]")
    comm1("   for the following box office years:")
    comm1("   " + repr(tgy_desired_years))
    comm1("==========================================")
    comm1("")

    for tgy_desired_year in tgy_desired_years:

        quite_done = False
        issues_found = True
        while not quite_done:

            comm4("============================================================================================")
            comm4("")

            comm3("Importing table of movies ranked by [domestic gross during box office year " + str(tgy_desired_year) + "]...")
            comm4("")

            tgy_table_path = tnURL_topGross_byYear(tgy_desired_year, as_path=True)
            tgy_table_url = tnURL_topGross_byYear(tgy_desired_year)

            comm4(tgy_table_path)
            comm4("")
            comm4(tgy_table_url)
            comm4("")

            if not local_curated_file_exists(tgy_table_path, tgy_table_url):
                print("Issue: Attempt to store webContent at URL {" +
                      tgy_table_url + "} as local file {" +
                      tgy_table_path + "} has failed.")
                quite_done = True
                break

            comm4("Getting HTML content from local file {" + tgy_table_path + "} as soup...")
            comm4("")
            try:
                with open(tgy_table_path, 'rb') as g:
                    localContent = g.read()
                    soup = BeautifulSoup(
                        localContent, "html.parser")
            except:
                print("Issue: Attempt retrieve curated webContent from local file {" +
                      tgy_table_path + "} has failed.")
                quite_done = True
                break

            comm4("Identifying tables in soup...")
            comm4("")

            soup_tables = soup.find_all("table")

            num_soup_tables = len(soup_tables)
            comm4("num_soup_tables = " + repr(num_soup_tables))
            comm4("")

            if num_soup_tables == 0:
                print("ISSUE: No tables were identified in soup.")
                print("Did not obtain tabular data from URL {" + tgy_table_url + "}.")
                quite_done = True
                break

            if num_soup_tables > 1:
                comm4("A total of " + str(num_soup_tables) + " tables were identified in soup.")
                comm4("Only the first table will be examined.")
            table = soup_tables[0]

            # Look for the header row, then extract headers from it, if found.
            comm4("Now processing the HEADER ROW...")
            comm4("")
            contains_th = lambda tag: len(tag.find_all("th")) > 0

            tr_list = [tr for tr in table.find_all("tr", recursive=False) if contains_th(tr)]

            num_header_rows = len(tr_list)
            comm4("num_header_rows = " + repr(num_header_rows))
            comm4("")
```

```
            if num_header_rows == 0:
                print("ISSUE: No header row was identified in the table.")
                quite_done = True
                break
            if num_header_rows > 1:
                comm4("A total of " + str(num_header_rows) + " header rows were identified in the table.")
                comm4("Only the first header row will be examined.")
                comm4("")

            original_headers = [" ".join(th.strings) for th in tr_list[0].find_all("th")]

            table_num_columns = len(original_headers)
            comm4("table_num_columns = " + repr(table_num_columns))
            comm4("")
            if table_num_columns == 0:
                print("ISSUE: No column headers were found in the header row.")
                quite_done = True
                break

            comm4("The following column headers were identified:")
            comm4(repr(original_headers))
            comm4("")

            # correlate original headers with column indices
            colHeader_to_j = dict(zip(original_headers, [j for j in range(table_num_columns)]))
            j_YYYY_DOMUS_GROSS_RANK = colHeader_to_j['Rank']
            j_MOVIE = colHeader_to_j['Movie']
            j_RELEASE_DATE = colHeader_to_j['Release Date']
            j_DISTRIBUTOR = colHeader_to_j['Distributor']
            j_GENRE = colHeader_to_j['Genre']
            j_YYYY_DOMUS_GROSS = colHeader_to_j[str(tgy_desired_year) + ' Gross']
            j_YYYY_DOMUS_TICKETS = colHeader_to_j['Tickets Sold']

            table_BOX_OFFICE_YEAR = original_headers[j_YYYY_DOMUS_GROSS][0:4]
            comm4("table_BOX_OFFICE_YEAR = " + table_BOX_OFFICE_YEAR)
            comm4("")

            # Make sure the box office year embedded in the column header is the
            # same as the desired year
            if table_BOX_OFFICE_YEAR != str(tgy_desired_year):
                print("ISSUE: A column header mentions year {" + table_BOX_OFFICE_YEAR +
                      "}, whereas {" + str(tgy_desired_year) + "} is the desired year.")
                quite_done = True
                break

            comm4("Now processing the DATA ROWS...")
            comm4("")

            contains_a = lambda tag: len(tag.find_all("a")) > 0
            contains_td = lambda tag: len(tag.find_all("td")) > 0
            tr_list = [tr for tr in table.find_all("tr", recursive=False) if contains_a(tr) & contains_td(tr)]

            num_data_rows = len(tr_list)
            comm4("num_data_rows = " + repr(num_data_rows))
            comm4("")
            if num_data_rows == 0:
                print("ISSUE: No data rows were identified in the table.")
                quite_done = True
                break

            for i, tr in enumerate(tr_list):
                td_list = tr.find_all("td")

                num_td = len(td_list)
                if num_td < table_num_columns:
                    print("ISSUE: Only "+ str(num_td) + " data elements were identified " +
                          "in the data row whose index is {" + str(i) + "} .")
                    print("Each data row in the table was expected to have {" + str(table_num_columns) + "} columns .")
                    quite_done = True
                    break

                tgy_YYYY = table_BOX_OFFICE_YEAR

                for td in [td_list[j_YYYY_DOMUS_GROSS_RANK]]:
                    if td is None:
                        tgy_domUSsales_dollars_rank_boxOfficeYear = None
```

```
                    else:
                        tgy_domUSsales_dollars_rank_boxOfficeYear = int(td.string)

                for tda in [td_list[j_MOVIE].a]:
                    if tda is None:
                        film_handle = None
                        tgy_film_href = None
                        tgy_film_title = None
                        tgy_film_url = None
                    else:
                        film_handle = get_movie_handle(str(tda.get('href')))
                        tgy_film_href = str(tda.get('href'))
                        tgy_film_title = tda.string
                        tgy_film_url = TN_MAIN_URL + str(tda.get('href'))

                for tda in [td_list[j_RELEASE_DATE].a]:

                    found_date = None
                    if not tda is None:
                        try:
                            found_string = tda.string
                        except:
                            found_string = "unknown"

                        if found_string.lower() != "unknown":
                            try:
                                found_date = dateparse(found_string).date()
                            except:
                                print("MINOR_ISSUE: unable to dateparse this: {" + repr(tda) + "}.")

                    if found_date is None:
                        comm2("MINOR_ISSUE: film {" + film_handle +
                              "} is missing a release date in the {" +
                              tgy_YYYY + "_topGross} table.")

                    tgy_release_date = found_date

                    if tda is None:
                        tgy_release_href = None
                        tgy_release_url = None
                    else:
                        tgy_release_href = str(tda.get('href'))
                        tgy_release_url = TN_MAIN_URL + str(tda.get('href'))

                for tda in [td_list[j_DISTRIBUTOR].a]:
                    if tda is None:
                        tgy_distrib_href = None
                        tgy_distrib_handle = None
                        tgy_distrib_name = None
                        tgy_distrib_url = None
                    else:
                        tgy_distrib_href = str(tda.get('href'))
                        tgy_distrib_handle = get_distrib_handle(str(tda.get('href')))
                        tgy_distrib_name = tda.string
                        tgy_distrib_url = TN_MAIN_URL + str(tda.get('href'))

                for tda in [td_list[j_GENRE].a]:
                    if tda is None:
                        tgy_genre_href = None
                        tgy_genre_handle = TN_GENRE_HANDLES[0]
                        tgy_genre_abbr = TN_GENRE_ABBRS[0]
                        tgy_genre_name = None
                        tgy_genre_url = None
                    else:
                        tgy_genre_href = str(tda.get('href'))
                        tgy_genre_handle = get_genre_handle(str(tda.get('href')))
                        tgy_genre_abbr = gHandle_to_gAbbr[get_genre_handle(str(tda.get('href')))]
                        tgy_genre_name = tda.string
                        tgy_genre_url = TN_MAIN_URL + str(tda.get('href'))

                for td in [td_list[j_YYYY_DOMUS_GROSS]]:
                    if td is None:
                        tgy_domUSsales_dollars_boxOfficeYear = -1
                    else:
                        tgy_domUSsales_dollars_boxOfficeYear = int("".join([r for r in td.string if not r in TN_SALES_OMIT]))

```

```
                for td in [td_list[j_YYYY_DOMUS_TICKETS]]:
                    if td is None:
                        tgy_domUSsales_tickets_boxOfficeYear = -1
                    else:
                        tgy_domUSsales_tickets_boxOfficeYear = int("".join([r for r in td.string if not r in TN_SEATS_OMIT]))

                if COMMENTARY_LEVEL >= 4:
                    print("01_film_handle = " + repr(film_handle))
                    print("tgy_film_title = " + repr(tgy_film_title))
                    print("tgy_film_url = " + repr(tgy_film_url))
                    print("tgy_release_date = " + repr(tgy_release_date))
                    print("tgy_release_url = " + repr(tgy_release_url))
                    print("tgy_distrib_handle = " + repr(tgy_distrib_handle))
                    print("tgy_distrib_name = " + repr(tgy_distrib_name))
                    print("tgy_distrib_url = " + repr(tgy_distrib_url))
                    print("tgy_genre_handle = " + repr(tgy_genre_handle))
                    print("tgy_genre_abbr = " + repr(tgy_genre_abbr))
                    print("tgy_genre_name = " + repr(tgy_genre_name))
                    print("tgy_genre_url = " + repr(tgy_genre_url))
                    print("tgy_YYYY = " + repr(tgy_YYYY))
                    print("tgy_domUSsales_dollars_rank_" + tgy_YYYY + " = " + repr(tgy_domUSsales_dollars_rank_boxOfficeYear))
                    print("tgy_domUSsales_dollars_" + tgy_YYYY + " = " + repr(tgy_domUSsales_dollars_boxOfficeYear))
                    print("tgy_domUSsales_tickets_" + tgy_YYYY + " = " + repr(tgy_domUSsales_tickets_boxOfficeYear))
                    print("")
                    print("========================")
                    print("")

                tgy_doss = dict()
                tgy_doss['01_film_handle'] = film_handle
                tgy_doss['03_genre_abbr'] = tgy_genre_abbr
                # tgy_doss['tgy_film_title'] = tgy_film_title
                # tgy_doss['tgy_film_url'] = tgy_film_url
                tgy_doss['04_distrib_handle'] = tgy_distrib_handle
                tgy_doss['tgy_boxOfficeYears'] = tgy_desired_year
                tgy_doss['tgy_release_dates'] = tgy_release_date
                # tgy_doss['tgy_domUSsales_dollars_rank_' + tgy_YYYY] = tgy_domUSsales_dollars_rank_boxOfficeYear
                tgy_doss['tgy_domUSsales_dollars_' + tgy_YYYY] = tgy_domUSsales_dollars_boxOfficeYear
                tgy_doss['tgy_domUSsales_tickets_' + tgy_YYYY] = tgy_domUSsales_tickets_boxOfficeYear

                # Check whether a dossier already exists for this movieHandle.
                # If a dossier does not already exist, create an empty dossier for it.
                try:
                    old_doss = movieHandle_to_movieDoss[film_handle]
                except:
                    movieHandle_to_movieDoss[film_handle] = dict()
                # Assign values for top-level keys
                # Top-level key-value pairs contain static (non-time-dependent) information about the film
                for k in tgy_doss.keys():
                    new_val = tgy_doss[k]
                    if k in ["tgy_boxOfficeYears", "tgy_release_dates"]:
                        # Check whether this k-list is already present for this movieHandle
                        # If the k-list does not already exist, create it as an empty list.
                        try:
                            old_val = movieHandle_to_movieDoss[film_handle][k]
                        except:
                            movieHandle_to_movieDoss[film_handle][k] = list()
                        # Append the new value to the k-list
                        movieHandle_to_movieDoss[film_handle][k].append(new_val)

                        if k in ["tgy_release_dates"]:
                            # force update of "tgy_release_date", because a new date
                            # entry may exist for the current movieHandle
                            dates_raw = movieHandle_to_movieDoss[film_handle][k]
                            dates_valid = [d for d in dates_raw if isinstance(d, datetime.date)]
                            if len(dates_valid) == 0:
                                date_earliest = None
                            else:
                                date_earliest = min(dates_valid)
                                if not isinstance(date_earliest, datetime.date):
                                    print("The earliest date in dates_valid is not a date..." + repr(dates_valid))
                                    print("dates_valid = " + repr(dates_valid))
                                    print("date_earliest = " + repr(date_earliest))
                                    print()
                            movieHandle_to_movieDoss[film_handle]["tgy_release_date"] = date_earliest

                    else:
```

```
                        # Check whether key k is already defined in the dossier
                        try:
                            old_val = movieHandle_to_movieDoss[film_handle][k]
                        except:
                            movieHandle_to_movieDoss[film_handle][k] = new_val

                if not tgy_distrib_handle is None:
                    distrib_doss = dict()
                    distrib_doss['distrib_href'] = tgy_distrib_href
                    distrib_doss['distrib_handle'] = tgy_distrib_handle
                    distrib_doss['distrib_name'] = tgy_distrib_name
                    distrib_doss['distrib_url'] = tgy_distrib_url

                    # Check whether a dossier already exists for this distribHandle.
                    # If a dossier does not already exist, create an empty dossier for it.
                    try:
                        old_doss = distribHandle_to_distribDoss[tgy_distrib_handle]
                    except:
                        distribHandle_to_distribDoss[tgy_distrib_handle] = dict()
                    # Assign values for top-level keys
                    # Top-level key-value pairs contain static (non-time-dependent) information about the film
                    for k in distrib_doss.keys():
                        # Check whether key k is already defined in the dossier
                        new_val = distrib_doss[k]
                        try:
                            old_val = distribHandle_to_distribDoss[tgy_distrib_handle][k]
                        except:
                            distribHandle_to_distribDoss[tgy_distrib_handle][k] = new_val

                if not tgy_genre_handle is None:
                    genre_doss = dict()
                    genre_doss['genre_href'] = tgy_genre_href
                    genre_doss['genre_handle'] = tgy_genre_handle
                    genre_doss['genre_abbr'] = tgy_genre_abbr
                    genre_doss['genre_name'] = tgy_genre_name
                    genre_doss['genre_url'] = tgy_genre_url

                    # Check whether a dossier already exists for this genreHandle.
                    # If a dossier does not already exist, create an empty dossier for it.
                    try:
                        old_doss = genreHandle_to_genreDoss[tgy_genre_handle]
                    except:
                        genreHandle_to_genreDoss[tgy_genre_handle] = dict()
                    # Assign values for top-level keys
                    # Top-level key-value pairs contain static (non-time-dependent) information about the film
                    for k in genre_doss.keys():
                        # Check whether key k is already defined in the dossier
                        new_val = genre_doss[k]
                        try:
                            old_val = genreHandle_to_genreDoss[tgy_genre_handle][k]
                        except:
                            genreHandle_to_genreDoss[tgy_genre_handle][k] = new_val

            tgy_rows_cumulative += num_data_rows
            comm4("All {" + str(num_data_rows)+ "} data rows in the table were processed as expected.")
            comm4("")

            quite_done = True
            issues_found = False

        if issues_found:
            print("ISSUES were encountered while processing URL {" + tgy_table_url + "}.")
            comm2("A total of {" + str(tgy_rows_cumulative)+ "} data rows were processed, so far.")
            comm2("")
        else:
            comm2("Data from URL {" + tgy_table_url + "} was processed without issues.")
            comm4("A total of {" + str(tgy_rows_cumulative)+ "} data rows were processed, so far.")
            comm4("")


# ============================================================================
#   IN THIS SECTION,
#       POST-PROCESSING OF THE DATA HARVESTED ABOVE IS PERFORMED.
#
#       EACH MODE OF POST-HARVEST PROCESSING
#       MAY OPTIONALLY BE PERFORMED AT THE DISCRETION OF THE USER
```

```
#       BY SETTING "identify_????" FLAGS AND/OR
#                  "mend_????" FLAGS
#       TO True or False, ACCORDINGLY
# ============================================================================


identify_unique_movieHandles = True
if not identify_unique_movieHandles:
    print("SKIPPING identification of unique movie handles...")
    print()
else:
    print("IDENTIFYING unique movie handles...")
    print()

    # Make a list of all unique movie handles that were encountered
    # during the harvesting of data
    movieHandles_all = sorted([k for k in movieHandle_to_movieDoss.keys()],
                              key=lambda s: s.lower())
    num_movieHandles = len(movieHandles_all)
    print("num_movieHandles = " + str(num_movieHandles))
    print()
    num_display = 50
    print("MOVIE HANDLES:  FIRST " + str(num_display) +
          " AND LAST " + str(num_display) + "...")
    i_all = list(range(num_movieHandles))
    i_show = i_all[0:num_display] + i_all[-num_display:]
    for i in i_show:
        print("movieHandles_all[" + str(i) + "] = " + movieHandles_all[i])
    print()

identify_unique_genreHandles = True
if not identify_unique_genreHandles:
    print("SKIPPING identification of unique genre handles...")
    print()
else:
    print("IDENTIFYING unique genre handles...")
    print()

    # Make a list of all unique genre handles that were encountered
    # during the harvesting of data
    genreHandles_all = sorted([k for k in genreHandle_to_genreDoss.keys()],
                              key=lambda s: s.lower())
    num_genreHandles = len(genreHandles_all)
    print("num_genreHandles = " + str(num_genreHandles))
    print()
    num_display = 50
    if num_genreHandles < (2 * num_display):
        print("GENRE HANDLES:  ALL " + str(num_genreHandles) + " HANDLES...")
        i_show = list(range(num_genreHandles))
    else:
        print("GENRE HANDLES:  FIRST " + str(num_display) +
              " AND LAST " + str(num_display) + "...")
        i_all = list(range(num_genreHandles))
        i_show = i_all[0:num_display] + i_all[-num_display:]
    for i in i_show:
        print("genreHandles_all[" + str(i) + "] = " + genreHandles_all[i])
    print()

identify_unique_distribHandles = True
if not identify_unique_distribHandles:
    print("SKIPPING identification of unique distributor handles...")
    print()
else:
    print("IDENTIFYING unique distributor handles...")
    print()

    # Make a list of all unique distributor handles that were encountered
    # during the harvesting of data
    distribHandles_all = sorted([k for k in distribHandle_to_distribDoss.keys()],
                                key=lambda s: s.lower())
    num_distribHandles = len(distribHandles_all)
    print("num_distribHandles = " + str(num_distribHandles))
    print()
    num_display = 50
    if num_distribHandles < (2 * num_display):
        print("DISTRIBUTOR HANDLES:  ALL " + str(num_distribHandles) + " HANDLES...")
```

```
        i_show = list(range(num_distribHandles))
    else:
        print("DISTRIBUTOR HANDLES:  FIRST " + str(num_display) +
              " AND LAST " + str(num_display) + "...")
        i_all = list(range(num_distribHandles))
        i_show = i_all[0:num_display] + i_all[-num_display:]
    for i in i_show:
        print("distribHandles_all[" + str(i) + "] = " + distribHandles_all[i])
    print()


mend_genre_abbrs = True
if not mend_genre_abbrs:
    print("SKIPPING mending of genre abbreviations...")
    print()
else:
    print("PERFORMING mending of genre abbreviations...")
    print()

    # Not all movies that were encountered during the harvesting of data
    # had genre information available to be stored in the movie's dossier.

    # Force those movies that are MISSING the genre information to have
    # genre abbreviation TN_GENRE_ABBRS[0], which is regarded as "undefined genre"

    for mh in movieHandles_all:
        g_abbr = None
        try:
            g_abbr = movieHandle_to_movieDoss[mh]["03_genre_abbr"]
        except:
            pass

        if not g_abbr in TN_GENRE_ABBRS:
            movieHandle_to_movieDoss[mh]["03_genre_abbr"] = TN_GENRE_ABBRS[0]


mend_release_dates = True
if not mend_release_dates:
    print("SKIPPING mending of release dates...")
    print()
else:
    print("PERFORMING mending of release dates...")
    print()

    # Not all movies that were encountered during the harvesting of data
    # had "release date" information available to be stored in the movie's dossier.

    # To complicate things further, the "release date" information harvested from
    # source A (if release date information was in fact obtained from source A)
    # MAY differ from the "release date" information harvested from source B (if
    # release date information was in fact obtained from source B).

    # In the event that there appear to be two different values of "release date"
    # for the same movie, the EARLIER of the two values will be regarded as correct.

    for mh in movieHandles_all:
        date_A = "--absent--"
        try:
            date_A = movieHandle_to_movieDoss[mh]["bgt_release_date"]
        except:
            pass

        date_B = "--absent--"
        try:
            date_B = movieHandle_to_movieDoss[mh]["tgy_release_date"]
        except:
            pass

        if not isinstance(date_A, datetime.date):
            if not isinstance(date_B, datetime.date):
                date_release = None
            else:
                date_release = date_B
        else:
            if not isinstance(date_B, datetime.date):
                date_release = date_A
```

```
            else:
                date_release = min(date_A, date_B)

        movieHandle_to_movieDoss[mh]["02_film_release_date"] = date_release
        year_release = date_to_boxOfficeYear(date_release)
        if isinstance(year_release, int):
            pass
        if not isinstance(year_release, int):
            if not year_release is None:
                print("Error: date_to_boxOfficeYear(" + repr(date_release)+ ") = " + repr(year_release))
        movieHandle_to_movieDoss[mh]["02_film_release_year"] = year_release

    print("all movieHandles whose 'release_date' is a non-date, AFTER mending of release dates:")
    movieHandles_undated = [mh for mh in movieHandles_all if not
                            isinstance(movieHandle_to_movieDoss[mh]['02_film_release_date'], datetime.date)]
    for mh in movieHandles_undated:
        mh_date_keys = [k for k in movieHandle_to_movieDoss[mh].keys() if "date" in k]
        mh_date_values = [movieHandle_to_movieDoss[mh][dk] for dk in mh_date_keys]
        mh_date = dict(zip(mh_date_keys, mh_date_values))
        print("     " + repr(mh) + ": " + repr(mh_date) )
    print()


display_selected_movie_dossiers = True
if not display_selected_movie_dossiers:
    print("SKIPPING display of selected movie dossiers...")
    print()
else:
    print("DISPLAYING selected movie dossiers...")
    print()

    num_display = 20
    print("MOVIE DOSSIERS:  FIRST " + str(num_display) +
          " AND LAST " + str(num_display) + "...")
    i_all = list(range(num_movieHandles))
    i_show = i_all[0:num_display] + i_all[-num_display:]
    for i in i_show:
        print("i = " + str(i) + ":")
        mh_doss = movieHandle_to_movieDoss[movieHandles_all[i]]
        mh_keys = sorted(mh_doss.keys(), key=lambda s: s.lower())
        for k in mh_keys:
            print(repr(k) + ": " + repr(mh_doss[k]))
        print()



# ============================================================================
#   IN THIS SECTION,
#       SOME OF THE INFORMATION STORED IN movieHandle_to_movieDoss
#       IS PORTED INTO A NEW DICTIONARY CALLED "eventHandle_to_eventDoss"
#
#       TO BE PORTED INTO THE EVENTS DICTIONARY, A PARTICULAR MOVIE
#       MUST ALREADY CONTAIN SPECIFIC ITEMS OF INFORMATION IN ITS MOVIE DOSSIER.
#
#       EACH MOVIE THAT APPEARS IN THE EVENTS DICTIONARY HAS AT LEAST ONE
#       "EVENT" ASSOCIATED WITH IT.
#
#       THE COMBINATION OF:
#           [ONE BOX-OFFICE SALES YEAR] + [ONE GENRE] + [ONE MOVIE FROM WITHIN THAT GENRE]
#       COMPRISE A UNIQUE "EVENT HANDLE".
#
#       FOR EXAMPLE, THE 2009 ACTION MOVIE "AVATAR" HAD DOMESTIC US BOX OFFICE
#       SALES DURING THE BOX OFFICE YEARS 2009 AND 2010.  THE INFORMATION FOR
#       THE MOVIE AVATAR IS THEREFORE STORED IN THE EVENTS DICTIONARY
#       UNDER THESE TWO EVENT HANDLES.
#            EVENT HANDLE 1 = "2009/ACTION/AVATAR"
#            EVENT HANDLE 2 = "2010/ACTION/AVATAR"
#
#       ORGANIZATION OF THE MOVIE SALES FIGURES IN THIS FASHION FACILITATES
#       ANALYSIS USING THE PANDAS "GROUPBY" FUNCTION.
# ============================================================================


movieHandles_in_eventDoss = []

create_eventHandle_to_eventDoss = True
```

```
if not create_eventHandle_to_eventDoss:
    print("SKIPPING creation of eventHandle_to_eventDoss...")
    print()
else:
    print("PERFORMING creation of eventHandle_to_eventDoss...")
    print()

    eventHandle_to_eventDoss = dict()

    for mh in movieHandles_all:
        mh_BOYs = None
        try:
            mh_BOYs = movieHandle_to_movieDoss[mh]['tgy_boxOfficeYears']
        except:
            print("didn't find movieHandle_to_movieDoss[" + repr(mh) + "]['tgy_boxOfficeYears']")
            pass
        if isinstance(mh_BOYs, list):
            mh_relDate = None
            try:
                mh_relDate = movieHandle_to_movieDoss[mh]['02_film_release_date']
            except:
                print("didn't find movieHandle_to_movieDoss[" + repr(mh) + "]['02_film_release_date']")
                pass
            if isinstance(mh_relDate, datetime.date):
                mh_pb = None
                try:
                    mh_pb = movieHandle_to_movieDoss[mh]['bgt_productionBudget_dollars']
                except:
                    print("didn't find movieHandle_to_movieDoss[" + repr(mh) + "]['bgt_productionBudget_dollars']")
                    pass
                if isinstance(mh_pb, int):
                    mh_domUS_gross = None
                    try:
                        mh_domUS_gross = movieHandle_to_movieDoss[mh]['bgt_domUS_gross_dollars']
                    except:
                        print("didn't find movieHandle_to_movieDoss[" + repr(mh) + "]['bgt_domUS_gross_dollars']")
                        pass
                    if isinstance(mh_domUS_gross, int):
                        mh_world_gross = None
                        try:
                            mh_world_gross = movieHandle_to_movieDoss[mh]['bgt_world_gross_dollars']
                        except:
                            print("didn't find movieHandle_to_movieDoss[" + repr(mh) + "]['bgt_world_gross_dollars']")
                            pass
                        if isinstance(mh_world_gross, int):
                            for iYear in mh_BOYs:
                                event_iYear = iYear
                                event_YYYY = str(event_iYear)
                                event_genre_abbr = movieHandle_to_movieDoss[mh]["03_genre_abbr"]
                                event_movie_handle = mh

                                event_handle = event_YYYY + "/" + event_genre_abbr + "/" + event_movie_handle
                                event_genreYear_handle = event_genre_abbr + "/" + event_YYYY

                                mh_sales = None
                                try:
                                    mh_sales = movieHandle_to_movieDoss[mh]['tgy_domUSsales_dollars_' + event_YYYY]
                                except:
                                    pass
                                if isinstance(mh_sales, int):
                                    mh_tickets = None
                                    try:
                                        mh_tickets = movieHandle_to_movieDoss[mh]['tgy_domUSsales_tickets_' + event_YYYY]
                                    except:
                                        pass
                                    if isinstance(mh_tickets, int):
                                        pb_year = movieHandle_to_movieDoss[mh]['02_film_release_year']
                                        pb_raw = movieHandle_to_movieDoss[mh]['bgt_productionBudget_dollars']
                                        pb_mult = inflation_multiplier(from_year=pb_year, into_year=2020)
                                        pb_adj = pb_raw * pb_mult

                                        domUS_raw = movieHandle_to_movieDoss[mh]['tgy_domUSsales_dollars_' + event_YYYY]
                                        domUS_mult = inflation_multiplier(from_year=event_iYear, into_year=2020)
                                        domUS_adj = domUS_raw * domUS_mult

                                        event_doss = dict()
```

```
                                    event_doss["event_handle"] = event_handle
                                    event_doss["genreYear_handle"] = event_genreYear_handle
                                    event_doss['boxOfficeYear'] = event_iYear
                                    event_doss["genre_abbr"] = event_genre_abbr
                                    event_doss["movie_handle"] = event_movie_handle
                                    event_doss["release_date"] = movieHandle_to_movieDoss[mh]['02_film_release_date']
                                    event_doss["productionBudget_year"] = pb_year
                                    event_doss["productionBudget_raw"] = pb_raw
                                    event_doss["productionBudget_mult"] = pb_mult
                                    event_doss["productionBudget_adj"] = pb_adj
                                    event_doss["total_world_gross"] = movieHandle_to_movieDoss[mh]['bgt_world_gross_dollars']
                                    event_doss["total_domUS_gross"] = movieHandle_to_movieDoss[mh]['bgt_domUS_gross_dollars']
                                    event_doss["boxOfficeYear_domUS_raw"] = domUS_raw
                                    event_doss["boxOfficeYear_domUS_mult"] = domUS_mult
                                    event_doss["boxOfficeYear_domUS_adj"] = domUS_adj
                                    event_doss["boxOfficeYear_domUS_tickets"] = movieHandle_to_movieDoss[mh]['tgy_domUSsales_tickets_' + event_YYYY]
                                    eventHandle_to_eventDoss[event_handle] = event_doss

                            movieHandles_in_eventDoss.append(mh)
    print()

    num_movieHandles_in_eventDoss = len(movieHandles_in_eventDoss)
    print("BEFORE ELMINATION OF DUPLICATES:")
    print("num_movieHandles_in_eventDoss = " + str(num_movieHandles_in_eventDoss))
    print()

    movieHandles_in_eventDoss = sorted(list(set(movieHandles_in_eventDoss)))
    num_movieHandles_in_eventDoss = len(movieHandles_in_eventDoss)
    print("AFTER ELMINATION OF DUPLICATES:")
    print("num_movieHandles_in_eventDoss = " + str(num_movieHandles_in_eventDoss))
    print()


    eventHandles_all = sorted([k for k in eventHandle_to_eventDoss.keys()],
                              key=lambda s: s.lower())
    num_eventHandles = len(eventHandles_all)
    print("num_eventHandles = " + str(num_eventHandles))
    print()

    # make the event handles in the events dictionary appear in increasing order by eventHandle
    eventHandle_to_eventDoss_RAW = eventHandle_to_eventDoss
    eventHandle_to_eventDoss = dict()
    for k in eventHandles_all:
        eventHandle_to_eventDoss[k] = eventHandle_to_eventDoss_RAW[k]
    eventHandle_to_eventDoss_RAW = None

    num_display = 50
    print("EVENT HANDLES:  FIRST " + str(num_display) +
          " AND LAST " + str(num_display) + "...")
    i_all = list(range(num_eventHandles))
    i_show = i_all[0:num_display] + i_all[-num_display:]
    for i in i_show:
        print("eventHandles_all[" + str(i) + "] = " + eventHandles_all[i])
    print()

    num_display = 20
    print("EVENT DOSSIERS:  FIRST " + str(num_display) +
          " AND LAST " + str(num_display) + "...")
    i_all = list(range(num_eventHandles))
    i_show = i_all[0:num_display] + i_all[-num_display:]
    for i in i_show:
        print("i = " + str(i) + ":")
        eh_doss = eventHandle_to_eventDoss[eventHandles_all[i]]
        eh_keys = eh_doss.keys()
        for k in eh_keys:
            print(repr(k) + ": " + repr(eh_doss[k]))
        print()


# Make a sound, to let the user know it's time to move on to the next cell
make_beeps()

# That should do it.  Thanks for reading my code.  -- MLC
print("Get thee to the next cell.")

do_extra_stuff = False
```

```
if do_extra_stuff:
    # This is extra stuff
    # so please don't knock me for lack of commentary

    further_review_data = False
    if not further_review_data:
        print("SKIPPING further review of data...")
        print()
    else:
        print("PERFORMING further review of data...")
        print()

        print("bgt_rows_cumulative = " + str(bgt_rows_cumulative))
        print()

        print("tgy_rows_cumulative = " + str(tgy_rows_cumulative))
        print()

        movieHandles_length = [len(h) for h in movieHandles_all]
        movieHandles_length_min = min(movieHandles_length)
        movieHandles_length_max = max(movieHandles_length)
        movieHandles_shortest = [h for h in movieHandles_all if len(h) == movieHandles_length_min]
        movieHandles_longest = [h for h in movieHandles_all if len(h) == movieHandles_length_max]
        print("movieHandles_length_min = " + str(movieHandles_length_min))
        print("movieHandles_length_max = " + str(movieHandles_length_max))
        print("movieHandles_shortest = " + repr(movieHandles_shortest))
        print("movieHandles_longest = " + repr(movieHandles_longest))
        print()

        try:
            movieHandles_with_releaseDate = [h for h in movieHandles_all if not movieHandle_to_movieDoss[h]['02_film_release_date'] is None]
            num_movieHandles_with_releaseDate = len(movieHandles_with_releaseDate)
            print("num_movieHandles_with_releaseDate = " + str(num_movieHandles_with_releaseDate))
            print()


            movieHandles_releaseDate = [movieHandle_to_movieDoss[h]['02_film_release_date'] for h in movieHandles_with_releaseDate]
            movieHandles_releaseDate_min = min(movieHandles_releaseDate)
            movieHandles_releaseDate_max = max(movieHandles_releaseDate)
            movieHandles_earliest = [h for h in movieHandles_with_releaseDate if
                                     movieHandle_to_movieDoss[h]['02_film_release_date'] == movieHandles_releaseDate_min]
            movieHandles_latest = [h for h in movieHandles_with_releaseDate if
                                   movieHandle_to_movieDoss[h]['02_film_release_date'] == movieHandles_releaseDate_max]
            print("movieHandles_releaseDate_min = " + str(movieHandles_releaseDate_min))
            print("movieHandles_releaseDate_max = " + str(movieHandles_releaseDate_max))
            print("movieHandles_earliest = " + repr(movieHandles_earliest))
            print("movieHandles_latest = " + repr(movieHandles_latest))
            print()
        except:
            print("skipping movieHandles_releaseDate analysis")
            print()

        try:
            movieHandles_numBOY = [len(movieHandle_to_movieDoss[h]['boxOfficeYears']) for h in movieHandles_all]
            movieHandles_numBOY_min = min(movieHandles_numBOY)
            movieHandles_numBOY_max = max(movieHandles_numBOY)
            movieHandles_numBOY_highest = [h for h, numBOY in zip(movieHandles_all, movieHandles_numBOY)
                                           if numBOY == movieHandles_numBOY_max]
            print("movieHandles_numBOY_min = " + str(movieHandles_numBOY_min))
            print("movieHandles_numBOY_max = " + str(movieHandles_numBOY_max))
            print("movieHandles_numBOY_highest = " + str(movieHandles_numBOY_highest))
            for mh in movieHandles_numBOY_highest:
                print("movieHandle_to_movieDoss[" + repr(mh) + "]['boxOfficeYears'] = ")
                print(repr(movieHandle_to_movieDoss[mh]['boxOfficeYears']))
                print()
        except:
            print("skipping movieHandles_numBOY analysis")
            print()

    # This is a nice way to end program execution, in a Jupyter Notebook
    print("Purposely Terminating program execution, now.")
    assert False
```

'productionBudget_year': 2019

'productionBudget_year': 2019
'productionBudget_raw': 65000000
'productionBudget_mult': 1.00934
'productionBudget_adj': 65607099.99999999
'total_world_gross': 39194264
'total_domUS_gross': 17291078
'boxOfficeYear_domUS_raw': 17291078
'boxOfficeYear_domUS_mult': 1.0
'boxOfficeYear_domUS_adj': 17291078.0
'boxOfficeYear_domUS_tickets': 1898032

i = 6785:
'event_handle': '2020/THRILLER/Unhinged-(2020)'
'genreYear_handle': 'THRILLER/2020'
'boxOfficeYear': 2020
'genre_abbr': 'THRILLER'
'movie_handle': 'Unhinged-(2020)'
'release_date': datetime.date(2020, 7, 16)
'productionBudget_year': 2020
'productionBudget_raw': 33000000

In [3]:

```
#  Follow-on activity:
#  Use values stored in the eventHandle_to_eventDoss dictionary
#  to populate a pandas dataframe, "df"
#
#  MICHAEL COLLINS, 2020-09-11_2107_MDT


# Force seaborn to make the backgrounds of graphs white, instead of transparent
# Kudos to my classmate Gustavo Chavez for offering a solution to this problem
sns.set_style("white")

# This deletes the file specified with name strFile, if the file exists.
# Then it issues a plt.savefig() to save the current matplotlib to a new
# instance of file strFile.
def delete_then_plt_savefig(strFile):
    if os.path.isfile(strFile):
        os.remove(strFile)   # Opt.: os.system("rm "+strFile)
    plt.savefig(strFile)
    return

# Construct a list of all unique event-level keys that will be encountered
print("All the event-level keys in eventHandle_to_eventDoss, as eventDoss_keys_all:")
keys_sorted = sorted(eventHandle_to_eventDoss.keys())
eventDoss_keys_all = []
for eh in keys_sorted:
    eventDoss = eventHandle_to_eventDoss[eh]
    eventDoss_keys = eventDoss.keys()
    for k in eventDoss_keys:
        if not k in eventDoss_keys_all:
            eventDoss_keys_all.append(k)
for i, dk in enumerate(eventDoss_keys_all):
    print("i=" + str(i) + ", " + dk)
print()

# Determine exactly which event-level keys should be ported into the dataframe
def keep_eventDoss_key(key):
    if key in ['event_handle']:
        return False
    return True
eventDoss_keys_use = [k for k in eventDoss_keys_all if keep_eventDoss_key(k)]

# Load the pandas dataframe with desired data from eventHandle_to_eventDoss
df = pd.DataFrame.from_dict(eventHandle_to_eventDoss, orient='index', columns=eventDoss_keys_use).fillna(0)
print("df {initial values, based on eventHandle_to_eventDoss}... ")
display(df)
print()

# These are names of existing (and some possible future) columns within the dataframe
col_genre = 'genre_abbr'
col_year = 'boxOfficeYear'
col_genreYear = "genreYear_handle"
col_budget = 'productionBudget_adj'
col_log10_budget = 'log10_productionBudget_adj'
col_boy_domUS_sales = 'boxOfficeYear_domUS_adj'
col_log10_boy_domUS_sales = 'log10_boxOfficeYear_domUS_adj'
col_world_gross = "total_world_gross"
col_log10_world_gross = "log10_total_world_gross"
col_symbol_size = "symbol_size"
col_genre_num = "genre_num"
col_sbRatio = "sbRatio"
col_log10_sbRatio = "log10_sbRatio"

# These are new columns being added to the dataframe
df[col_sbRatio] = df[col_world_gross]/df[col_budget]
df[col_log10_sbRatio] = np.log10(df[col_sbRatio])
df[col_log10_boy_domUS_sales] = np.log10(df[col_boy_domUS_sales])
df[col_log10_budget] = np.log10(df[col_budget])
df[col_log10_world_gross] = np.log10(df[col_world_gross])
budget_max = np.max(df[col_budget])
df[col_symbol_size] = 100*df[col_budget]/budget_max
u, df[col_genre_num] = np.unique(df[col_genre], return_inverse=True)

# Show the dataframe after the new columns are added
print("df {after adding new columns}... ")
display(df)
```

```
print()

make_IMAGE_1 = True
if make_IMAGE_1:
    fig, ax = None, None
    print("making a strip plot... ")
    print()
    fig, ax = plt.subplots(figsize=(10,8))
    sns.stripplot(data=df, y=col_genre, x=col_year, order=TN_GENRE_ABBRS,
                  alpha=0.2, jitter=0.25, dodge=True, orient='h', marker="o", s=8)
    plt.ylabel(None, size=26)
    plt.xlabel("Box Office Year", size=16)
    plt.grid()
    plt.yticks(fontsize=14)
    plt.xlim([1975, 2025])
    plt.xticks(np.arange(1975, 2026, 5.0))
    plt.xticks(fontsize=14)
    plt.title("Movies by Genre and Box Office Year", size=20)
    plt.tight_layout()
    delete_then_plt_savefig("DOSFLIX_Movies_by_Genre_and_boxOfficeYear.png")
    plt.show()


make_IMAGE_2 = True
if make_IMAGE_2:
    fig, ax = None, None
    print("making [sales vs. boxOfficeYear] scatter plot... ")
    print("for movies from ALL GENRES combined... ")
    print()
    fig, ax = plt.subplots(figsize=(10,8))
    sc = ax.scatter(data=df, y=col_log10_boy_domUS_sales, x=col_year, c=col_genre_num,
                    alpha=0.8, marker="o", s=col_symbol_size)
    ax.legend(sc.legend_elements()[0], u)
    ax.grid()
    ax.set_ylim(ymin=3, ymax=10)
    plt.yticks(np.arange(3, 11, 1.0))
    plt.yticks(fontsize=14)
    ax.set_xlim(xmin=1975, xmax=2025)
    plt.xticks(np.arange(1975, 2026, 5.0))
    plt.xticks(fontsize=14)
    s_title = "ALL GENRES represented: Log10(Sales) vs. boxOfficeYear"
    plt.title(s_title, size=24)
    plt.ylabel('Log10(Annual_US_Sales, ADJUSTED usd)', size=18)
    plt.xlabel('Box Office Year', size=18)
    fig_saveat = "DOSFLIX_Annual_US_Sales_vs_boxOfficeYear,ALL_GENRES.png"
    delete_then_plt_savefig(fig_saveat)
    plt.show()
    for n, grp in df.groupby(col_genre):
        fig, ax = None, None
        print("making [sales vs. boxOfficeYear] scatter plot")
        print("for movies from GENRE=" + str(n) + ", specifically... ")
        print()
        fig, ax = plt.subplots(figsize=(10,8))
        c_n = gAbbr_to_gIndex[n]
        c_list = [c_n for i in range(grp.shape[0])]
        ax.scatter(data=grp, y=col_log10_boy_domUS_sales, x=col_year,
                   alpha=0.8, marker="o", s=col_symbol_size)
        ax.legend(title="Genre = " + n)
        ax.grid()
        ax.set_ylim(ymin=3, ymax=10)
        plt.yticks(np.arange(3, 11, 1.0))
        plt.yticks(fontsize=14)
        ax.set_xlim(xmin=1975, xmax=2025)
        plt.xticks(np.arange(1975, 2026, 5.0))
        plt.xticks(fontsize=14)
        s_title = "GENRE=" + str(n) + ": Log10(Sales) vs. boxOfficeYear"
        plt.title(s_title, size=24)
        plt.ylabel('Log10(Annual_US_Sales, ADJUSTED usd)', size=18)
        plt.xlabel('Box Office Year', size=18)
        fig_saveat = "DOSFLIX_Annual_US_Sales_vs_boxOfficeYear,GENRE=" + str(n) + ".png"
        delete_then_plt_savefig(fig_saveat)
        plt.show()
    # the plots generated using delete_then_plt_savefig, above, were
    # stitched together into an animated GIF "manually" via the website
    # https://ezgif.com/

```

```
152
153 make_IMAGE_3 = True
154 if make_IMAGE_3:
155     fig, ax = None, None
156     print("making [sales vs. budget] scatter plot")
157     print("for movies from ALL GENRES combined... ")
158     print()
159     fig, ax = plt.subplots(figsize=(10,8))
160     for n, grp in df.groupby(col_genre):
161     #     print("n = " + repr(n))
162     #     print("grp.shape[0] = " + repr(grp.shape[0]))
163         c_n = gAbbr_to_gIndex[n]
164         c_list = [c_n for i in range(grp.shape[0])]
165         ax.scatter(data=grp, y=col_log10_world_gross, x=col_log10_budget, label=n,
166                    alpha=0.8, marker=",", s=1)
167     ax.legend(title="Genre")
168     plt.ylim([3, 10])
169     plt.yticks(np.arange(3, 11, 1.0))
170     plt.yticks(fontsize=14)
171     plt.xlim([3, 9])
172     plt.xticks(np.arange(3, 10, 1.0))
173     plt.xticks(fontsize=14)
174     plt.grid()
175     plt.title("ALL GENRES represented: World Gross vs. Budget", size=24)
176     plt.ylabel('Log10(TotalWorldwideSales, usd)', size=18)
177     plt.xlabel('Log10(Production_Budget, ADJUSTED usd)', size=18)
178     delete_then_plt_savefig("DOSFLIX_worldGross_vs_ProdBudget,ALL_GENRES.png")
179     plt.show()
180
181     fig, ax = None, None
182     for n, grp in df.groupby(col_genre):
183         print("making [sales vs. budget] scatter plots")
184         print("for movies from GENRE=" + str(n) + ", specifically... ")
185         print()
186         fig, ax = plt.subplots(figsize=(10,8))
187         c_n = gAbbr_to_gIndex[n]
188         c_list = [c_n for i in range(grp.shape[0])]
189         ax.scatter(data=grp, y=col_log10_world_gross, x=col_log10_budget,
190                    alpha=0.2, marker="o", s=10)
191         ax.legend(title="Genre = " + n)
192         plt.ylim([3, 10])
193         plt.yticks(np.arange(3, 11, 1.0))
194         plt.yticks(fontsize=14)
195         plt.xlim([3, 9])
196         plt.xticks(np.arange(3, 10, 1.0))
197         plt.xticks(fontsize=14)
198         plt.grid()
199         s_title = "GENRE=" + str(n) + ": World Gross vs. Budget"
200         plt.title(s_title, size=24)
201         plt.ylabel('Log10(TotalWorldwideSales, usd)', size=18)
202         plt.xlabel('Log10(Production_Budget, ADJUSTED usd)', size=18)
203
204         fig_saveat = "DOSFLIX_worldGross_vs_ProdBudget,GENRE=" + str(n) + ".png"
205         delete_then_plt_savefig(fig_saveat)
206         plt.show()
207     # the plots generated using delete_then_plt_savefig, above, were
208     # stitched together into an animated GIF "manually" via the website
209     # https://ezgif.com/
210
211 make_IMAGE_4 = True
212 if make_IMAGE_4:
213     fig, ax = None, None
214     print("making [sbRatio vs. budget] scatter plot")
215     print("for movies from ALL GENRES combined... ")
216     print()
217     fig, ax = plt.subplots(figsize=(10,8))
218     for n, grp in df.groupby(col_genre):
219     #     print("n = " + repr(n))
220     #     print("grp.shape[0] = " + repr(grp.shape[0]))
221         c_n = gAbbr_to_gIndex[n]
222         c_list = [c_n for i in range(grp.shape[0])]
223         ax.scatter(data=grp, y=col_log10_sbRatio, x=col_log10_budget, label=n,
224                    alpha=0.8, marker=",", s=1)
225     ax.legend(title="Genre")
226     plt.ylim([-6, 3])
227     plt.yticks(np.arange(-6, 4, 1.0))
```

```
    plt.yticks(fontsize=14)
    plt.xlim([3, 9])
    plt.xticks(np.arange(3, 10, 1.0))
    plt.xticks(fontsize=14)
    plt.grid()
    plt.title("ALL GENRES represented: sbRatio vs. Budget", size=24)
    plt.ylabel('Log10(TotalWorldwideSales / ProdBudget)', size=18)
    plt.xlabel('Log10(Production_Budget, ADJUSTED usd)', size=18)
    delete_then_plt_savefig("DOSFLIX_sbRatio_vs_ProdBudget,ALL_GENRES.png")
    plt.show()

    fig, ax = None, None
    for n, grp in df.groupby(col_genre):
        print("making [sbRatio vs. budget] scatter plots")
        print("for movies from GENRE=" + str(n) + ", specifically... ")
        print()
        fig, ax = plt.subplots(figsize=(10,8))
        c_n = gAbbr_to_gIndex[n]
        c_list = [c_n for i in range(grp.shape[0])]
        ax.scatter(data=grp, y=col_log10_sbRatio, x=col_log10_budget,
                   alpha=0.2, marker="o", s=10)
        ax.legend(title="Genre = " + n)
        plt.ylim([-6, 3])
        plt.yticks(np.arange(-6, 4, 1.0))
        plt.yticks(fontsize=14)
        plt.xlim([3, 9])
        plt.xticks(np.arange(3, 10, 1.0))
        plt.xticks(fontsize=14)
        plt.grid()
        s_title = "GENRE=" + str(n) + ": sbRatio vs. Budget"
        plt.title(s_title, size=24)
        plt.ylabel('Log10(TotalWorldwideSales / ProdBudget)', size=18)
        plt.xlabel('Log10(Production_Budget, ADJUSTED usd)', size=18)

        fig_saveat = "DOSFLIX_sbRatio_vs_ProdBudget,GENRE=" + str(n) + ".png"
        delete_then_plt_savefig(fig_saveat)
        plt.show()
    # the plots generated using delete_then_plt_savefig, above, were
    # stitched together into an animated GIF "manually" via the website
    # https://ezgif.com/




# That should do it.  Thanks for reading my code.  -- MLC
print("That's all, Folks!")

do_extra_stuff = False
if do_extra_stuff:
    # This is extra stuff
    # so please don't knock me for lack of commentary

    def inspect(x):
        print(repr(type(x)) + " " + repr(x))
        return -1

    def robust_weighted_average(x_i, w_i):
        if np.sum(w_i) == 0.0:
            return 0.0
        return np.average(x_i, weights=w_i)

    genre_pb_avg = df.groupby(by=col_genreYear).apply(lambda gb: robust_weighted_average(gb[col_budget], gb[col_boy_domUS_sales])/gb[col_budget].shape[0])
    print("genre_pb_avg... ")
    display(genre_pb_avg)
    print()

    df[col_pb_avg] = df[col_genreYear].map(genre_pb_avg)
    print("df {AFTER col_pb_avg was added to df}... ")
    display(df)
    print()

    genre_pb_avg_sum = df.groupby(by=col_genreYear)[col_pb_avg].sum()
    print("genre_pb_avg_sum... ")
    display(genre_pb_avg_sum)
    print()
```

```
304
```

```
All the event-level keys in eventHandle_to_eventDoss, as eventDoss_keys_all:
i=0, event_handle
i=1, genreYear_handle
i=2, boxOfficeYear
i=3, genre_abbr
i=4, movie_handle
i=5, release_date
i=6, productionBudget_year
i=7, productionBudget_raw
i=8, productionBudget_mult
i=9, productionBudget_adj
i=10, total_world_gross
i=11, total_domUS_gross
i=12, boxOfficeYear_domUS_raw
i=13, boxOfficeYear_domUS_mult
i=14, boxOfficeYear_domUS_adj
i=15, boxOfficeYear_domUS_tickets
```

This is the end of Project1.

/