final_project

March 19, 2024

1 Final Project

2 Disney Data Analysis

2.1 Massimo Savino

2.2 Introduction

- What are we trying to investigate?
- Reasoning
- Dataset details

2.3 Methods & Results

- Précis
- Exceptions and exclusions
- Report characteristics:
  - Read in data
  - Summarize which data is relevant (and which is not)
  - At least 2 relevant data visualizations
  - Chained methods where possible
  - black formatting as per PEP 8
- Python script 1:
  - At least 1 well-designed function for data wrangling & formatting that imports into the notebook
  - Use TDD with at least 1 exception
  - Follows well-designed function fundamentals
- Python script 2:
  - Includes at least 2 unit tests to test the function

2.4 Discussion

- Summarize your findings
- Discuss whether you expected to find this
- Discuss findings' impact
- Discuss what other questions you would like to answer

2.5 References

Note: All tables and figures should have a figure / table number and a title

2.6 Table of Contents

Section	Focus	Report Contents

||

Introduction.

| - **Introductory comments** & research questions | ||| - **Methods and Results**, including Précis, Exceptions & exclusions, and Report characteristics | |

0a.

|

Cleaning files

| - **Methods used to clean the data**, in broad strokes only | |

0b.

|

Tools for scripting and unit tests

| - **Reusable methods and their tests** | ||| - **Unit tests** should use either **Given / When / Then** or **Arrange / Act / Assert** commentary as guides | |

Ia.

|

Movies (general)

| - **movie** by revenue table | ||| - movie by revenue plot | ||| - movie table by inflation-adjusted revenue | ||| - movie plot by inflation-adjusted revenue | |

Ib.

|

Movies (by genre)

| - **genre** by revenue table | ||| - genre by revenue plot | ||| - genre table by inflation-adjusted revenue | ||| - genre plot by inflation-adjusted revenue | |

II.

|

Directors

| - **directors** by revenue table | ||| - directors by revenue plot | ||| - directors table by inflation-adjusted revenue | ||| - directors plot by inflation-adjusted revenue | ||| - directors table in the Adventure genre | ||| - directors plot in the Adventure genre | |

III.

|

Voice actors

| - **voice actors** by revenue table | ||| - voice actors by revenue plot | ||| - voice actors table by inflation-adjusted revenue | ||| - voice actors plot by inflation-adjusted revenue | ||| - voice actors table in the Adventure genre | ||| - voice actors plot in the Adventure genre | |

IVa.

|

Hero (characters)

| - **heroes** by revenue table | ||| - heroes by revenue plot | ||| - heroes table by inflation-adjusted revenue | ||| - heroes plot by inflation-adjusted revenue | ||| - heroes table in the Adventure genre | ||| - heroes plot in the Adventure genre | |

IVb.

|

Villains (characters)

| - **villains** by revenue table | ||| - villains by revenue plot | ||| - villains table by inflation-adjusted revenue | ||| - villains plot by inflation-adjusted revenue | ||| - villains table in the Adventure genre | ||| - villains plot in the Adventure genre | |

V.

|

Conclusions

| **Final observations and wrap-up** |

2.7 Introduction

What are we trying to investigate?

- Who were the most profitable movies, directors, voice-actors and characters for Disney over the reporting period?
- When accounting for inflation, do these change?

Why these questions?

- Discovering which characters, directors, voice actors are most profitable might prove to be useful in determining future revenue potentials for Disney films to come.

Dataset details

- Our dataset used is the Disney Character Success dataset by Kelly Garrett, available at https://data.world/kgarrett/disney-character-success-00-16

2.7.1 Methods and Results

Précis We need to understand the movies file and dataframe as the key driver behind our entire analysis.

I will analyse movies on its own to see how movies perform by revenue, first in a table, then in a plot to visualise them. I will then repeat this for movies against inflation-adjusted revenue, in a table and plot. Finally, I will identify the five most popular genres by revenue, then compare these to inflation-adjusted revenue totals.

Later on, I will merge each 'persons' file (ie `directors`, `voice-actors`, `characters`) with the `movies` file, and generate a table and plot for each.

Spoiler alert: For each 'persons' class I will also find the 5 most popular `person-type`s for the `Adventure` genre category, and publish a table for each, as well as a plot to visualise the table. (So, directors x Adventure, voice-actors x Adventure, heroes x Adventure, and villains x Adventure)

Assembly and cleaning (first run) In responding to the questions we want to answer, we will need to process and clean the five files presented here.

I then loaded needed libraries and the five files into this notebook for preliminary processing.

After looking at the data, I decided to write separate methods to clean each of the files used.

Analysis methods

Exceptions and exclusions / Further cleaning during merges Some of the files only reveal additional quandaries during merging, and so I have tried to limit major cleaning to the files themselves.

However, when merging these files together in various combinations, I have found further issues, and so I limit a further cleaning to the merges themselves, as I believe doing so too early would be bad for the final analysis by excluding too much information.

For example, when merging total film grosses with voice actors, the listing with the highest gross is in fact 'None' where no voice-actor was cited in the film credits. I have taken this entry out, and note this in that section below.

Excluding `revenue` The business unit file `disney_revenue_1991_2016.csv` presents a number of issues to the analysis, not least of which is the fact that much of the data presented there is incomplete (necessarily so, as some units are created from scratch, and then later re-absorbed into different business units of the company, and data is missing that is difficult to reconcile with our needed analysis.

Retaining "0 revenue films" in the `movies` file In the films file (`disney-movies-total-gross.csv`) there are 4 films that did not show any revenue whatsoever. However as the characters file also refers to these in one important case ("The Many Adventures of Winnie the Pooh") I have decided for now to retain these in the final analysis.

Dropping items from the `directors` file Dropping "Full credits" ##### Dropping items from the `voice-actors` file Dropping "None" ##### Dropping items from the `characters` file Dropping no-hero / no-villain items??

Report characteristics

Reading in the data We'll need to import several libraries for proper processing.

We'll also import the whole first script (`reusable_disney_processing.py`) & just call it directly without a namespacing prefix (ie just `method1`, `method2`, etc, without say `rdp.method1` and so forth). For the testing script, I will instead use a namespace, to keep it conceptually separate.

These are: - `altair` for graphics plotting (histograms, frequency counts, etc) - `numpy` for large multi-dimensional arrays and matrix operations - `pandas` for dataframe processing (akin to SQL in pure database form) - `datetime` for the release date fields

As well, we'll need to import the five Disney files; please see below.

[2]:
```
# Library imports
import datetime as dt
import altair as alt
import numpy as np
import pandas as pd

# Importing script files
from reusable_disney_processing import *
from tests_reusable_disney_processing import *

# Reading in the data files
characters = pd.read_csv("data/disney-characters.csv",␣
 ↪parse_dates=["release_date"])
directors = pd.read_csv("data/disney-director.csv")
movies = pd.read_csv("data/disney_movies_total_gross.csv",␣
 ↪parse_dates=["release_date"])
revenue = pd.read_csv("data/disney_revenue_1991-2016.csv")
voice_actors = pd.read_csv("data/disney-voice-actors.csv")

list_of_objects_to_display = [characters, directors, movies, revenue,␣
 ↪voice_actors]

# Want to get a sense of what the raw data looks like
def print_sample_from(list_input):
    for item in list_input:
        print(f"{item.head()}")
        print(f"{item.shape}")
        print(f"{item.dtypes}")

print_sample_from(list_of_objects_to_display)
```

```
                      movie_title release_date        hero     villian  \
0  \nSnow White and the Seven Dwarfs   1937-12-21  Snow White  Evil Queen
1                      \nPinocchio   1940-02-07   Pinocchio   Stromboli
2                       \nFantasia   1940-11-13         NaN   Chernabog
3                            Dumbo   1941-10-23       Dumbo  Ringmaster
4                          \nBambi   1942-08-13       Bambi      Hunter

                           song
0  Some Day My Prince Will Come
1     When You Wish upon a Star
2                           NaN
3                     Baby Mine
4                Love Is a Song
(56, 5)
movie_title             object
release_date    datetime64[ns]
hero                    object
villian                 object
song                    object
dtype: object
                              name        director
0  Snow White and the Seven Dwarfs      David Hand
1                        Pinocchio  Ben Sharpsteen
2                         Fantasia    full credits
3                            Dumbo  Ben Sharpsteen
4                            Bambi      David Hand
(56, 2)
name        object
director    object
dtype: object
                       movie_title release_date      genre MPAA_rating  \
0  Snow White and the Seven Dwarfs   1937-12-21    Musical           G
1                        Pinocchio   1940-02-09  Adventure           G
2                         Fantasia   1940-11-13    Musical           G
3                Song of the South   1946-11-12  Adventure           G
4                       Cinderella   1950-02-15      Drama           G

    total_gross inflation_adjusted_gross
0  $184,925,485           $5,228,953,251
1   $84,300,000           $2,188,229,052
2   $83,320,000           $2,187,090,808
3   $65,000,000           $1,078,510,579
4   $85,000,000             $920,608,730
(579, 6)
movie_title                         object
release_date                datetime64[ns]
genre                               object
MPAA_rating                         object
```

```
total_gross                         object
inflation_adjusted_gross            object
dtype: object
   Year  Studio Entertainment[NI 1]  Disney Consumer Products[NI 2]  \
0  1991                      2593.0                           724.0
1  1992                      3115.0                          1081.0
2  1993                      3673.4                          1415.1
3  1994                      4793.0                          1798.2
4  1995                      6001.5                          2150.0

   Disney Interactive[NI 3][Rev 1]  Walt Disney Parks and Resorts  \
0                              NaN                         2794.0
1                              NaN                         3306.0
2                              NaN                         3440.7
3                              NaN                         3463.6
4                              NaN                         3959.8

  Disney Media Networks  Total
0                   NaN   6111
1                   NaN   7502
2                   NaN   8529
3                   359  10414
4                   414  12525
(26, 7)
Year                                  int64
Studio Entertainment[NI 1]          float64
Disney Consumer Products[NI 2]      float64
Disney Interactive[NI 3][Rev 1]     float64
Walt Disney Parks and Resorts       float64
Disney Media Networks                object
Total                                 int64
dtype: object
        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
(935, 3)
character      object
voice-actor    object
movie          object
dtype: object
```

2.8 0. Cleaning up the data

`Movies` turns out to be the effective 'primary key' (borrowing from database terminology) or linchpin data points of the whole dataset - it is coded into every other file in the overall dataset.

Let's look at cleaning it up first. ### i. Movies' cleanup

[3]:
```
def cleanup_movies(movie_df,
                   title_string,
                   genre_string,
                   rating_string,
                   caption_string):
    try:
        # Unavailable to fill in the blanks
        movie_df[genre_string] = movie_df[genre_string].fillna("Unavailable")
        movie_df[rating_string] = movie_df[rating_string].fillna("Unavailable")

        # Strip whitespace and other gunk from edges of title field
        movie_df = movie_df.assign(movie_title=movie_df["movie_title"].str.
 ↪strip())

        # Convert total gross to a float, remove currency formatting
        movie_df['total_gross'] = movie_df['total_gross'].replace('[\$,]', '',␣
 ↪regex=True).astype(float)

        # Same for the grosses adjusted for inflation
        movie_df['inflation_adjusted_gross'] =␣
 ↪movie_df['inflation_adjusted_gross'].replace('[\$,]', '', regex=True).
 ↪astype(float)
        return movie_df

    except Exception as e:
        print("Error: ", e)
        raise ValueError("unable to fill movie N/A values") from e


movies = cleanup_movies(movies, 'movie_title', 'genre', 'MPAA_rating', 'Figure␣
 ↪0. Movies by revenue, 1991-2016')
movies
```

[3]:
```
                          movie_title release_date      genre MPAA_rating  \
0    Snow White and the Seven Dwarfs   1937-12-21    Musical           G
1                          Pinocchio   1940-02-09  Adventure           G
2                           Fantasia   1940-11-13    Musical           G
3                  Song of the South   1946-11-12  Adventure           G
4                         Cinderella   1950-02-15      Drama           G
..                                …          …        …         …
574        The Light Between Oceans   2016-09-02      Drama       PG-13
575                  Queen of Katwe   2016-09-23      Drama          PG
576                  Doctor Strange   2016-11-04  Adventure       PG-13
577                           Moana   2016-11-23  Adventure          PG
578     Rogue One: A Star Wars Story   2016-12-16  Adventure       PG-13
```

```
     total_gross  inflation_adjusted_gross
0    184925485.0              5.228953e+09
1     84300000.0              2.188229e+09
2     83320000.0              2.187091e+09
3     65000000.0              1.078511e+09
4     85000000.0              9.206087e+08
..           …                         …
574   12545979.0              1.254598e+07
575    8874389.0              8.874389e+06
576  232532923.0              2.325329e+08
577  246082029.0              2.460820e+08
578  529483936.0              5.294839e+08

[579 rows x 6 columns]
```

2.8.1 ii. Test for 'n/a' value substitution

[4]:
```
# simple test to see if the repopulation worked in the genre column

test_of_changed = movies[movies["genre"] == "Unavailable"]
test_of_changed
```

[4]:
```
                                  movie_title release_date        genre  \
20   The Many Adventures of Winnie the Pooh   1977-03-11  Unavailable
22                 Herbie Goes to Monte Carlo   1977-06-24  Unavailable
23                             The Black Hole   1979-12-21  Unavailable
24                           Midnight Madness   1980-02-08  Unavailable
25                 The Last Flight of Noah's Ark   1980-06-25  Unavailable
26                    The Devil and Max Devlin   1981-01-01  Unavailable
121                                   Newsies   1992-04-08  Unavailable
122                              Passed Away   1992-04-24  Unavailable
128                 A Gun in Betty Lou's Handbag   1992-08-21  Unavailable
146                            Bound by Honor   1993-04-16  Unavailable
155                        My Boyfriend's Back   1993-08-06  Unavailable
156                               Father Hood   1993-08-27  Unavailable
168                            Red Rock West   1994-01-28  Unavailable
251                           The War at Home   1996-11-20  Unavailable
304                                 Endurance   1999-05-14  Unavailable
350                   High Heels and Low Lifes   2001-10-26  Unavailable
355                        Frank McKlusky C.I.   2002-01-01  Unavailable

     MPAA_rating  total_gross  inflation_adjusted_gross
20   Unavailable          0.0                       0.0
22   Unavailable   28000000.0               105847527.0
23   Unavailable   35841901.0               120377374.0
24   Unavailable    2900000.0                 9088096.0
```

25	Unavailable	11000000.0	34472116.0
26	Unavailable	16000000.0	48517980.0
121	PG	2706352.0	5497481.0
122	PG-13	4030793.0	8187848.0
128	PG-13	3591460.0	7295423.0
146	R	4496583.0	9156084.0
155	PG-13	3218882.0	6554384.0
156	PG-13	3268203.0	6654819.0
168	R	2502551.0	5170709.0
251	R	34368.0	65543.0
304	PG	229128.0	380218.0
350	R	226792.0	337782.0
355	Unavailable	0.0	0.0

2.8.2 iii. Business unit cleanup:

Decided against this and skipped cleanup due to business units overall not seeming relevant at this stage.

2.8.3 iv. Director cleanup

This CSV has directors listed for every entry, but lists 'full credits' where more than one is present. This file looks clean for the time being.

2.8.4 v. Voice Actor cleanup

One column needing work is the voice-actor column. While it has no blank N/A entries, it sometimes includes multiple voice actors for the same part. The file delimits these with a semi-colon. There are also a few entries where no voice-actor is listed. I will remove these.

[5]:
```
def cleanup_voice_actors(voice_actor_df):
    try:
        # Multiple VAs are listed in some cases - turn this off for now
        # voice_actor_df["voice-actor"] = voice_actor_df["voice-actor"].str.
 ↪split(";")

        voice_actor_df = voice_actor_df[voice_actor_df["voice-actor"] != 'None']
        return voice_actor_df

    except Exception as e:
        print("Error:", e)
        raise ValueError("Unable to cleanup voice actor dataframe") from e

voice_actors = cleanup_voice_actors(voice_actors)
voice_actors
```

[5]:

	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
5               Adella        Sherry Lynn             The Little Mermaid
..                 …                …                               …
930               Zeus           Rip Torn                       Hercules
931  Ziggy the Vulture        Digby Wolfe                The Jungle Book
932               Zini        Max Casella                       Dinosaur
933             Zipper       Corey Burton      Chip 'n Dale Rescue Rangers
934               Zira  Suzanne Pleshette  The Lion King II: Simba's Pride

[882 rows x 3 columns]
```

Comments: We have stripped about 48 entries from this table, presumably from filtering out 'None' where no one in the film was a voice actor.

2.8.5 vi. Characters cleanup:

Stripped whitespace characters, subbed in 'unknown' for blank cells in the datafile. While the 'villian' header [sic] is misspelled, I decided against renaming it for the time being.

[6]:
```python
def cleanup_characters(characters_df):
    try:
        # Unavailable to fill in the blanks
        characters_df["hero"] = characters_df["hero"].fillna("unknown")
        characters_df["villian"] = characters_df["villian"].fillna("unknown")
        characters_df["song"] = characters_df["song"].fillna("unknown")

        # Strip whitespace and other gunk from edges of title field
        characters_df = characters_df.assign(
            movie_title=characters_df["movie_title"].str.strip()
        )
        characters_df = characters_df.assign(hero=characters_df["hero"].str.
 ↪strip())

        # Take out entries where there isn't a hero
        # characters_df = characters_df[characters_df['hero'] != 'None']

        characters_df = characters_df.assign(
            villian=characters_df["villian"].str.strip()
        )
        # Decided not to rename the header for the time being to keep the data␣
 ↪as close to source as possible.
        # characters_df = characters_df.rename(columns={"villian": "villain"})
        characters_df = characters_df.assign(song=characters_df["song"].str.
 ↪strip())
```

```
        return characters_df
    except Exception as e:
        print("Error ")
        ValueError("unable to complete cleanup, check characters table")

# Decided against this for the time being
def split_multiple_characters(characters_df):
    try:
        # Split the 'hero' and 'villain' columns at ' and '
        # (Note spaces surrounding the word "and")
        characters_df["hero"] = characters_df["hero"].str.split(" and ")
        characters_df["villian"] = characters_df["villian"].str.split(" and ")
        return characters_df

    except Exception as e:
        print("Error:", e)
        raise ValueError("Unable to split field in two") from e

characters = cleanup_characters(characters)
# characters = split_multiple_characters(characters)
characters
```

[6]:

```
                                 movie_title release_date  \
0          Snow White and the Seven Dwarfs   1937-12-21
1                                Pinocchio   1940-02-07
2                                 Fantasia   1940-11-13
3                                    Dumbo   1941-10-23
4                                    Bambi   1942-08-13
5                           Saludos Amigos   1943-02-06
6                     The Three Caballeros   1945-02-03
7                          Make Mine Music   1946-04-20
8                      Fun and Fancy Free   1947-09-27
9                              Melody Time   1948-05-27
10  The Adventures of Ichabod and Mr. Toad   1949-10-05
11                              Cinderella   1950-02-15
12                    Alice in Wonderland   1951-07-28
13                               Peter Pan   1953-02-05
14                     Lady and the Tramp   1955-06-22
15                          Sleeping Beauty   1959-01-29
16          One Hundred and One Dalmatians   1961-01-25
17                  The Sword in the Stone   1963-12-25
18                         The Jungle Book   1967-10-18
19                          The Aristocats   1970-12-24
20                              Robin Hood   1973-11-08
21  The Many Adventures of Winnie the Pooh   1977-03-11
22                            The Rescuers   1977-06-22
23                    The Fox and the Hound   1981-07-10
```

```
24                The Black Cauldron   1985-07-24
25         The Great Mouse Detective   1986-07-02
26                  Oliver & Company   1988-11-18
27                 The Little Mermaid   1989-11-17
28           The Rescuers Down Under   1990-11-16
29              Beauty and the Beast   1991-11-22
30                           Aladdin   1992-11-25
31                     The Lion King   1994-06-24
32                        Pocahontas   1995-06-23
33       The Hunchback of Notre Dame   1996-06-21
34                          Hercules   1997-06-27
35                             Mulan   1998-06-19
36                            Tarzan   1999-06-18
37                     Fantasia 2000   1999-12-17
38                          Dinosaur   2000-05-19
39          The Emperor's New Groove   2000-12-15
40         Atlantis: The Lost Empire   2001-06-15
41                     Lilo & Stitch   2002-06-21
42                   Treasure Planet   2002-11-27
43                       Brother Bear   2003-11-01
44                 Home on the Range   2004-04-02
45                    Chicken Little   2005-11-04
46                Meet the Robinsons   2007-03-30
47                              Bolt   2008-11-21
48         The Princess and the Frog   2009-12-11
49                           Tangled   2010-11-24
50                  Winnie the Pooh   2011-07-15
51                    Wreck-It Ralph   2012-11-02
52                            Frozen   2013-11-27
53                        Big Hero 6   2014-11-07
54                         Zootopia   2016-03-04
55                             Moana   2016-11-23

                         hero                               villian  \
0                  Snow White                            Evil Queen
1                   Pinocchio                             Stromboli
2                     unknown                            Chernabog
3                       Dumbo                            Ringmaster
4                       Bambi                                Hunter
5                 Donald Duck                               unknown
6                 Donald Duck                               unknown
7                     unknown                               unknown
8                Mickey Mouse                      Willie the Giant
9                     unknown                               unknown
10  Mr. Toad and Ichabod Crane  Mr. Winkie and The Headless Horseman
11                 Cinderella                         Lady Tremaine
12                      Alice                        Queen of Hearts
```

13	Peter Pan	Captain Hook
14	Lady and Tramp	Si and Am
15	Aurora	Maleficent
16	Pongo	Cruella de Vil
17	Arthur	Madam Mim
18	Mowgli	Kaa and Shere Khan
19	Thomas and Duchess	Edgar Balthazar
20	Robin Hood	Prince John
21	Winnie the Pooh	unknown
22	Bernard and Miss Bianca	Madame Medusa
23	Tod and Copper	Amos Slade
24	Taran	Horned King
25	Basil	Professor Ratigan
26	Oliver	Sykes
27	Ariel	Ursula
28	Bernard and Miss Bianca	Percival C. McLeach
29	Belle	Gaston
30	Aladdin	Jafar
31	Simba	Scar
32	Pocahontas	Governor Ratcliffe
33	Quasimodo	Claude Frollo
34	Hercules	Hades
35	Mulan	Shan Yu
36	Tarzan	Clayton
37	unknown	unknown
38	Aladar	Kron
39	Kuzco	Yzma
40	Milo Thatch	Commander Rourke
41	Lilo and Stitch	unknown
42	Jim Hawkins	John Silver
43	Kenai	Denahi
44	Maggie	Alameda Slim
45	Ace Cluck	Foxy Loxy
46	Lewis	Doris
47	Bolt	Dr. Calico
48	Tiana	Dr. Facilier
49	Rapunzel	Mother Gothel
50	Winnie the Pooh	unknown
51	Ralph	Turbo
52	Elsa	Prince Hans
53	Hiro Hamada	Professor Callaghan
54	Judy Hopps	unknown
55	Moana	unknown

	song
0	Some Day My Prince Will Come
1	When You Wish upon a Star

2	unknown
3	Baby Mine
4	Love Is a Song
5	Saludos Amigos
6	unknown
7	unknown
8	unknown
9	Little Toot
10	The Merrily Song
11	Bibbidi-Bobbidi-Boo
12	The Unbirthday Song
13	You Can Fly!
14	Bella Notte
15	Once Upon a Dream
16	Cruella De Vil
17	Higitus Figitus
18	The Bare Necessities
19	Ev'rybody Wants to Be a Cat
20	Oo De Lally
21	Winnie the Pooh
22	The Journey
23	Best of Friends
24	unknown
25	The World's Greatest Criminal Mind
26	Once Upon a Time in New York City
27	Under the Sea
28	
29	Be Our Guest
30	A Whole New World
31	Circle of Life
32	Colors of the Wind
33	God Help the Outcasts
34	Go the Distance
35	I'll Make a Man Out of You
36	You'll Be in My Heart
37	unknown
38	unknown
39	My Funny Friend and Me
40	Where the Dream Takes You
41	He Mele No Lilo
42	I'm Still Here
43	Look Through My Eyes
44	unknown
45	unknown
46	Little Wonders
47	I Thought I Lost You
48	Almost There

```
49                        I See the Light
50                        Winnie the Pooh
51                             Sugar Rush
52                              Let It Go
53                              Immortals
54                         Try Everything
55                        How Far I'll Go
```

Comments: I have decided against splitting these heroes and villains where two are listed in each category, in the interest of time.

2.9 0b. Reusable tools and unit testing

2.9.1 i. Building a reusable function for merging dataframes, with docstring

See `reusable_disney_processing.py` for `merge_with_movies` details - we'll print the docstring for the method below.

[7]:
```
?merge_with_movies
```

```
Signature:
merge_with_movies(
    movie_df,
    right_df,
    movie_key_col,
    right_key_col,
    col_of_interest,
    col_revenue,
    shouldLimit=False,
    top_num=None,
)
Docstring:
Merges the movie_df with a right_hand_df

Parameters
    movie_df - Object, Pandas Left-hand (LH) side dataframe
    right_df - Object, Pandas Right-hand (RH) side dataframe
    movie_key_col - String, LH primary key, needs to be matched against the next␣
 ↪parameter below
    right_key_col - String, RH primary key
    col_of_interest - String, The column we're really interested in
    col_revenue - Float, total gross revenue, or adj for inflation
    shouldLimit - Optional bool, when true you should enter a number below
    top_n - Optional int, number to restrict output against

Raises
    Exception as e
    ValueError via Exception as e
```

```
Returns
    A merged DataFrame, which then can be plugged into Altair

Examples
    directors_analysis = merge_with_movies(
        movies,
        directors,
        'movie_title',
        'name',
        'director',
        'total_gross'
    )
    directors_analysis <Pandas.DataFrame>
File:      ~/prog-python-ds-students/release/final_project/
 ↪reusable_disney_processing.py
Type:      function
```

2.9.2 ii. Unit testing the merge movies method

I learned iOS programming in Swift & Objective-C a number of years ago, and learned to use the Given / When / Then (`GWT`) comment structure when building unit tests at work. I will use that mnemonic here in my tests.

Alternatively data scientists and developers can use the Arrange / Act / Assert (`AAA`) comment structure mnemonic. In essence, GWT focuses on the scenario to be tested, while AAA puts its efforts into method behavior.

However, the working differences between these are minimal in practice, and can be effectively swapped as the user prefers.

As the tests do not have docstrings, we won't be able to glean any useful info from them indirectly. Please consult `tests_reusable_disney_processing.py` to inspect the four tests directly.

Test 1: Basic merge functionality tests

[8]:
```
# Run test 1
test_merge_with_movies_basic()
```

```
Test 1: Basic merge functionality testing - PASSED
```

Test 2: Limiting results

[9]:
```
# Run test 2
test_merge_with_movies_limit()
```

```
Test 2: Limiting results - PASSED
```

2.9.3 ii. Reusable function for the simple Altair plot

See `reusable_disney_processing.py` for `simple_plot_from` details - we'll print the docstring for the method below.

[10]:
```
?simple_plot_from
```

```
Signature:
simple_plot_from(
    input_df,
    col_of_interest,
    class_letter,
    revenue_type,
    interest_title=None,
    revenue_title=None,
    plot_title=None,
    sort='y',
    top_n=None,
)
Docstring:
Constructs a simple Altair plot from defined inputs

Parameters:
    input_df - input dataframe
    col_of_interest - String, what we're looking to investigate
    class_letter - String, classification
    revenue_type - gross or adj revenues
    interest_title=None - optional String
    revenue_title=None - optional String
    plot_title=None - optional String
    sort="y" - optional String
    top_n=None - optional Int

Raises:
    Exception as e
    ValueError from within the exception

Returns
    Simple ranked Altair bar chart graph

Examples
    cinematographers_plotted = simple_plot_from(
        cinematographers_analysed,
        'cinematographer',
        'N',
        "total_gross",
        "Cinematographers",
        "Total gross revenue",
```

```
        "Top cinematographers by revenue")

    cinematographers_plotted <Altair chart>
File:      ~/prog-python-ds-students/release/final_project/
↪reusable_disney_processing.py
Type:      function
```

Test 3: Generating a basic plot

[11]:
```
# Run test 3
test_simple_plot_from_basic()
```

```
Test 3: Basic plotting is working - PASSED
```

Test 4: Custom sort and title

[12]:
```
# Run test 4
test_simple_plot_from_sort_and_title()
```

```
Test 4: Custom sort and title - PASSED
```

2.9.4 iii. Reusable functions for finding and plotting specific persons (real or fictional) against the Adventure genre of Disney films.

[13]:
```
?analyze_by_genre
```

```
Signature:
analyze_by_genre(
    movie_df,
    right_df,
    movie_key_col,
    right_key_col,
    genre_col,
    col_of_interest,
    col_revenue,
)
Docstring:
Analyzes the profitability of people (ie directors) by movie genre.

Parameters:
    movie_df - DataFrame containing movie data
    right_df - DataFrame containing data of the people (e.g., directors)
    movie_key_col - Column in movie_df to merge on (e.g., 'movie_title')
    right_key_col - Column in right_df to merge on (e.g., 'name')
    genre_col - Column in movie_df representing the genre
    col_of_interest - Column in right_df representing the person of interest (e.
↪g., 'director')
```

```
    col_revenue - Column in movie_df representing the revenue (e.g.,␣
↪'total_gross')

Raises:
    ValueError via an Exception

Returns:
    DataFrame with each genre and the total gross revenue for each person in␣
↪that genre
File:      ~/prog-python-ds-students/release/final_project/
↪reusable_disney_processing.py
Type:      function
```

[14]:

```
?find_top_n_genre_people
```

```
Signature:
find_top_n_genre_people(
    movie_df,
    right_df,
    movie_key_col,
    right_key_col,
    genre_col,
    col_of_interest,
    col_revenue,
    genre_of_interest,
    top_n,
)
Docstring:
Picks out the top_n [personnel] (your choice) by genre (also your choice)

Parameters
    movie_df - DataFrame containing movie data
    right_df - DataFrame containing data of the people (e.g., directors)
    movie_key_col - Column in movie_df to merge on (e.g., 'movie_title')
    right_key_col - Column in right_df to merge on (e.g., 'name')
    genre_col - Column in movie_df representing the genre (NOT 'Adventure', but␣
↪'genre')
    col_of_interest - Column in right_df representing the person of interest (e.
↪g., 'director')
    col_revenue - Column in movie_df representing the revenue (e.g.,␣
↪'total_gross')
    genre_of_interest - The film category we want to explore
    top_n - Fill with the number of highest-performing people in this genre you␣
↪want to see

Results
```

```
    Returns a small number (n) of top performers by your chosen genre

Raises
    Exception on error
    Raises ValueError from that Exception

Examples
    adventure_directors_test = find_top_n_genre_people(
        movies,
        directors,
        'movie_title',
        'name',
        'genre',
        'director',
        'total_gross',
        'Adventure',
        7
    )
    <Returns a Pandas dataframe suitable for plotting in Altair.>
File:      ~/prog-python-ds-students/release/final_project/
↪reusable_disney_processing.py
Type:      function
```

[15]:
```
?side_plot_for_genre_people
```

```
Signature:
side_plot_for_genre_people(
    df_to_load,
    x_col,
    x_letter,
    x_title,
    y_col,
    y_letter,
    y_title,
    plot_title,
)
Docstring:
Outputs an Altair horizontal plot of personnel by genre in dollar terms

Parameters
    df_to_load - the dataframe we want to analyse (see find_top_n_genre_people)
    x_col - the horizontal axis to plot (this is the y-axis in our other plot␣
↪method)
    x_letter - Ordinality, etc for units of measure along the x-axis
    x_title - The display title for the x-axis
    y_col - the vertical axis we want
    y_letter - Ordinality, etc for the x-axis
```

```
    y_title - The display title for the y-axis
    plot_title - Overall title for the plot at large

Results
    Returns a horizontally-aligned plot for display

Raises
    General exception is created on error
    ValueError is raised from this general exception

Examples
    test_df # Created by find_top_n_genre_people, use its parameters here

    side_plot = side_plot_for_genre_people(
        test_df,
        x_col,
        x_letter,
        x_title,
        y_col,
        y_letter,
        y_title,
        plot_title
    )
File:      ~/prog-python-ds-students/release/final_project/
↪reusable_disney_processing.py
Type:      function
```

2.10 Ia. Movies analysis

2.10.1 1. Movies by revenue, table

[16]:
```python
def movies_top_n_analysis(movie_df, column_to_rank_by, top_n):
    movies_for_analysis = movie_df
    movies_for_analysis = movies_for_analysis.sort_values(by=column_to_rank_by,␣
↪ascending=False).head(top_n)
    return movies_for_analysis

# Sort the dataframe by the total_gross column in descending order and take the␣
↪top 20
top_20_movies_analysed = movies_top_n_analysis(movies, 'total_gross', 20)
top_20_movies_analysed
```

[16]:
```
                              movie_title release_date          genre  \
564    Star Wars Ep. VII: The Force Awakens   2015-12-18      Adventure
524                            The Avengers   2012-05-04         Action
578            Rogue One: A Star Wars Story   2016-12-16      Adventure
571                            Finding Dory   2016-06-17      Adventure
```

```
558                 Avengers: Age of Ultron   2015-05-01             Action
441    Pirates of the Caribbean: Dead Man’…   2006-07-07          Adventure
179                           The Lion King   1994-06-15          Adventure
499                             Toy Story 3   2010-06-18          Adventure
532                              Iron Man 3   2013-05-03             Action
569              Captain America: Civil War   2016-05-06             Action
539                                  Frozen   2013-11-22          Adventure
384                            Finding Nemo   2003-05-30          Adventure
567                         The Jungle Book   2016-04-15          Adventure
560                              Inside Out   2015-06-19          Adventure
566                                Zootopia   2016-03-04          Adventure
494                     Alice in Wonderland   2010-03-05          Adventure
549                 Guardians of the Galaxy   2014-08-01          Adventure
457    Pirates of the Caribbean: At World’…   2007-05-24          Adventure
385  Pirates of the Caribbean: The Curse o…   2003-07-09          Adventure
309                         The Sixth Sense   1999-08-06  Thriller/Suspense

    MPAA_rating  total_gross  inflation_adjusted_gross
564       PG-13  936662225.0               936662225.0
524       PG-13  623279547.0               660081224.0
578       PG-13  529483936.0               529483936.0
571          PG  486295561.0               486295561.0
558       PG-13  459005868.0               459005868.0
441       PG-13  423315812.0               544817142.0
179           G  422780140.0               761640898.0
499           G  415004880.0               443408255.0
532       PG-13  408992272.0               424084233.0
569       PG-13  408084349.0               408084349.0
539          PG  400738009.0               414997174.0
384           G  380529370.0               518148559.0
567          PG  364001123.0               364001123.0
560          PG  356461711.0               356461711.0
566          PG  341268248.0               341268248.0
494          PG  334191110.0               357063499.0
549       PG-13  333172112.0               343771168.0
457       PG-13  309420425.0               379129960.0
385       PG-13  305411224.0               426967926.0
309       PG-13  293506292.0               485424724.0
```

Comments: I limited this table to the top 20 movies for summary purposes only.

2.10.2 2. Movies by revenue, plotted

[17]:
```
movies_plotted = alt.Chart(top_20_movies_analysed).mark_bar().encode(
    x=alt.X('movie_title:N', title='Movie', sort='y'),
    y=alt.Y('total_gross:Q', title="Gross revenue, in dollars")
).properties(
```

```
    title='Top 20 movies by revenue in dollars, 1991-1996'
)
movies_plotted
```

[17]: alt.Chart(…)

2.10.3 3. Top movies by inflation-adjusted gross

[18]:
```
top_20_movies_inflation_adjusted = movies_top_n_analysis(movies,␣
↪'inflation_adjusted_gross', 20)
top_20_movies_inflation_adjusted
```

[18]:
```
                                movie_title release_date              genre  \
0           Snow White and the Seven Dwarfs   1937-12-21            Musical
1                                 Pinocchio   1940-02-09          Adventure
2                                  Fantasia   1940-11-13            Musical
8                            101 Dalmatians   1961-01-25             Comedy
6                        Lady and the Tramp   1955-06-22              Drama
3                         Song of the South   1946-11-12          Adventure
564  Star Wars Ep. VII: The Force Awakens   2015-12-18          Adventure
4                                Cinderella   1950-02-15              Drama
13                          The Jungle Book   1967-10-18            Musical
179                           The Lion King   1994-06-15          Adventure
524                            The Avengers   2012-05-04             Action
441  Pirates of the Caribbean: Dead Man’…   2006-07-07          Adventure
578          Rogue One: A Star Wars Story   2016-12-16          Adventure
5            20,000 Leagues Under the Sea   1954-12-23          Adventure
384                            Finding Nemo   2003-05-30          Adventure
571                            Finding Dory   2016-06-17          Adventure
309                         The Sixth Sense   1999-08-06  Thriller/Suspense
558                 Avengers: Age of Ultron   2015-05-01             Action
499                             Toy Story 3   2010-06-18          Adventure
135                                 Aladdin   1992-11-11             Comedy

     MPAA_rating  total_gross  inflation_adjusted_gross
0              G  184925485.0              5.228953e+09
1              G   84300000.0              2.188229e+09
2              G   83320000.0              2.187091e+09
8              G  153000000.0              1.362871e+09
6              G   93600000.0              1.236036e+09
3              G   65000000.0              1.078511e+09
564        PG-13  936662225.0              9.366622e+08
4              G   85000000.0              9.206087e+08
13     Not Rated  141843000.0              7.896123e+08
179            G  422780140.0              7.616409e+08
524        PG-13  623279547.0              6.600812e+08
441        PG-13  423315812.0              5.448171e+08
```

```
578         PG-13  529483936.0                5.294839e+08
5     Unavailable   28200000.0                5.282800e+08
384             G  380529370.0                5.181486e+08
571            PG  486295561.0                4.862956e+08
309         PG-13  293506292.0                4.854247e+08
558         PG-13  459005868.0                4.590059e+08
499             G  415004880.0                4.434083e+08
135             G  217350219.0                4.419692e+08
```

2.11 4. Top movies by inflation-adjusted gross, plotted

[19]:
```
movies_plotted_inflation = alt.Chart(top_20_movies_inflation_adjusted).
↪mark_bar().encode(
    x=alt.X('movie_title:N', title='Movie', sort='y'),
    y=alt.Y('inflation_adjusted_gross:Q', title="Revenue adjusted for␣
↪inflation, in 2016 dollars")
).properties(
    title='Movies by inflation-adjusted revenue in 2016 dollars, 1991-1996'
)
movies_plotted_inflation
```

[19]: alt.Chart(...)

Comments: The highest-grossing Disney movies during the time period were Snow White and the Seven Dwarfs, Pinocchio, Fantasia, 101 Dalmatians, and Lady and the Tramp. Note that several of these have had multiple show dates over the years, enabling multiple kicks at the revenue can.

2.12 Ib. Movie genres by revenue

2.12.1 1. Table

Now let's show the genre by revenue break down.

[23]:
```
def genre_x_revenue(movie_df, genre_column, revenue_column):
    # Group by 'genre' and sum up the 'total_gross' to get total revenue for␣
↪each genre
    genre_totals = movie_df.groupby(genre_column)[revenue_column].sum().
↪sort_values()

    # Display the genre totals from least to most revenue
    # - need to reset the index in order for this to work.
    genre_totals_df = genre_totals.reset_index(name=revenue_column)
    return genre_totals_df

genre_revenue = genre_x_revenue(movies, 'genre', 'total_gross')
genre_revenue
```

[23]:

	genre	total_gross
0	Horror	8.706887e+07
1	Black Comedy	9.754321e+07
2	Concert/Performance	1.034565e+08
3	Unavailable	1.180470e+08
4	Documentary	1.806856e+08
5	Western	3.590115e+08
6	Romantic Comedy	1.152207e+09
7	Musical	1.157284e+09
8	Thriller/Suspense	1.406807e+09
9	Drama	4.106973e+09
10	Action	4.184563e+09
11	Comedy	8.119620e+09
12	Adventure	1.638907e+10

2.12.2 2. Genre by revenue, plotted

[22]:
```
genres_plotted = alt.Chart(genre_revenue).mark_bar().encode(
    x=alt.X('genre:N', title='Movie', sort="y"),
    y=alt.Y('total_gross:Q', title='Revenue, in dollars')
).properties(
    title='Movies, 1991-2016, by genre and revenue (in dollars)'
)
genres_plotted
```

[22]: alt.Chart(…)

The most popular genres over the time period 1991-2016 are, in order, Adventure, Comedy, Action, Drama, and Thriller/Suspense.

2.12.3 3. Genre by inflation-adjusted revenue

[410]:
```
genre_revenue_inflation = genre_x_revenue(movies, 'genre',␣
↪'inflation_adjusted_gross')
genre_revenue_inflation
```

[410]:

	genre	inflation_adjusted_gross
0	Concert/Performance	1.148217e+08
1	Horror	1.404831e+08
2	Black Comedy	1.567305e+08
3	Documentary	2.034884e+08
4	Unavailable	3.676034e+08
5	Western	5.167099e+08
6	Romantic Comedy	1.788873e+09
7	Thriller/Suspense	2.151691e+09
8	Action	5.498937e+09
9	Drama	8.195804e+09

```
10          Musical         9.657566e+09
11           Comedy         1.540953e+10
12        Adventure         2.456127e+10
```

Comments: The most notable change from the unadjusted table is that Musicals over time have proven to be popular, launching into the overall 3rd spot.

2.12.4 4. Genre by inflation-adjusted revenue, plotted

[411]:
```
genre_x_revenue_inflation = alt.Chart(genre_revenue_inflation).mark_bar().
 ↪encode(
    x=alt.X('genre:N', title='Movie', sort="y"),
    y=alt.Y('inflation_adjusted_gross:Q', title="gross adjusted for inflation,␣
 ↪2016 dollars")
).properties(
    title="Movie genre, 1991-2016, by inflation-adjusted gross, 2016 dollars"
)
genre_x_revenue_inflation
```

[411]: alt.Chart(…)

The most popular genres over the time period 1991-2016 by inflation-adjusted gross (2016 dollars) are, in order, Adventure, Comedy, Musical (!!), Drama, and Action.

2.13 II. Directors' analysis

2.13.1 1. Top grossing directors, tabled

[412]:
```
directors_analysed = merge_with_movies(
    movies,
    directors,
    "movie_title",
    "name",
    "director",
    "total_gross"
)
directors_analysed
```

[412]:

	director	total_gross
22	Ted Berman	21288692.0
0	Art Stevens	43899231.0
11	George Scribner	49576671.0
24	Will Finn	50026353.0
26	full credits	83320000.0
2	Ben Sharpsteen	84300000.0
18	Robert Walker	85336277.0
12	Hamilton Luske	93600000.0

```
6        Chris Williams  114053759.0
1            Barry Cook  120620254.0
21  Stephen J. Anderson  124515017.0
16          Ralph Zondag  137748063.0
5         Chris Sanders  145771527.0
14         Mike Gabriel  169511234.0
8            David Hand  184925485.0
17           Rich Moore  189412677.0
15         Nathan Greno  200821936.0
9              Don Hall  222527828.0
13          Mark Dindal  224683238.0
23      Wilfred Jackson  286151353.0
3          Byron Howard  341268248.0
7        Clyde Geronimi  343655718.0
10      Gary Trousdale  403143238.0
19        Roger Allers  422780140.0
4            Chris Buck  571829828.0
20         Ron Clements  840214815.0
25  Wolfgang Reitherman  966009582.0
```

Comments: Wolfgang Reitherman, Ron Clements, Chris Buck, Roger Allers and Gary Trousdale are the top 5 Disney directors from 1991-2016.

2.13.2 2. Top grossing directors, plotted

[413]:
```
directors_plotted = simple_plot_from(
    directors_analysed,
    'director',
    'N',
    "total_gross",
    "Directors",
    "Total gross revenue",
    "Top directors by revenue")

directors_plotted
```

[413]:
```
alt.Chart(…)
```

Comments: The top 5 directors taken together account for almost $5bn in Disney revenue over the years.

2.13.3 3. Top grossing directors, by inflation-adjusted gross

[414]:
```
directors_analysed_inflation = merge_with_movies(
    movies,
    directors,
```

```
    "movie_title",
    "name",
    "director",
    "inflation_adjusted_gross"
)
directors_analysed_inflation
```

[414]:

```
               director  inflation_adjusted_gross
22           Ted Berman              5.055314e+07
24            Will Finn              6.791017e+07
11      George Scribner              1.022545e+08
18       Robert Walker               1.192183e+08
0           Art Stevens              1.331189e+08
6        Chris Williams              1.337025e+08
21  Stephen J. Anderson              1.482365e+08
17           Rich Moore              2.003550e+08
5         Chris Sanders              2.115067e+08
15         Nathan Greno              2.143885e+08
16         Ralph Zondag              2.154390e+08
1            Barry Cook              2.168078e+08
9              Don Hall              2.292492e+08
13          Mark Dindal              3.147439e+08
14         Mike Gabriel              3.301677e+08
3          Byron Howard              3.412682e+08
7        Clyde Geronimi              3.785693e+08
10       Gary Trousdale              6.791946e+08
4            Chris Buck              6.988974e+08
19         Roger Allers              7.616409e+08
23      Wilfred Jackson              1.121760e+09
12       Hamilton Luske              1.236036e+09
20         Ron Clements              1.318950e+09
26         full credits              2.187091e+09
2        Ben Sharpsteen              2.188229e+09
25  Wolfgang Reitherman              3.432920e+09
8            David Hand              5.228953e+09
```

Comments: Changing over to inflation-adjusted figures means that David Hand shoots to the top of the directors' pile. Notably, multiple directors collectively take the number 4 spot in this adjusted table.

2.13.4 4. Top directors by inflation-adjusted gross, 1991-2016, 2016 dollars, plotted

[415]:

```
director_plot_inflation = simple_plot_from(
    directors_analysed_inflation,
    'director',
    "N",
    "inflation_adjusted_gross",
```

```
    "Director",
    "Inflation-adjusted grosses",
    "Revenue! (Inflation-adj'd), by director, 1991!2016")

director_plot_inflation
```

[415]: alt.Chart(…)

2.13.5 5. Top 5 directors in the Adventure genre, tabled

Let's use our newest methods to drill down into a comparison of Directors that have worked on genre films in the Adventure category.

First, we'll work out who the top 5 directors are in adventure, and show them in a table.

[416]:
```
adventure_directors_test = find_top_n_genre_people(
    movies,
    directors,
    'movie_title',
    'name',
    'genre',
    'director',
    'total_gross',
    'Adventure',
    5
)

adventure_directors_test
```

[416]:

	genre	director	total_gross
16	Adventure	Ron Clements	622864596.0
3	Adventure	Chris Buck	571829828.0
19	Adventure	Wolfgang Reitherman	479302031.0
15	Adventure	Roger Allers	422780140.0
2	Adventure	Byron Howard	341268248.0

Comments: Byron Howard makes an appearance in the tabled stats when we focus on the Adventure genre.

2.13.6 6. Top 5 directors in the Adventure genre, plotted

Next, we'll plot these.

[417]:
```
top5_directors_in_adventure = side_plot_for_genre_people(
    adventure_directors_test,
    'total_gross',
    'Q',
    'Total Gross Revenue, $',
```

```
    'director',
    'N',
    'Director',
    'Top 5 Directors in Adventure by Gross Revenue'
)

top5_directors_in_adventure
```

[417]: alt.Chart(…)

Comments: Ron Clements appears to exceed his next competing director, Chris Buck, by over $100m.

2.14 III. Voice actors' analysis

2.14.1 1. Voice actors by revenue, tabled (top 20)

[418]:
```
voices_analysed = merge_with_movies(
    movies,
    voice_actors,
    "movie_title",
    "movie",
    "voice-actor",
    "total_gross",
    True,
    20
)
voices_analysed
```

[418]:

	voice-actor	total_gross
432	Taylor Holmes	9464608.0
301	Mary Costa	9464608.0
122	Eleanor Audley	9464608.0
24	Barbara Dirikson	9464608.0
33	Bill Shirley	9464608.0
25	Barbara Jo Allen	9464608.0
160	Haley Joel Osment	16988996.0
254	Justin Berfield	16988996.0
181	James Gammon	16988996.0
83	Dal McKennon	17871174.0
386	Robert Holt	17871174.0
270	Kyle Stanger	18098433.0
118	Eda Reiss Merin	21288692.0
138	Freddie Jones	21288692.0
235	John Hurt	21288692.0
19	Arthur Malet	21288692.0
154	Grant Bardsley	21288692.0

```
35        Billie Hayes   21288692.0
3     Adele Malis-Morey   21288692.0
426     Susan Sheridan   21288692.0
```

Comments: Susan Sheridan was known to English audiences in the UK as the voice of Noddy, a beloved children's character.

2.14.2 2. Voice actors by revenue, plotted

[419]:
```
voice_plotttt = simple_plot_from(voices_analysed,
                                  'voice-actor',
                                  "N",
                                  "total_gross",
                                  "Voice actor",
                                  "Revenue, total gross",
                                  "Voice actors by revenue"
                                 )
voice_plotttt
```

[419]:
```
alt.Chart(…)
```

Comments: The top eight voice actors, including the legendary John Hurt, all figured in films worth $22m each over the years.

2.14.3 3. Voice actors by inflation-adjusted revenue

[420]:
```
voices_analysed_inflation = merge_with_movies(
    movies,
    voice_actors,
    "movie_title",
    "movie",
    "voice-actor",
    "inflation_adjusted_gross",
    True,
    20
)
voices_analysed_inflation
```

[420]:

	voice-actor	inflation_adjusted_gross
33	Bill Shirley	21505832.0
432	Taylor Holmes	21505832.0
301	Mary Costa	21505832.0
25	Barbara Jo Allen	21505832.0
24	Barbara Dirikson	21505832.0
122	Eleanor Audley	21505832.0
270	Kyle Stanger	23801835.0

```
160    Haley Joel Osment                24650121.0
254       Justin Berfield               24650121.0
181          James Gammon               24650121.0
138         Freddie Jones               50553142.0
35           Billie Hayes               50553142.0
426        Susan Sheridan               50553142.0
235             John Hurt               50553142.0
19           Arthur Malet               50553142.0
118       Eda Reiss Merin               50553142.0
362        Phil Fondacaro               50553142.0
3       Adele Malis-Morey               50553142.0
154        Grant Bardsley               50553142.0
427   Susanne Pollatschek               53637367.0
```

2.14.4 4. Voice actors by inflation-adjusted revenue, plotted

[421]:
```
voices_analysed_infl_plot = simple_plot_from(
    voices_analysed_inflation,
    'voice-actor',
    "N",
    "inflation_adjusted_gross",
    "Voice actor",
    "Revenue in inflation-adjusted dollars"
    "Voice actors by inflation-adjusted revenue, 1991-2016"
)
voices_analysed_infl_plot
```

[421]: alt.Chart(…)

Comments: Susanne Pollatschek voiced 'Olivia Flaversham' in the 1986 film The Great Mouse Detective at 8 years of age, and catapulted to the top of our inflation-adjusted grosses.

2.14.5 5. Top voice actors in Adventure genre, table

[422]:
```
voice_adventures = find_top_n_genre_people(
    movies,
    voice_actors,
    "movie_title",
    "movie",
    "genre",
    "voice-actor",
    "total_gross",
    "Adventure",
    5
)
voice_adventures
```

[422]:

	genre	voice-actor	total_gross
130	Adventure	J. Pat O'Malley	1.819261e+09
5	Adventure	Alan Tudyk	1.177501e+09
178	Adventure	John DiMaggio	1.023805e+09
333	Adventure	Verna Felton	7.425352e+08
103	Adventure	Frank Welker	6.777765e+08

Comments: Notable for the presence of Alan Tudyk, a sci-fi actor, and John DiMaggio, voice of Bender on Futurama.

2.14.6 6. Top voice actors in Adventure, plot

[423]:
```
voice_adventure_plot = side_plot_for_genre_people(
    voice_adventures,
    'total_gross',
    'Q',
    'Total Gross Revenue, $',
    'voice-actor',
    'N',
    'Voice actor',
    'Top 5 voice actors in Adventure by Gross Revenue'
)
voice_adventure_plot
```

[423]: alt.Chart(…)

2.15 IVa. Heroes (Characters)

2.15.1 1. Heroes by revenue, tabled

[351]:
```
heroes_analysed = merge_with_movies(
    movies,
    characters,
    'movie_title',
    'movie_title',
    'hero',
    'total_gross',
    True,
    20
)
heroes_analysed
```

[351]:

	hero	total_gross
6	Aurora	9464608.0
35	Taran	21288692.0
5	Arthur	22182353.0
7	Basil	23605534.0

```
40          Winnie the Pooh   26692846.0
15              Jim Hawkins   38120554.0
39           Tod and Copper   43899231.0
27                   Oliver   49576671.0
22                   Maggie   50026353.0
37       Thomas and Duchess   55675257.0
9   Bernard and Miss Bianca   76707060.0
41                  unknown   83320000.0
23              Milo Thatch   84052762.0
28                Pinocchio   84300000.0
17                    Kenai   85336277.0
18                    Kuzco   89296573.0
19           Lady and Tramp   93600000.0
20                    Lewis   97822171.0
13                 Hercules   99112101.0
30                Quasimodo  100138851.0
```

Comments: Iconic heroes such as Quasimodo and Hercules top out our heroes' table.

2.15.2 2. Heroes by revenue, plotted

[352]:
```
heroes_plotted = simple_plot_from(
    heroes_analysed,
    'hero',
    'N',
    "total_gross",
    "Hero (character)",
    "Total gross",
    "Revenue by hero (character)"
)

heroes_plotted
```

[352]:
```
alt.Chart(…)
```

Comments: The top 5 (6 as Lady and Tramp are considered as one here) characters are present in films grossing around $90m or above.

2.15.3 3. Inflation-adj revenue by hero, 1991-2016

[353]:
```
heroes_analysed_inflation = merge_with_movies(
    movies,
    characters,
    'movie_title',
    'movie_title',
    'hero',
```

```
    'inflation_adjusted_gross',
    True,
    20
)
heroes_analysed_inflation
```

[353]:

```
                hero  inflation_adjusted_gross
6             Aurora                21505832.0
40   Winnie the Pooh                28375869.0
35             Taran                50553142.0
7              Basil                53637367.0
15       Jim Hawkins                55189145.0
22            Maggie                67910166.0
27            Oliver               102254492.0
38             Tiana               116316457.0
17             Kenai               119218333.0
20             Lewis               119860589.0
23       Milo Thatch               125188122.0
39    Tod and Copper               133118889.0
10              Bolt               133702498.0
18             Kuzco               136789252.0
5             Arthur               153870834.0
0          Ace Cluck               177954661.0
13          Hercules               182029412.0
30         Quasimodo               190988799.0
31             Ralph               200354959.0
21   Lilo and Stitch               211506702.0
```

Comments: More modern 'hero' characters such as Lilo and Stitch and Ace Cluck make an appearance when figures are adjusted to inflation.

2.15.4 4. Heroes by inflation-adj rev, plot

[354]:

```
heroes_infl_plot = simple_plot_from(
    heroes_analysed_inflation,
    'hero',
    "N",
    "inflation_adjusted_gross",
    "Heroes",
    "Inflation-adjusted dollars",
    "Heroes by inflation-adj $ 1991-2016"
)

heroes_infl_plot
```

[354]: alt.Chart(…)

2.15.5 5. Top heroes in the Adventure genre, tabled

[430]:
```
hero_adventures = find_top_n_genre_people(
    movies,
    characters,
    "movie_title",
    "movie_title",
    "genre",
    "hero",
    "total_gross",
    "Adventure",
    5
)
hero_adventures
```

[430]:

	genre	hero	total_gross
26	Adventure	Simba	422780140.0
18	Adventure	Mowgli	408344079.0
7	Adventure	Elsa	400738009.0
11	Adventure	Judy Hopps	341268248.0
2	Adventure	Alice	334191110.0

Comments: Jungle Book characters shoot to the top of our Heroes breakdown in the Adventure genre.

2.15.6 6. Top heroes in the Adventure genre, plotted

[431]:
```
hero_adventures_plot = side_plot_for_genre_people(
    hero_adventures,
    'total_gross',
    'Q',
    'Total Gross Revenue, $',
    'hero',
    'N',
    'Hero',
    'Top 5 Heroes in Adventure by Gross Revenue'
)

hero_adventures_plot
```

[431]: alt.Chart(…)

Comments: Is it significant that the top 3 heroes in the Adventure genre are all animated characters?

2.16 IVb. Villain (characters) analysis

2.16.1 1. Villain analysed, table

[355]:
```
villains_analysed = merge_with_movies(
    movies,
    characters,
    'movie_title',
    'movie_title',
    'villian',
    'total_gross',
    True,
    20
)
villains_analysed
```

[355]:

	villian	total_gross
24	Maleficent	9464608.0
16	Horned King	21288692.0
22	Madam Mim	22182353.0
29	Professor Ratigan	23605534.0
26	Percival C. McLeach	27931461.0
18	John Silver	38120554.0
1	Amos Slade	43899231.0
23	Madame Medusa	48775599.0
35	Sykes	49576671.0
0	Alameda Slim	50026353.0
10	Edgar Balthazar	55675257.0
2	Chernabog	83320000.0
5	Commander Rourke	84052762.0
34	Stromboli	84300000.0
6	Denahi	85336277.0
38	Yzma	89296573.0
33	Si and Am	93600000.0
7	Doris	97822171.0
15	Hades	99112101.0
3	Claude Frollo	100138851.0

Comments: I'm not familiar with Disney villains in this genre, unfortunately. However the top 5 have all appeared in films grossing over $90m each.

2.16.2 2. Villains analysed, plotted

[357]:
```
villains_analysed_plot = simple_plot_from(
    villains_analysed,
    'villian',
    "N",
    'total_gross',
```

```
    "Villain!!",
    "Dollars grossed",
    "Villains plotted against revenue"
)
villains_analysed_plot
```

[357]: alt.Chart(…)

2.16.3 3. Villains vs revenue adjusted for inflation

[358]:
```
villains_adj_inflation = merge_with_movies(
    movies,
    characters,
    'movie_title',
    'movie_title',
    'villian',
    'inflation_adjusted_gross',
    True,
    20
)
villains_adj_inflation
```

[358]:

	villian	inflation_adjusted_gross
24	Maleficent	21505832.0
16	Horned King	50553142.0
29	Professor Ratigan	53637367.0
18	John Silver	55189145.0
26	Percival C. McLeach	55796728.0
0	Alameda Slim	67910166.0
35	Sykes	102254492.0
9	Dr. Facilier	116316457.0
6	Denahi	119218333.0
7	Doris	119860589.0
5	Commander Rourke	125188122.0
1	Amos Slade	133118889.0
8	Dr. Calico	133702498.0
38	Yzma	136789252.0
22	Madam Mim	153870834.0
23	Madame Medusa	159743914.0
12	Foxy Loxy	177954661.0
15	Hades	182029412.0
3	Claude Frollo	190988799.0
36	Turbo	200354959.0

2.16.4 4. Villains vs rev adj for inflation, plotted

[359]:
```
villains_adj_inflation_plotted = simple_plot_from(
    villains_adj_inflation,
    "villian",
    "N",
    "inflation_adjusted_gross",
    "Adj gross dollars",
    "Villains vs inflation-adj dollars"
)
villains_adj_inflation_plotted
```

[359]: alt.Chart(...)

Comments: Adjusting for inflation shoots these villains to the moon...

2.16.5 5. Top villains in the Adventure genre, tabled

[24]:
```
villain_adventures = find_top_n_genre_people(
    movies,
    characters,
    "movie_title",
    "movie_title",
    "genre",
    # sic
    "villian",
    "total_gross",
    "Adventure",
    5
)
villain_adventures
```

[24]:

	genre	villian	total_gross
28	Adventure	unknown	759814650.0
21	Adventure	Scar	422780140.0
11	Adventure	Kaa and Shere Khan	408344079.0
17	Adventure	Prince Hans	400738009.0
20	Adventure	Queen of Hearts	334191110.0

2.16.6 6. Top villains in the Adventure genre, plotted

[433]:
```
villain_adventures_plot = side_plot_for_genre_people(
    villain_adventures,
    'total_gross',
    'Q',
    'Total Gross Revenue, $',
    # sic
```

```
    'villian',
    'N',
    'Villain',
    'Top 5 Villains in Adventure by Gross Revenue'
)

villain_adventures_plot
```

[433]: alt.Chart(…)

Comments: I believe the 2nd through 5th entries in this adventure graph are all animated villains. I should have controlled for the large 'unknown' contingent, though.

2.17 V. Conclusions

The most profitable films at Disney from 1991-2016 demonstrate the company's wide range of popular titles, themes and genres. Across the board, Disney live-action and animated films are popular with audiences around the globe, and show the firm's longevity over time, with multiple and significant re-releases of popular films, particularly in children's-themed films in the categories of adventure, comedy, drama, action, and thriller/suspense films.

This trend of profitability is even more pronounced when you consider the numerous children's classics such as Fantasia and Snow White and the Seven Dwarfs, which have also been re-released to consumers and re-marketed in varying formats such as DVD and Blu-Ray, as well as more modern distribution channels such as Disney+.

These films show renewed interest and profitability, especially when one considers the inflation-adjusted numbers. In these cases, they are popular with audiences and the company alike, but perhaps for different reasons. For audiences, seeing old classics ensures continuity of experience and common touchstones that later get labelled as 'classics' or favourite perennials over time. For the firm, re-releasing old films makes enormous business sense as sunk costs beyond film stock / digital preservation are next to zero, and become engines of pure profit without any additional significant expense.

[ ]:

```
```