LGBTQIA+ Characters on Children's TV

Introduction

This report analyses the current state of LGBTQIA+ representation in children's media, specifically queer characters on kids TV shows, including their diversity and accessibility. First, some context around LGBTQIA+ portrayal is introduced, with reference to other reports. Then data acquisition is described with evaluation and summaries, followed by thorough analysis with reference to research questions. Lastly, the findings are related to previous research and the project's process is evaluated.

Based on this exploration, the inclusivity of children's programming has increased significantly over the last few years, as more diverse identities are brought on screen. However, the diversity of LGBTQIA+ characters themselves, regarding gender identity, disability representation and sexuality is not wast, with racial diversity being an exception. Lastly, most of inclusive children's TV is not widely accessible, since the platform the shows air on often require a subscription or an extra fee.

Research topic

Inclusive storytelling drives greater acceptance of the queer community and fights stigma, GLAAD (2021), an American non-government media monitoring organisation has found. Moreover, Craig et al. (2015) report that media serves as a catalyst for resilience of LGBTQIA+ youth, who disproportionately encounter discrimination and harassment, by "buffering discriminatory experiences". Yet, in 2021 only 9.1% of regular characters on US primetime scripted broadcast series are LGBTQIA+ (GLAAD, 2021), meaning the access to inclusive storylines is limited.

Queer attraction as the first milestone of sexual orientation identity development was found to be experienced on average at around the age of 12.66, with questioned orientation at 13.22 (Hall et al., 2021). This means that for media to positively affect the wellbeing of LGBTQIA+ youth, it should be catered to a younger audience, as well. Moreover, it should not enforce harmful LGBTQIA+ stereotypes.

As a young queer person experiencing the positive impact of such representation myself, albeit in adolescence, not childhood, I aim to examine areas where children's programming is doing well in catering for queer youth and what areas still need to be improved upon. Thus, this exploration could potentially contribute to the development of a more broadly inclusive media landscape.

Concept definitions:

- **LGBTQIA+**: "umbrella term for all people who have a non-normative gender identity or sexual orientation" (Montz & Solomon, 2021)
- **Queer**: "an umbrella term used by some to describe members of the LGBT+ community. It can also be used as an identity in its own right that rejects specific labels of romantic/sexual orientation and/or gender identity." (Brook, 2021)

- **Sexual orientation/sexuality**: "a holistic term for someone's sexual behaviours, attractions, likes, dislikes, kinks and preferences. Sexual orientation makes up a part of someone's sexuality, and sexuality is sometimes used interchangeably with sexual orientation."
- **Gender identity**: "A person's internal, deeply held sense of their gender." (GLAAD, 2021)
- **Asexual**: "describes people who do not experience sexual attraction"
- **Bi+**: "An encompassing term for people with the capacity to be attracted to more than one gender. Includes people who identify as bisexual, pansexual, fluid, and more."
- **Cis(gender)**: "people whose gender identity and expression match the sex they were assigned at birth" (Montz & Solomon, 2021)
- **Trans(gender)**: - "an umbrella term for people whose gender identity and/or gender expression differs from what is typically associated with the sex they were assigned at birth" (GLAAD, 2021)
- **Non-binary**: "used by some people who experience their gender identity and/or gender expression as falling outside the categories of man and woman"
- **Broadcast**: "a program that is broadcast on radio or television or over the Internet" (Webster, 1977)
- **Pay TV**: "television that you can watch only if you pay a fee such as a subscription to a satellite or cable television company" (Collins, 2010)
- **Streaming service**: "a service that allows you to play films and TV programmes on your TV, phone, or computer directly from the internet, without needing to download them"
- **TV Parental Guidelines** (TV Parental Guidelines Monitoring Board, n.d.): "a TV ratings system, giving information regarding the content and age-appropriateness of TV programs"
  - Y: program appropriate for all children (including ages 2–6)
  - Y7: program designed for children age 7 and above
  - Y7-FV: as Y7 with fantasy violence
  - G: suitable for general audience of all ages
  - PG: parental guidance suggested - program contains material that parents may find unsuitable for younger children

In GLAAD's reports (2021) several aspects of the media landscape are analysed, including the race, identity and neuro- diversity of queer characters, finding that whilst LGBTQIA+ people of colour outnumber white LGBTQIA+ characters, disabled and queer characters of different identities are severely underrepresented. Moreover, they observe that LGBTQIA+ inclusion is impacted by a handful of creators, who prioritise such portrayal in their work, and so it is not widespread. Based on these findings, I construct three key research questions, with which the children's subset of queer programming can be compared to the general, bigger picture.

1. What are the characteristics of children's TV media landscape regarding the presence of LGBTQIA+ characters?
2. To what extent are queer characters in children's programming diverse?
3. To what extent are children's shows with LGBTQIA+ characters accessible?

Data collection

The data for this project was mostly scraped from Insider (White & Chik, 2021), a research firm and news website, who have gathered the data about 259 LGBTQIA+ characters, appearing on 70 US animated children's television from 1983 until 2021. (See Appendix 1 for the scraping script.) I came across it after searching for different queer datasets that are not strictly demographics based.

Whilst the scraped dataset does not include any original metadata, the researchers have described the process of their data acquisition, e.g., scraping reports from specific organisations, interviews with experts to determine what children could recognise about the queer community, social-media posts and direct communication with the shows' creators. Thus, it is clear they conducted rigorous research into a field and the data seems reliable and valid. Moreover, it does not seem to include biases, since it included many fact-checking steps and appropriately labelled data that was not directly confirmed with the shows' creators.

I scraped the data into a csv file and used other sources – mostly individual show's IMDB pages to collect data about the show's first and last air, as well as confirm their parental rating.

I've also manually added data for the characters' disabilities, the shows' ratings (White & Chik, 2021), the platforms' type of service (broadcast, streaming...) and the subscription fee of streaming platforms. The latter was collected from the streaming platforms' websites. Lastly, I combined all data into Excel spreadsheets, adding indices, enabling me to join the tables in the analysis process.

Data overview and pre-processing

The dataset includes a variety of variables. Albeit mostly being nominal, there are some interval and ratio, however there are no clear ordinal variables. The table below lists all variables and their data types. Note that the Parental rating could to an extent be considered ordinal, as it determines appropriateness relation to age.

Variable	Variable data type
Character name	nominal
Gender	nominal
Race	nominal
Disability	nominal
Show	nominal
Year of show release	interval
Year of last episode	interval
Parental rating	nominal (/ordinal)
Platform	nominal
Service type	nominal
Minimum monthly streaming fee	ratio
Ad-free monthly streaming fee	ratio
Studio(s)	nominal

Variable	Variable data type
Creator	nominal
Representation	dichotomous nominal
Way of queernes confirmation	nominal

Whilst scraping, I made sure that the acquired data would be as clean and tidy as possible. However, there was data that needed to be parsed correctly and many variables made tidier by melting. Considerations had to be made regarding grouping variables (gender, orientation) into sensible categories and missing/erroneous values had to be handled.

In [2]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina' # Set this if you have a retin
```

In [3]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.interpolate import make_interp_spline # For smooth curve plotting

# Increase the size of the figures
plt.rcParams['savefig.dpi'] = 90
plt.rcParams['figure.dpi'] = 90

# Set themes and styles
sns.set_context("talk", rc={"lines.linewidth": 1.2})
sns.set_theme(style="whitegrid", palette="Set2", font_scale=1.05)
```

In [4]:
```
"""
| ------------------ |
|     Character      |
|    Data parsing    |
| ------------------ |
"""

df_characters = pd.read_excel("./data.xlsx", sheet_name='characters', header=
                              # Only set datatypes for certain variables, the
                              dtype={
                                  'Name': 'category',
                                  'show_id': 'int64',
                                  'Race': 'category',
                                  'Role': 'category',
                                  'Representation': 'category'
                              })

df_characters.head(5)
```

Out[4]:
```
                Name  show_id      Gender  Orientation      Race  Disability      Year
                                                                              Confirmed
character_id
           0    Acid       56  Genderfluid     Unknown       NaN         NaN        2018
               Storm
```

	Name	show_id	Gender	Orientation	Race	Disability	Year Confirmed
character_id							
1	Adam	55	Cis, Man	Gay	POC	NaN	2020
2	Adam	37	Cis, Man	Gay	Unknown	Blurred Vision	2018
3	Adora	50	Cis, Woman	Lesbian	White	NaN	2020
4	Alexander	46	Cis, Man	Gay	Undetermined	NaN	2018

In [5]:
```
def group_gender(val):
    """
        Groups gender values into less categories.
    """
    if val == 'Cis, Man' or val == 'Man':
        return '(Cis) man'
    elif val == 'Cis, Woman' or val == 'Woman':
        return '(Cis) woman'
    elif val == 'Trans, Man' or val == 'Trans, Nonbinary' or val == 'Trans, W
        return val.replace(',', '')
    elif val == 'Nonbinary':
        return val
    elif val == 'Unknown':
        return np.NaN
    else:
        return 'Other nonbinary'

def group_orientation(val):
    """
        Groups orientation values into less categories.
    """
    if val == 'Gay' or val == 'Lesbian' or val == 'Undetermined' or val == 'A
        return val
    elif val == 'Unknown':
        return np.NaN
    else:
        return 'Bi+'
```

In [6]:
```
"""

  Grouping

"""

# Grouping gender categories
df_characters['Gender'] = df_characters['Gender'].apply(group_gender).astype(

# Grouping sexuality values
df_characters['Orientation'] = df_characters['Orientation'].apply(group_orien
```

In [7]:
```
"""

  Dealing with unknown values
```

```
"""

# Replacing 'Unknown' and NaN race values and setting them as Non-human
df_characters['Race'] = df_characters['Race'].replace('Unknown', 'Non-human')
df_characters['Race'].fillna('Non-human', inplace=True)
```

In [8]:
```
"""

  Parsing time

"""

# Parsing yearly values as datetime objects
df_characters['Year Confirmed'] = pd.to_datetime(df_characters['Year Confirme
```

In [9]:
```
"""

  Helper function to melt observational values that include two values to cre

"""

def create_multiindex_series(input_series, new_series_name, index_names, _typ
    multi_ser = input_series.fillna('Unknown') # Fill the input NaN with Unkn
    # Expand the series, which creates new NaN for objects that have only one
    # We don't want to repeat ourselves, so in the stack() we lose those NaNs
    multi_ser = multi_ser.str.split(', ', expand=True)
    multi_ser = multi_ser.stack()
    multi_ser.name = new_series_name
    multi_ser.index.names = index_names
    multi_ser = multi_ser.replace('Unknown', np.NaN) # Put any existing unkno
    multi_ser = multi_ser.astype(_type)
    return multi_ser
```

In [10]:
```
"""

  Melting character observations that include two values into a multiindex se

"""

# Multiindex series to join with selected variables from the main dataframe i

# Disabiility multiindex series
disability_ser = create_multiindex_series(df_characters['Disability'], 'disab
#disability_ser.head(5)

# To join
#df_character_disab = df_characters.join(disability_ser)

# Type of confirmation multiindex series
confirm_ser = create_multiindex_series(df_characters['Confirmation'], 'confir
#confirm_ser.head(10)
```

In [11]:
```
# Drop the variables that were passed into multiindex series from the base da
df_characters.drop(labels=['Disability', 'Confirmation'], axis=1, inplace=Tru
df_characters.head(5)
```

Out[11]:

	Name	show_id	Gender	Orientation	Race	Year Confirmed	Role
character_id							
0	Acid Storm	56	Other nonbinary	NaN	Non-human	2018-01-01	Recurring Character
1	Adam	55	(Cis) man	Gay	POC	2020-01-01	Main Character
2	Adam	37	(Cis) man	Gay	Non-human	2018-01-01	Guest Character
3	Adora	50	(Cis) woman	Lesbian	White	2020-01-01	Main Character
4	Alexander	46	(Cis) man	Gay	Undetermined	2018-01-01	Recurring Character

In [12]:

```
"""
| ------------------- |
|        Show         |
|    Data parsing     |
| ------------------- |
"""

df_shows = pd.read_excel("./data.xlsx", sheet_name='shows', header=1, index_c
                              # Only set datatypes for certain variables, the
                              dtype={
                                  'Title': 'category',
                                  'Rating': 'category',
                                  'platform_id1': 'object',
                                  'platform_id2': 'object'
                              })
```

In [13]:

```
"""
    Parsing time (years)
"""
# Parsing yearly values as datetime objects
# Year of release
df_shows['Year of release'] = pd.to_datetime(df_shows['Year of release'], for
```

In [14]:

```
# Year of last run
# Replace the 'present' values as NaN so we can parse as Timestamps
df_shows['Year of last run'] = df_shows['Year of last run'].replace('present'
df_shows['Year of last run'] = pd.to_datetime(df_shows['Year of last run'], f
```

In [15]:

```
"""

  Dealing with variables spread across multiple columns - platform ids
  Melting shows observations into multiindex series

"""

# Melt all columns into a single variable
df_plat = df_shows[['platform_id1', 'platform_id2']]
df_plat = df_plat.melt(ignore_index=False, var_name='platform_num', value_nam
# Set dtypes
df_plat['platform_num'] = df_plat['platform_num'].astype('category')
df_plat.dropna(subset=['platform_id'], inplace=True)
```

```
df_plat = df_plat.astype({"platform_id": int})
df_plat['platform_num'] = df_plat['platform_num'].astype(
    pd.CategoricalDtype(categories=['platform_id1', 'platform_id2'], ordered=
df_plat.sort_index(inplace=True)
```

In [16]:
```
df_plat.sample(5).sort_index()
```

Out[16]:

show_id	platform_num	platform_id
4	platform_id1	19
17	platform_id1	22
45	platform_id1	17
61	platform_id1	2
69	platform_id1	6

In [17]:
```
# Drop old platform columns
df_shows.drop(['platform_id1', 'platform_id2'], axis=1, inplace=True)
```

In [18]:
```
df_shows.sample(4).sort_index()
```

Out[18]:

show_id	Title	Year of release	Year of last run	Rating	studio(s)	creator
16	6Teen	2004-01-01	2010-01-01	PG	Nelvana, Fresh TV	Jennifer Pertsch, Tom McGillis
29	The Legend of Korra	2012-01-01	2014-01-01	PG	Nickelodeon Animation Studio	Michael Dante DiMartino, Bryan Konietzko
38	Danger & Eggs	2017-01-01	2017-01-01	G	PUNY Entertainment, Amazon Studios	Mike Owens, Shadi Petosky
42	OK K.O.! Let's Be Heroes	2017-01-01	2019-01-01	Y7-FV	Cartoon Network Studios	Ian Jones-Quartey

In [19]:
```
"""
| ------------------ |
|      Platform      |
|    Data parsing    |
| ------------------ |
"""

df_platforms = pd.read_excel("./data.xlsx", sheet_name='platforms', header=1,
                             # Only set datatypes for certain variables, the
                             dtype={
                                 'Name': 'category',
                                 'Service': 'category'
                             })
```

In [20]:
```
"""
```

```
  Subscription fee data parsing

"""

# Strip the fees of dollar signs $$ and parse them as floats
df_platforms['ad_free_streaming'] = df_platforms['ad_free_streaming'].str.rep
df_platforms['ad_free_streaming'] = df_platforms['ad_free_streaming'].astype(

# Manage the 'Free ($4.99)' value from the Peacock platform
df_platforms['min_monthly_streaming_fee'] = df_platforms['min_monthly_streami
# Strip the fees of dollar signs $$ and parse them as floats
df_platforms['min_monthly_streaming_fee'] = df_platforms['min_monthly_streami
df_platforms['min_monthly_streaming_fee'] = df_platforms['min_monthly_streami

df_platforms.head(5)
```

Out[20]:

	Name	Service	min_monthly_streaming_fee	ad_free_streaming
platform_id				
0	ABC Kids	Broadcast	NaN	NaN
1	Amazon Prime Video	Streaming	8.99	NaN
2	Cartoon Network	Cable	NaN	NaN
3	DC Universe	Streaming	6.25	NaN
4	Discovery Family	Cable	NaN	NaN

Analysis

Current state of LGBTQIA+ representation

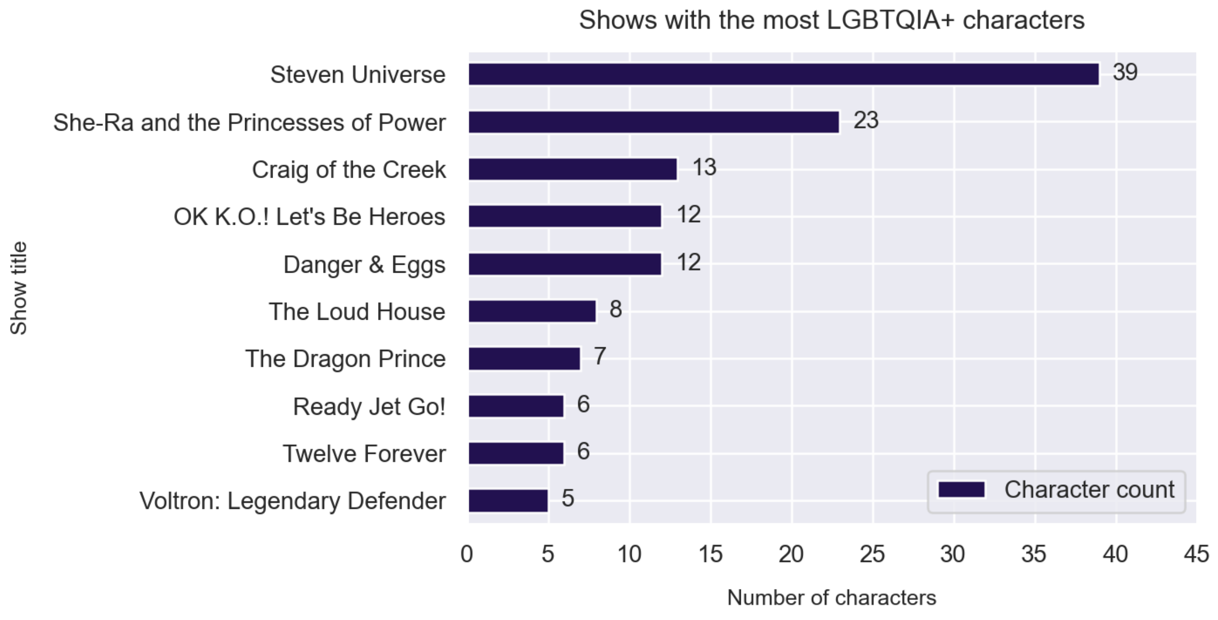
To get a general overview of the current state of LGBTQIA+ representation in children's programming, analysis of variables such as the *name* of the TV show, its *rating*, the *year it was released* and the *year* a character was *confirmed to be queer* and *how*, as well as how their queernes is *represented*, is done.

In [21]:
```
data_period = df_shows['Year of last run'].max().year - df_shows['Year of rel

print(f"In the dataset, there are {len(df_characters)} characters from {len(d

print(f"The most common parental guidance rating is {df_shows['Rating'].mode(
```

```
In the dataset, there are 259 characters from 70 shows, spanning across 38 yea
rs, starting in 1983.
The most common parental guidance rating is Y7, meaning these shows are most c
ommonly appropriate for older children.
```

The show with the most (39) characters is *Steven Universe*, followed by *She-Ra and the Princesses of Power* with 23 and *Craig of the Creek* with 13 characters.

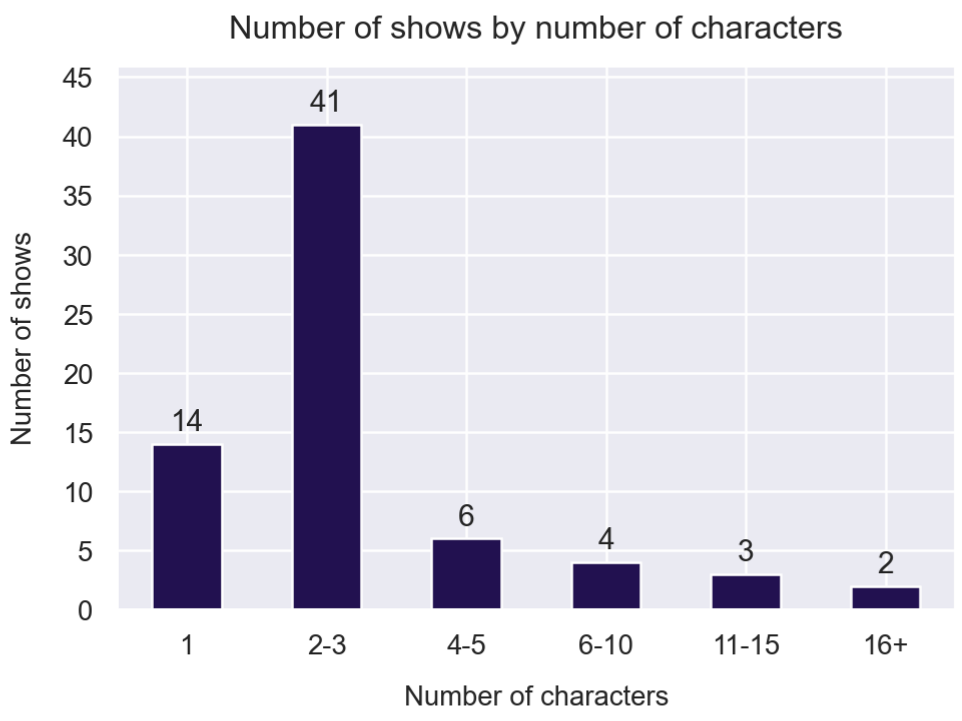
In [22]:
```
sns.set_theme(style="darkgrid", palette="magma")

ax = df_characters.join(df_shows, on='show_id') \
                        .groupby(by='Title', dropna=False).size() \
                        .sort_values(ascending=False).iloc[:10] \
                        .plot(x='Title',
                              y='character_count',
                              kind="barh", legend=False)

ax.invert_yaxis()
ax.set_title('Shows with the most LGBTQIA+ characters', pad=10)
ax.set_xlabel('Number of characters', fontsize=10, labelpad=10)
ax.set_ylabel('Show title', fontsize=10, labelpad=10)
ax.bar_label(ax.containers[0], padding=6, fontsize=11)
ax.set_xlim(right=45) # adjust xlim to fit labels
ax.legend(['Character count'])

plt.show()
```



Grouping the number of characters into bins for more effective communication, the plot below shows that children's animations most commonly have between 2 and 3 queer characters, with 14 shows having only one.

Instead of a histogram, a barplot is used, since the bins could mathematically be of various widths yet visually the same. This allows for better comparison, as it is clear that the data is clustered in the lower values, whereas such level of detail would not aid the understanding in the upper values, where the data is significantly more scarce and spread out. Moreover, this type of plot requires little information processing, since the features (the length of the bars) are highly distinct.

In [23]:
```
# Putting shows into bins based on the number of characters
df_char_show = df_characters['show_id'].value_counts(sort=True, ascending=Fal
                                        .rename_axis('show_id') \
                                        .reset_index(name='character_count').
char_bins = [0, 1, 3, 5, 10, 15, df_char_show['character_count'].max()+1]
df_char_show_bins = pd.cut(df_char_show['character_count'], char_bins, right=


ax = df_char_show_bins.value_counts(sort=False).plot(kind='bar', rot=0)

ax.set_xticklabels(['1', '2-3', '4-5', '6-10', '11-15', '16+'])
ax.set_yticks([0, 5, 10, 15, 20, 25, 30, 35, 40, 45])
ax.set_title('Number of shows by number of characters', pad=11, fontsize=13)
ax.set_xlabel('Number of characters', fontsize=11, labelpad=10)
ax.set_ylabel('Number of shows', fontsize=11, labelpad=10)
ax.bar_label(ax.containers[0], padding=3, fontsize=12)
ax.set_ylim(top=46) # adjust ylim to fit labels
plt.show()
```

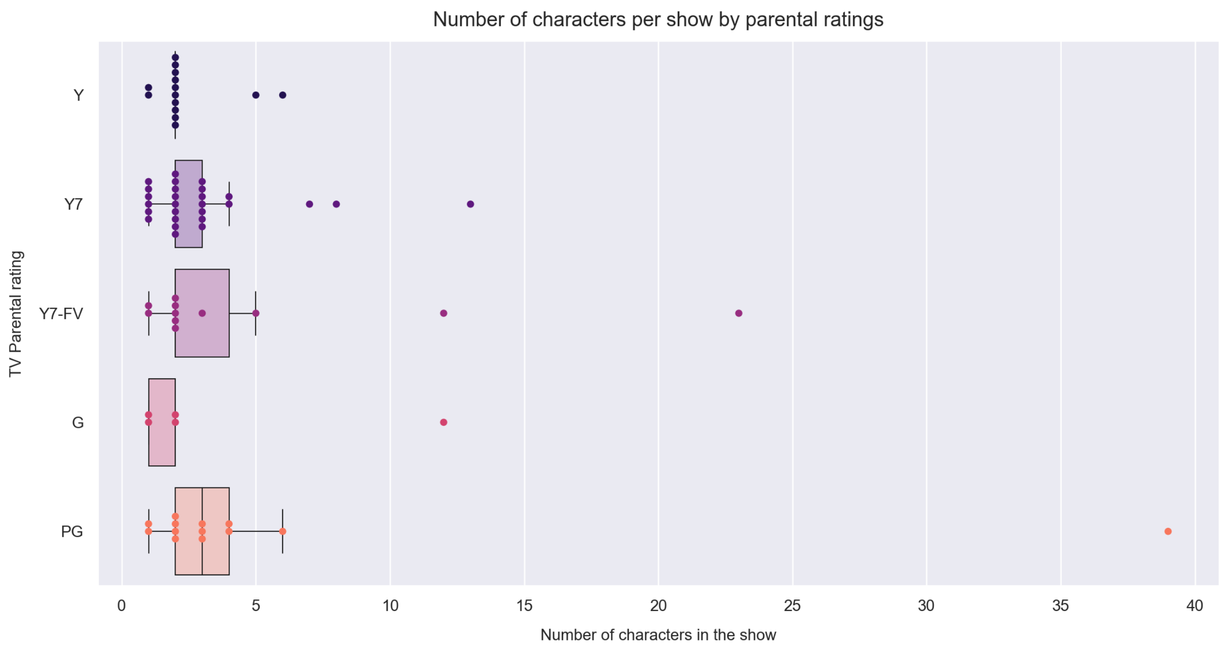


Looking at what audience the shows are appropriate for, we observe that there is no identifiable pattern between the number of LGBTQIA+ characters a show portrays and its parental rating. This is a promising finding as it means that queer representation is not excluded from younger audiences.

The box and swarm plots below illustrate this, showing how the character counts of the shows are distributed based on the show's ratings. The parental ratings variable is interpreted as ordinal in this case as shown by the choice of a sequential color scheme. The categories are ordered from those classifying appropriateness for the youngest children to those advising parental guidace, i.e., being more suitable for children of older ages.

In [24]:
```
sns.set_theme(style="darkgrid", palette="magma") # Use a sequential color sch

plt.figure(figsize=(14, 7))

rating_order = ['Y', 'Y7', 'Y7-FV', 'G', 'PG']

dat = df_shows[['Title', 'Rating']].join(df_characters['show_id'] \
                                         .value_counts(sort=False) \
                                         .rename_axis('show_id').reset_index(
                                         .set_index('show_id').sort_index(),
                                         on='show_id')


ax = sns.boxplot(y=dat.Rating, x=dat.character_count,
                 order=rating_order, linewidth=0.75, orient='h',
                 saturation=1, showfliers=False) #Don't show outliers, becaus

# Change opacity only for the box plot box faces
# Resource: (Mwaskom, 2016)
for patch in ax.artists:
    r, g, b, a = patch.get_facecolor()
    patch.set_facecolor((r, g, b, .3)) # Set lower opacity for box plot face
```

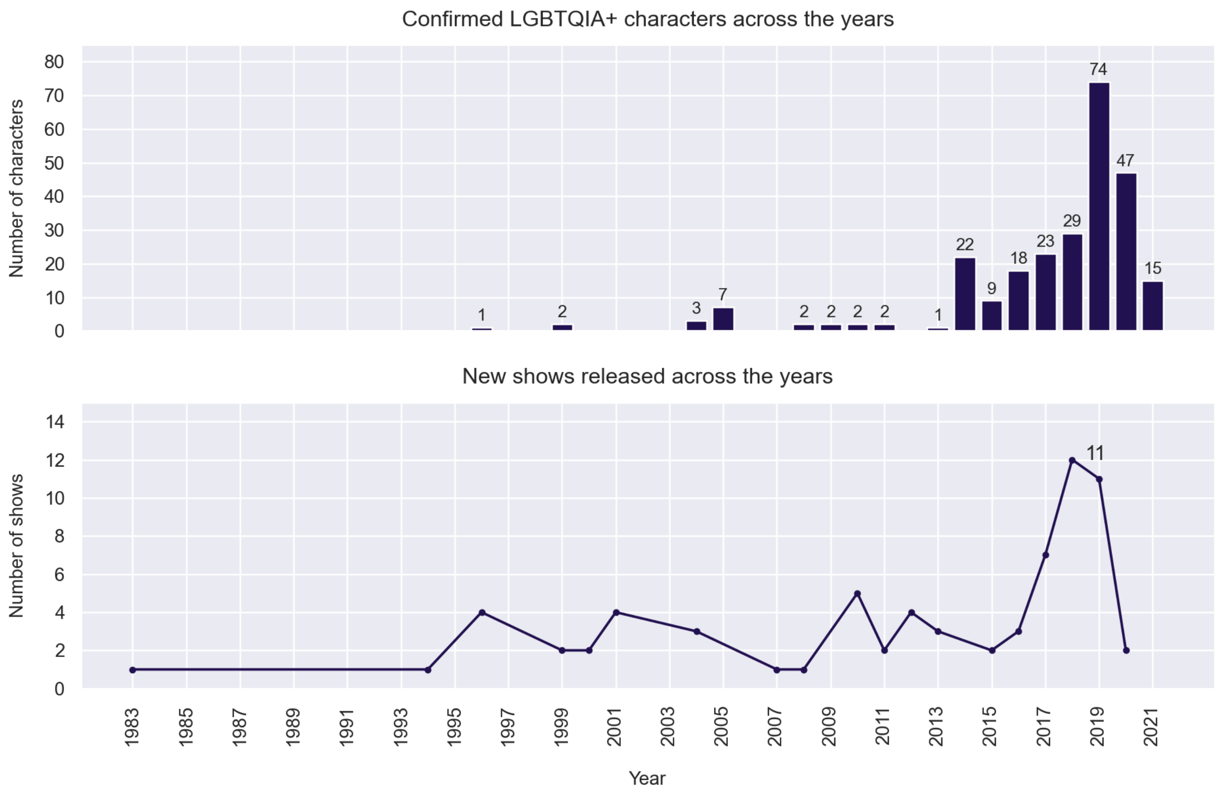
```
ax = sns.swarmplot(y=dat.Rating, x=dat.character_count, order=rating_order, o

ax.set_title('Number of characters per show by parental ratings', pad=10, fon
ax.set_ylabel('TV Parental rating', fontsize=11, labelpad=10)
ax.set_xlabel('Number of characters in the show', fontsize=11, labelpad=10)

plt.show()
```



Looking at change over time, we observe that the number of LGBTQIA+ characters has increased, with 2019 being the year when the most (74) characters were confirmed to be queer. It is worth noting that this is also the year with the second largest number of shows premiering, which may account for the big spike in queer representation. This comparison is clearly illustrated by two subplots, using the same x-axes, both with features that require little cognitive energy to be perceived and understood.

In [25]:
```
# Count characters confirmed each year
count_char_year = df_characters['Year Confirmed'].value_counts(sort=False)
count_char_year.index = count_char_year.index.year
count_char_year.sort_index(inplace=True)

# Group the shows by year of release to count how many shows were release eac
release_y = df_shows.groupby(by='Year of release').size()
release_y.index = release_y.index.year

# Plotting
fig, axes = plt.subplots(2, 1, figsize=(12, 7), sharex=True)
plt.subplots_adjust(hspace=0.25)

# Bar plot
axes[0].bar(count_char_year.index, count_char_year.values)
axes[0].set_title('Confirmed LGBTQIA+ characters across the years', pad=11, f
axes[0].set_yticks(np.arange(0, 81, 10))
axes[0].set_ylim(top=85) # adjust ylim to fit labels
axes[0].set_ylabel('Number of characters', fontsize=11, labelpad=10)
axes[0].bar_label(axes[0].containers[0], padding=2, fontsize=10)

# Time series
release_y.plot(subplots=True, ax=axes[1], kind='line', rot=90,
               marker='o', ms=3)
```

```
axes[1].set_title('New shows released across the years', pad=11, fontsize=13)
axes[1].set_xlabel('Year', fontsize=11, labelpad=15)
axes[1].set_ylabel('Number of shows', fontsize=11, labelpad=10)
axes[1].set_xticks(np.arange(1983, 2022, 2))
axes[1].set_yticks(np.arange(0, 15, 2))
axes[1].annotate(str(release_y.loc[2019]), xy=(2019-0.5, release_y.loc[2019]+
axes[1].set_ylim(top=15) # adjust ylim to fit labels

plt.show()
```



Comparing this with the shows' release dates, we see that shows often confirm the existence of a queer character in the first years after the release, most commonly in the same one. Thus, we could infer that such shows incorporate queer storylines right at the beginning.

One big outlier, however, is the Skeleton character from *SuperTed*, which premiered in 1983, yet Skeleton's queernes was confirmed only 31 years later, in 2014, despite being the main character. Similarly, 33 characters were confirmed only at a year after the show has finished running or more.

Moreover, the scatterplot on the right illustrates that LGBTQIA+ characters seem to be confirmed in newer shows earlier and the characters whose queernes was confirmed years after the show's release are generally older. However, this doesn't necessarily mean that a character was introduced but only confirmed to be queer later, it could also be the case that older shows have been running for a long time before the character first appeared. Nevertheless, it is clear that newer shows introduce queer characters earlier in production.

In [26]:
```
# Calculating the number of years it took a show to confirm a character is qu
df_chsh_conf = df_characters.join(df_shows, on='show_id')
df_chsh_conf['confirm_delta'] = (df_chsh_conf['Year Confirmed'].dt.year - df_

# Add boolean column for whethe the character was confirmed to be LGBTQIA+ af
df_chsh_conf['confirmed_after_end'] = (df_chsh_conf['Year of release'].dt.yea
df_chsh_conf.reset_index(drop=False, inplace=True)
df_chsh_conf.set_index('show_id', append=False, inplace=True, drop=True)
df_chsh_conf.sort_index(inplace=True)
df_chsh_conf.set_index('character_id', append=True, inplace=True) # Create mu

# Get when the show confirmed a first character in years
dat = df_chsh_conf.groupby('show_id')['confirm_delta'].min()
```

In [27]:

```
# Get the character that took the longest to be confirmed
df_chsh_conf.loc[df_chsh_conf['confirm_delta'] == df_chsh_conf['confirm_delta
```

Out[27]:

		Name	Gender	Orientation	Race	Year Confirmed	Role	Represent
show_id	**character_id**							
0	**206**	Skeleton	(Cis) man	Gay	Non-human	2014-01-01	Main Character	Im

In [28]:

```
# Compute how many shows confirmed a character is queer after for the first t
df_chsh_conf['confirmed_after_end'].value_counts().sort_index()
```

Out[28]:

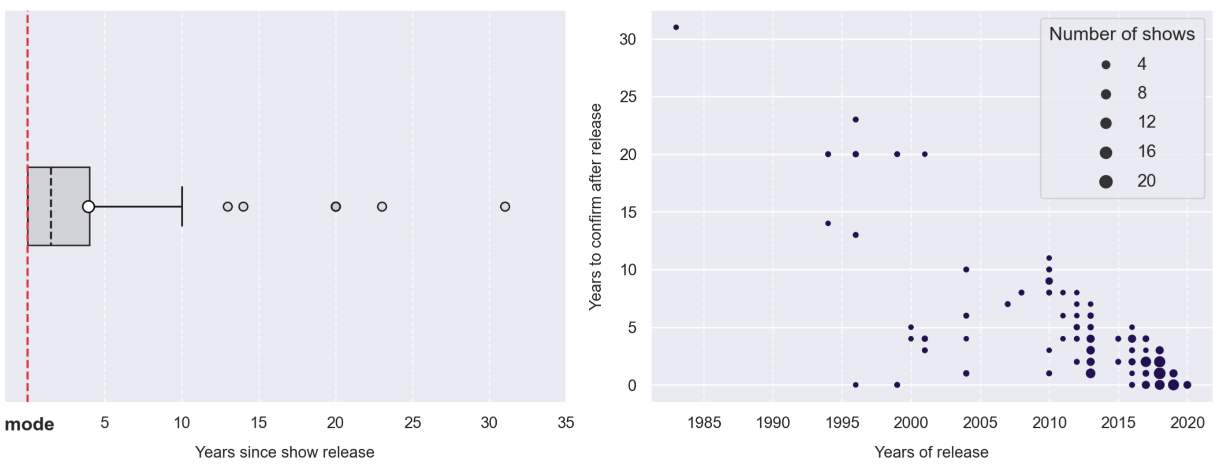
```
False    226
True      33
Name: confirmed_after_end, dtype: int64
```

In [29]:

```
fig,axes = plt.subplots(1, 2, figsize=(15, 5), sharex=False, sharey=False)
plt.subplots_adjust(wspace=0.15)
fig.suptitle('Years of first LGBTQIA+ character confirmation since show relea


# Box plot
sns.boxplot(ax=axes[0], x=dat.values, orient='h', linewidth=1.25, width=0.2, 
            flierprops=dict(marker='o', markersize=6, markerfacecolor=(0.75,
            medianprops=dict(alpha=1, linewidth=1.4, linestyle='--'),
            boxprops=dict(facecolor="None", linewidth=1),
            meanprops=dict(marker='o',
                           markerfacecolor="white",
                           markeredgecolor="black",
                           markersize=8))

# Compute the mode so we can show it on the graph
mode = dat.mode()[0]
axes[0].axvline(mode, color=sns.color_palette("rocket")[3], linestyle='--')
axes[0].annotate("mode", xy=(mode, 0), xytext=(-1.5, 0.57), size=13, fontweig
axes[0].set_xlabel('Years since show release', fontsize=11, labelpad=10)
axes[0].set_xticks([5, 10, 15, 20, 25, 30, 35])

# Add a horizontal grid to the plot, but make it very light in color
# so we can use it for reading data values but not be distracting
# Resource Hunter et al.
axes[0].xaxis.grid(True, linestyle='dashed', which='major', color='white', al
axes[0].set(axisbelow=True) # Hide the grid behind plot objects

# Box plot face opacity
for patch in axes[0].artists:
    r, g, b, a = patch.get_facecolor()
    patch.set_facecolor((r, g, b, 0.1))
```

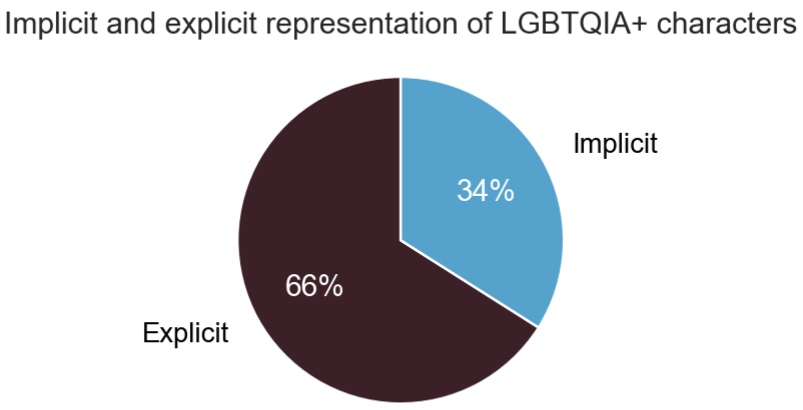
```
# Scatter plot
counts = pd.DataFrame(df_chsh_conf.groupby(by=['Year of release', 'confirm_de

counts.columns = ['count']
counts.reset_index(drop=False, inplace=True)
counts['Year of release'] = counts['Year of release'].dt.year
counts.set_index('Year of release', drop=True, inplace=True)

sns.scatterplot(ax=axes[1], x=counts.index, y=counts['confirm_delta'], size=c

axes[1].set_ylabel('Years to confirm after release', fontsize=11, labelpad=10
axes[1].set_xlabel('Years of release', fontsize=11, labelpad=10)
axes[1].legend(title='Number of shows', fontsize=12, title_fontsize=13, borde

# Customise the grid
axes[1].xaxis.grid(True, linestyle='dashed', which='major', color='white',
           alpha=0.75)

plt.show()
```

Years of first LGBTQIA+ character confirmation since show release



Looking at how shows confirm that their characters are queer, the below pie chart shows that 2 out of 3 queer characters in children's animation are explicitly represented. This is rather surprising since the most common way of confirmation was implicit social-media posts - confirming the character's identity outside the storyplot. Insider researchers have classified recent interviews with the shows' creative teams as explicit, which might account for this majority. Nevertheless, it is clear that inclusivity in children's programming is not covert, with other most common ways of confirmation being showing affection, a romantic relationship, parenting and kissing.

In [30]:
```
fig, ax = plt.subplots(figsize=(3, 3))

patches, texts, autotexts = ax.pie(df_characters['Representation'].value_coun
                                   labels=[df_characters['Representation'].va
                                           df_characters['Representation'].va
                                   autopct='%1.0f%%',
                                   counterclock=True,
                                   wedgeprops = {'linewidth': 1},
                                   labeldistance=1.2, startangle=90,
                                   colors=[sns.color_palette("icefire")[3], #
                                           sns.color_palette("icefire")[0]])

# Setting custom colors for individual text in the chart
# Resource: (Shahnoza, 2021)
[text.set_color('black') for text in texts]
[autotext.set_color('white') for autotext in autotexts]

ax.set_title('Implicit and explicit representation of LGBTQIA+ characters', p

plt.show()
```



In [31]:

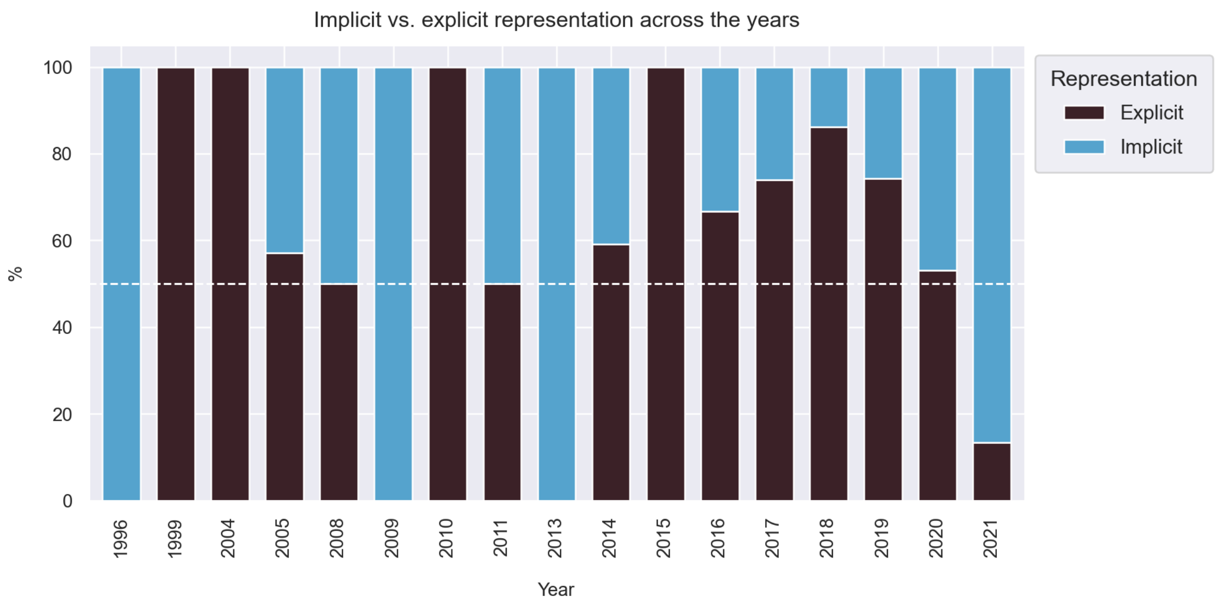
```
# Computing most common way of confirmation

# Join the multiindex series with the character dataframe
confirm_df = pd.DataFrame(data=confirm_ser)
confirm_df.index.names = ['character_id', 'confirmation_id']

df_char_conf = confirm_df.join(df_characters, on='character_id')
df_char_conf.reset_index(inplace=True)
df_char_conf.set_index(['character_id', 'Representation'], inplace=True, appe
df_char_conf.drop('confirmation_id', axis=1, inplace=True)

d = pd.DataFrame(df_char_conf['confirmation_type'].value_counts())
d.index.name = 'Type of confirmation'
d.columns=['Number of characters']
d.head(10)
```

Out[31]:

	Number of characters
Type of confirmation	
Social Media	110
Creative Team Interview	58
Affection	49
Romantic Partnership	43
Parenting	43
Kiss	30
Character Proximity	24
Insider Confirmation	20
Voice Actor	19
Marketing	16

The implicit-explicit relationship has stayed true across 11 out of the 25 years.

```
In [32]: table = pd.crosstab(df_characters['Representation'], df_characters['Year Conf
         table = table.transpose()

         fig, ax = plt.subplots(figsize=(10, 5))

         table.plot(ax=ax, kind='bar', stacked=True, width=0.7,
                    color=[sns.color_palette("icefire")[3], # Use colors from a diverg
                           sns.color_palette("icefire")[0]])
         ax.set_xlabel('Year', fontsize=11, labelpad=15)
         ax.set_ylabel('%', fontsize=11, labelpad=10)
         ax.set_title('Implicit vs. explicit representation across the years', pad=11,
         ax.legend(title='Representation', bbox_to_anchor=(1.0, 1.0), fontsize=12, tit
         plt.axhline(y=50, color='#ffffff', linestyle='--', linewidth=1.2, alpha=1)

         plt.show()
```



Diversity

When it comes to diversity in representation, the variable related to a character's characteristics are of interest; their *gender identity*, *race*, *sexual orientation* and *disability*, as well as the kind of *role* they play.

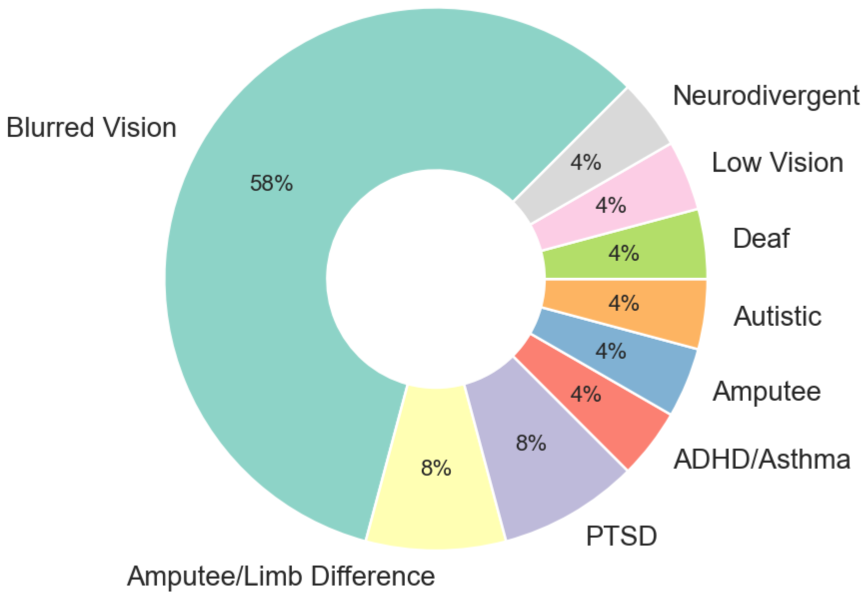
Firstly, looking at some basic summaries, we observe that the most common identity (102) characters was a (cis) woman, whilst the most common sexuality was gay (50 characters). 27.4% of the characters are White, which is the most common race and only 24 out of 259 characters or approximately 1 in 11 has a disability.

```
In [33]: df_characters['Gender'].describe()

Out[33]: count              243
         unique               7
         top        (Cis) woman
         freq               102
         Name: Gender, dtype: object
```

```
In [34]: race_human = df_characters['Race'].replace('Non-human', np.NaN) # Ignore non-
         race_mode_perc = round((race_human.describe().freq/len(race_human)) * 100, 1)
         print(f"Most commmon race of human characters is {race_human.mode()[0]} with

         Most commmon race of human characters is White with 27.4% of characters.
```

In [35]:
```
print(f"Number of characters with a disability: {disability_ser.notna().sum()
```
```
Number of characters with a disability: 24.
```

In [36]:
```
sns.set_theme(palette="Set3") # Use a qualitative color scheme

fig, ax = plt.subplots(figsize=(10, 5))
wedges, texts, autotexts = ax.pie(disability_ser.value_counts(),
                                  wedgeprops=dict(width=0.60),
                                  startangle=45,
                                  autopct='%1.0f%%',
                                  pctdistance=0.7,
                                  labels=disability_ser.value_counts().index,
                                  labeldistance=1.1)

[autotext.set(fontsize = 9) for autotext in autotexts]
plt.show()
```

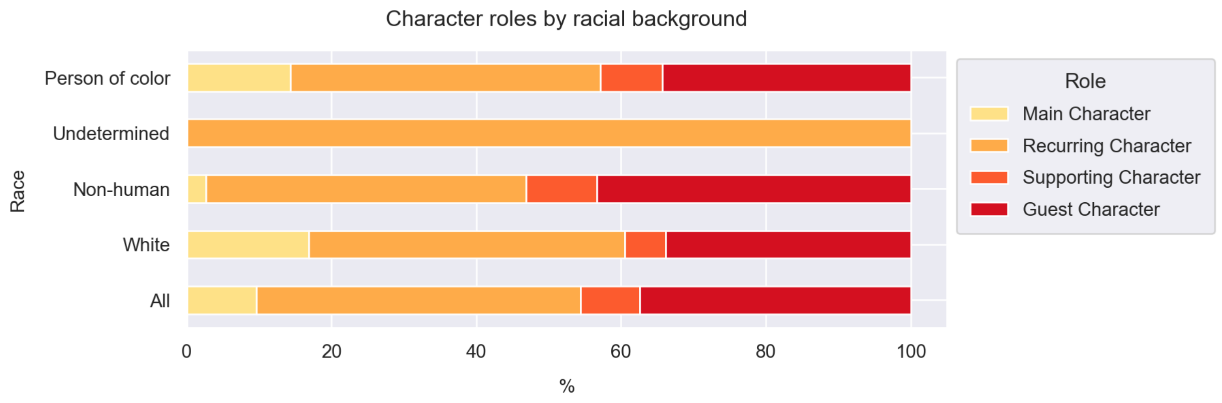


Looking at who takes centre stage in the analysed shows, we observe that this is not LGBTQIA+ characters. Rather, they predominantly fall into recurring or guest character roles. This distribution is arguably constant across all races, excluding characters with undetermined race, i.e., non-human and human-hybrid characters.

The below stacked bar plot makes these proportions clear. The selected sequential color scheme is intentional, as the type of role can contextually be ordered with the main role being on one side of the spectrum and guest on the other.

In [37]:
```
# Compute crosstabulation
table = pd.crosstab(df_characters['Race'], df_characters['Role'], margins=Tru
table.index = table.index.str.replace('POC', 'Person of color')

fig, ax = plt.subplots(figsize=(8, 3))

order = ['Main Character', 'Recurring Character', 'Supporting Character', 'Gu
table = table.reindex(columns=order) # Set the order for the columns (roles)
table.plot(ax=ax, kind='barh', stacked=True,
           color=sns.color_palette("YlOrRd", 4)) # Use a sequential color sch
```

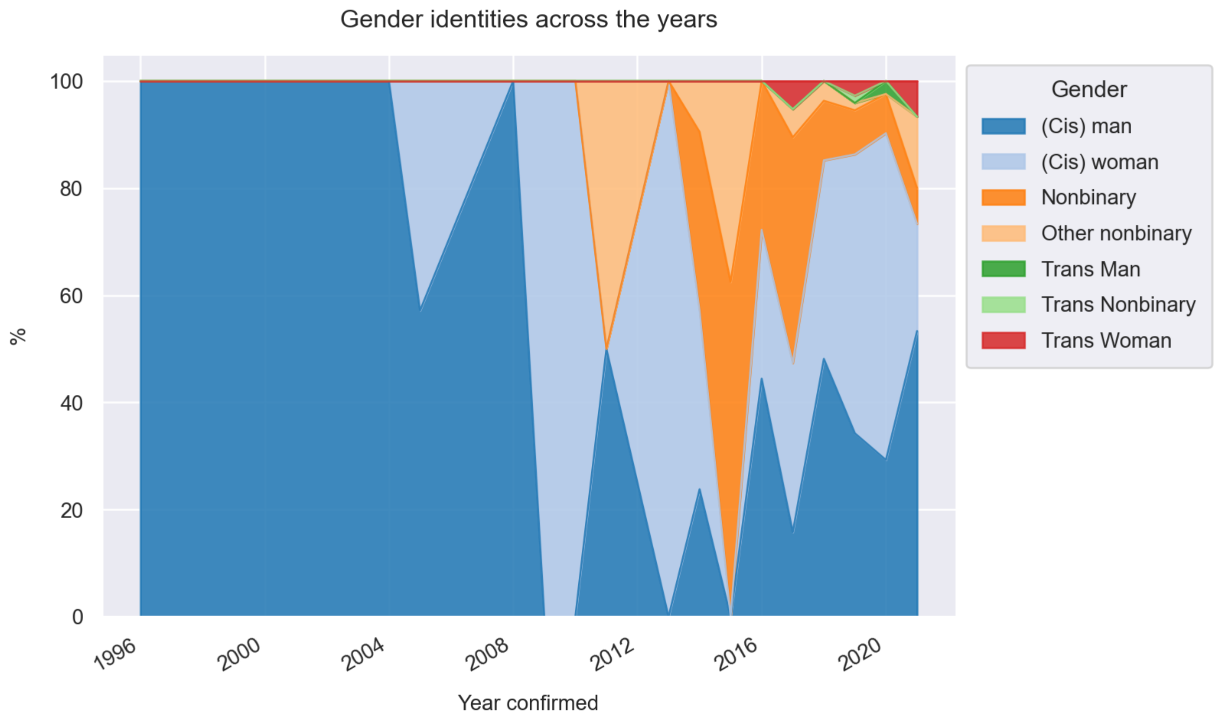
```
ax.invert_yaxis()
ax.set_xlabel('%', fontsize=11, labelpad=10)
ax.set_ylabel('Race', fontsize=11, labelpad=10)
ax.set_title('Character roles by racial background', pad=14, fontsize=13)
ax.legend(title='Role', bbox_to_anchor=(1.0, 1.0), fontsize=11, title_fontsiz

plt.show()
```



Being particularily interested in different identity minorities, the below plots visualise the ratios between several identities across the years. It is clear that the children's media landscape has become more diverse, with more gender identities, sexualities and racial backgrounds. It is worth noting that in terms of gender, cis characters are still highly dominant, with trans and nonbinary characters representing a mere fraction. Similarly, asexual and bi+ characters are underrepresented.

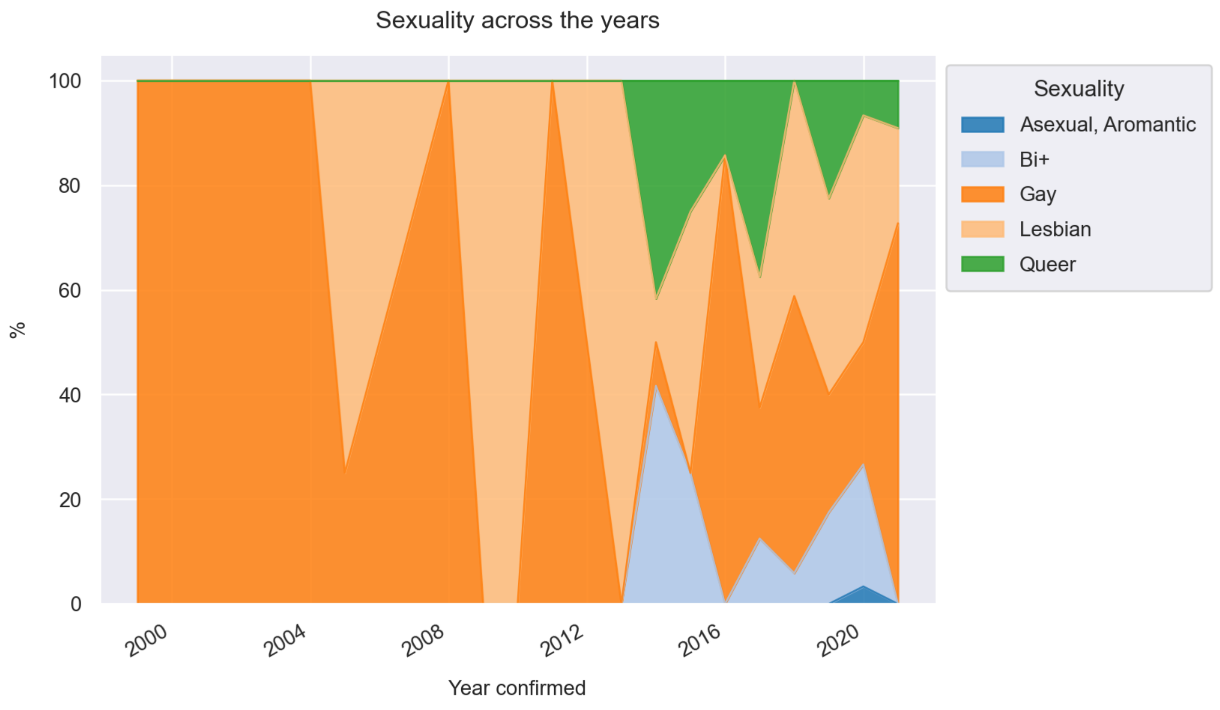
A percentage stacked area plot is chosen to illustrate these trends, because with the overall number of characters increasing, it is harder to compare the ratios between the categories. With the human brain being sensitive to patterns of change, combining this with a qualitative color scheme, the change over time is more pronounced, despite it being rather irregular. Specifically in the gender category, a paired color scheme is used to group together similar (minority) identities.

In [38]:
```
# Plotting Gender identities across the years
# normalized crosstabulation in %
tab = pd.crosstab(df_characters['Year Confirmed'], df_characters['Gender'], n
ax = tab.plot(kind='area', figsize=(8, 6), linewidth=1, alpha=0.85,
         color=sns.color_palette("tab20", 8)) # use a paired color scheme wit

ax.set_xlabel('Year confirmed', fontsize=11, labelpad=10)
ax.set_ylabel('%', fontsize=11, labelpad=10)
ax.set_title('Gender identities across the years', pad=14, fontsize=13)
ax.legend(title='Gender', bbox_to_anchor=(1.0, 1.0), fontsize=11, title_fonts

plt.show()
```



In [39]:

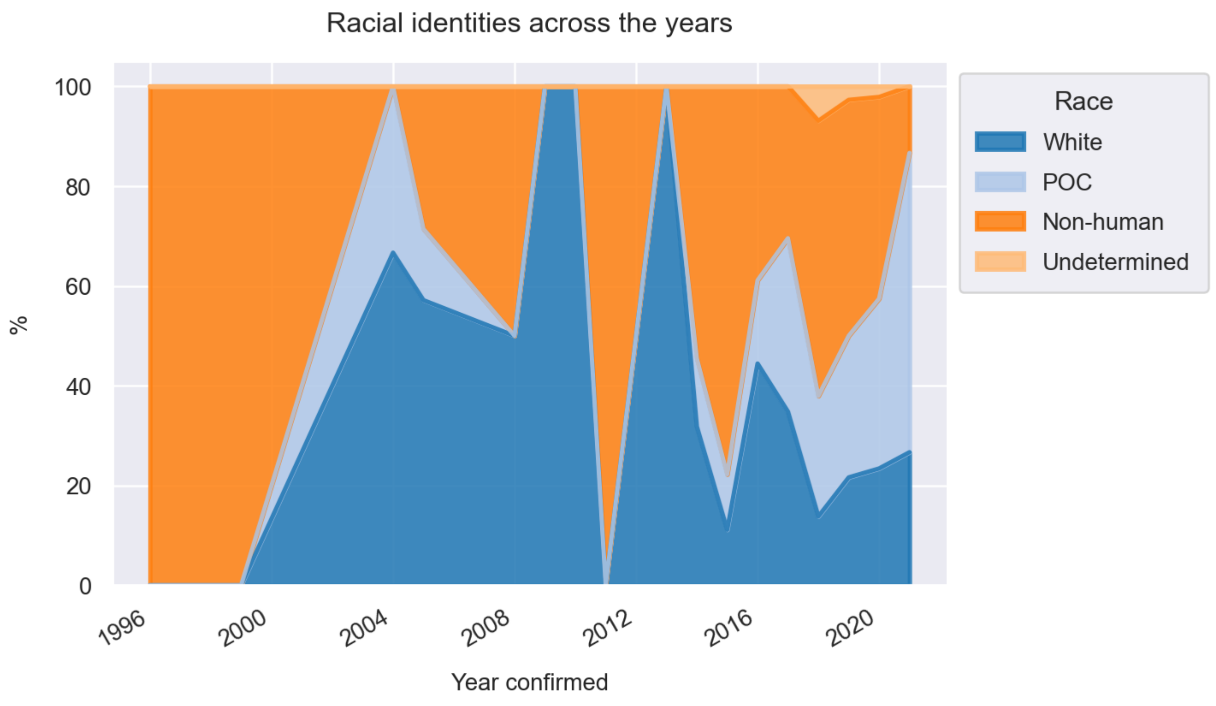
```
# Plotting sexuality across the years
d = df_characters['Orientation'].replace('Undetermined', np.NaN)
tab = pd.crosstab(df_characters['Year Confirmed'], d, normalize='index') * 10
ax = tab.plot(kind='area', figsize=(8, 6), linewidth=1, alpha=0.85,
              color=sns.color_palette("tab20", 5))

ax.set_xlabel('Year confirmed', fontsize=11, labelpad=10)
ax.set_ylabel('%', fontsize=11, labelpad=10)
ax.set_title('Sexuality across the years', pad=14, fontsize=13)
ax.legend(title='Sexuality', bbox_to_anchor=(1.0, 1.0), fontsize=11, title_fo

plt.show()
```



In [40]:

```
# Plotting race across the years
tab = pd.crosstab(df_characters['Year Confirmed'], df_characters['Race'], nor
# Reorder the columns
```

```
order = ['White', 'POC', 'Non-human', 'Undetermined']
tab = tab.reindex(columns=order)

ax = tab.plot(kind='area', figsize=(7, 5), linewidth=2, alpha=0.85,
         color=sns.color_palette("tab20", 4))

ax.set_xlabel('Year confirmed', fontsize=11, labelpad=10)
ax.set_ylabel('%', fontsize=11, labelpad=10)
ax.set_title('Racial identities across the years', pad=14, fontsize=13)
ax.legend(title='Race', bbox_to_anchor=(1.0, 1.0), fontsize=11, title_fontsiz

plt.show()
```



Disecting the representation of different identites further, the below plot disects the characters based on their gender, race and role. With the help of facets, the difference between cis and trans representation is striking. Another observation that stands out is that nonbinary characters are disproportionately represented as non-human characters.

In [2]:
```
'''
    Helper Function to group the genders into custom groups
    Based on: Zelazny7 (2020)
'''
def f(row):
    if row['Gender'] == 'Nonbinary' or row['Gender'] == 'Other nonbinary':
        val = 'Nonbinary'
    elif row['Gender'] == 'Trans Man' or row['Gender'] == 'Trans Woman' or ro
        val = 'Trans'
    elif row['Gender'] == '(Cis) woman':
        val = '(Cis) woman'
    elif row['Gender'] == '(Cis) man':
        val = '(Cis) man'
    else:
        val = np.NaN
    return val
```

In [73]:
```
dat = df_characters[['Gender', 'Race', 'Role']].copy()
# Apply the function and add a column with the grouped gender
```

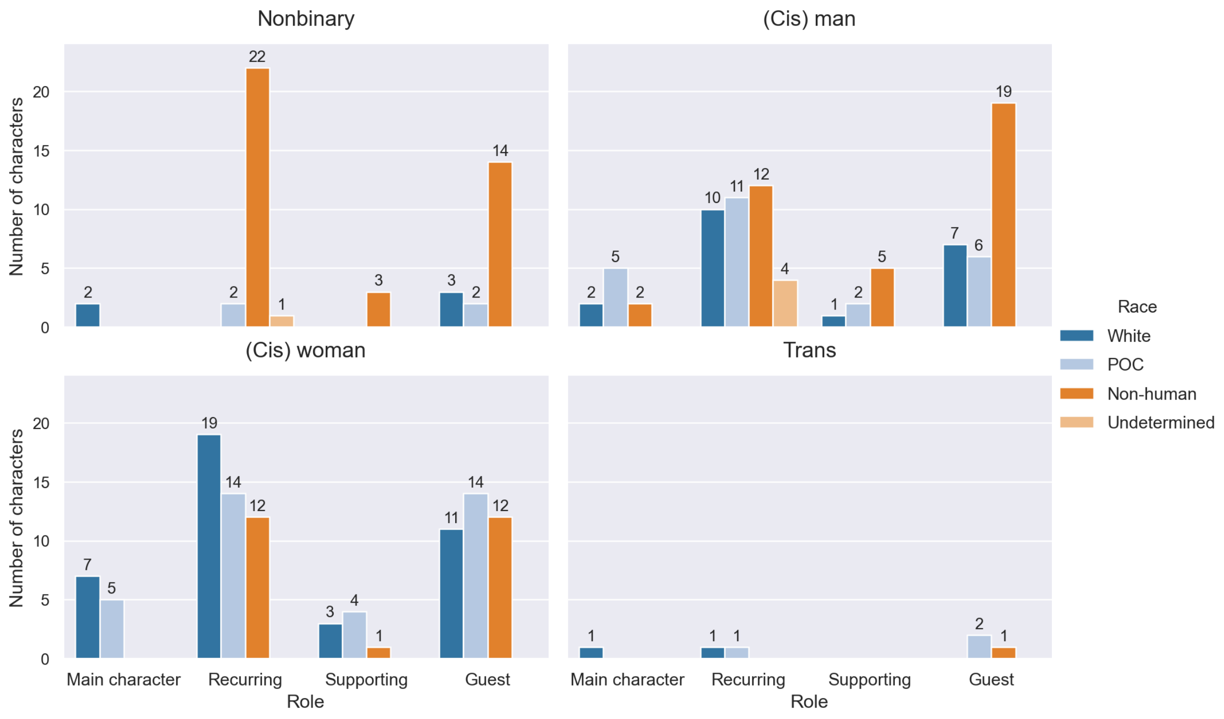
```
dat['Grp_gender'] = dat.apply(f, axis=1)
dat.head(3)
```

Out[73]:

character_id	Gender	Race	Role	Grp_gender
0	Other nonbinary	Non-human	Recurring Character	Nonbinary
1	(Cis) man	POC	Main Character	(Cis) man
2	(Cis) man	Non-human	Guest Character	(Cis) man

In [102…

```
# Using facets to visualise multiple variables

g = sns.catplot(col='Grp_gender', x="Role", col_wrap=2,
                order=['Main Character', 'Recurring Character', 'Supporting C
                facet_kws={},
                hue='Race',
                hue_order=['White', 'POC', 'Non-human', 'Undetermined'],
                data=dat,
                dodge=True,
                legend=False,
                kind="count", height=3.5, aspect=1.5,
                palette=sns.color_palette("tab20", 4))

#g.set(ylim=(0, None))
g.set_axis_labels("Role", "Number of characters", size=13)
g.set_xticklabels(['Main character', 'Recurring', 'Supporting', 'Guest'], siz
g.set_titles("{col_name}", size=15, pad=13)
g.add_legend(title='Race', bbox_to_anchor=(1.0, 0.5), fontsize=12, title_font

# Bar annotations

for ax in g.axes:
    ax.set_ylim(0, 24)
    for i in range(0, len(ax.containers), 1):
        ax.bar_label(ax.containers[i], padding=2, fontsize=11)

plt.show()
```



Accessibility

With the last aspect of the exploration being accessibility of LGBTQIA+ friendly TV shows, the variables of interest are the *platform* on which the show airs, the type of platform (*service*) it is and how much are the *subscription fees* (the minimum and ad-free) for streaming services.

Looking at descriptive statistics, we observe that there are 28 different platforms across 5 different types of services namely broadband, cable, pay TV, streaming and Youtube, with pay TV being the most common one.

In [103…

```
# Compute descriptive statistics
df_platforms['Service'].describe()
```

Out[103…

```
count         28
unique         5
top       Pay TV
freq           8
Name: Service, dtype: object
```

In [104…

```
list(df_platforms['Service'].cat.categories)
```

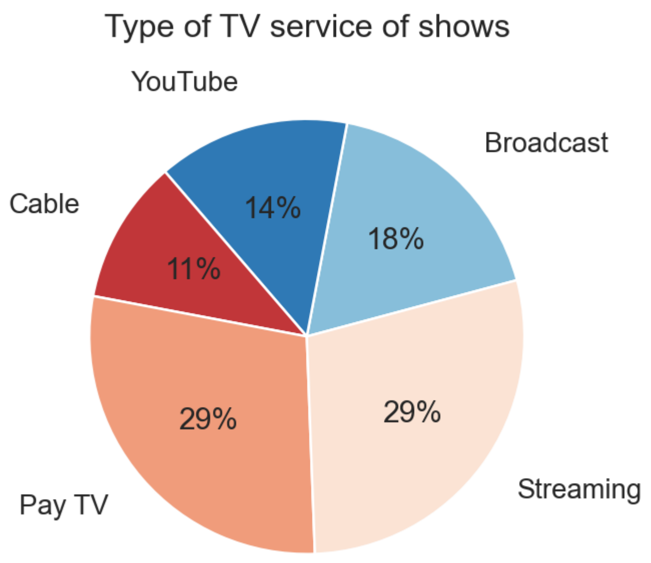
Out[104…

```
['Broadcast', 'Cable', 'Pay TV', 'Streaming', 'YouTube']
```

Considering that cable, pay TV and streaming are all services with a fee, the pie chart below makes it clear that more than a half of inclusive children's TV shows are not accessible to everyone. This is illustrated by appropriatelly selected colours that consider Gestalt's law of similarity, i.e., they are more similar in hue.

In [112…

```
fig, ax = plt.subplots()

# Get selected colors from the palette to group paid services together
cols = list(sns.color_palette("RdBu", 6)[4:])
[cols.append(col) for col in sns.color_palette("RdBu", 6)[:3]]

# Set custom order for the services
key_dict = {"Broadcast" : 0, 'YouTube': 1, 'Cable': 2, 'Pay TV': 3}
dat = df_platforms['Service'].value_counts().sort_index(key=lambda x: x.map(k

patches, texts, autotexts = ax.pie(dat,
                                   labels=[dat.index[0],
                                           dat.index[1],
                                           dat.index[2],
                                           dat.index[3],
                                           dat.index[4]],
                                   colors=cols,
                                   autopct='%1.0f%%',
                                   counterclock=True,
                                   wedgeprops = {'linewidth': 1},
                                   labeldistance=1.2, startangle=15)

ax.set_title('Type of TV service of shows', pad=11, fontsize=13)

plt.show()
```



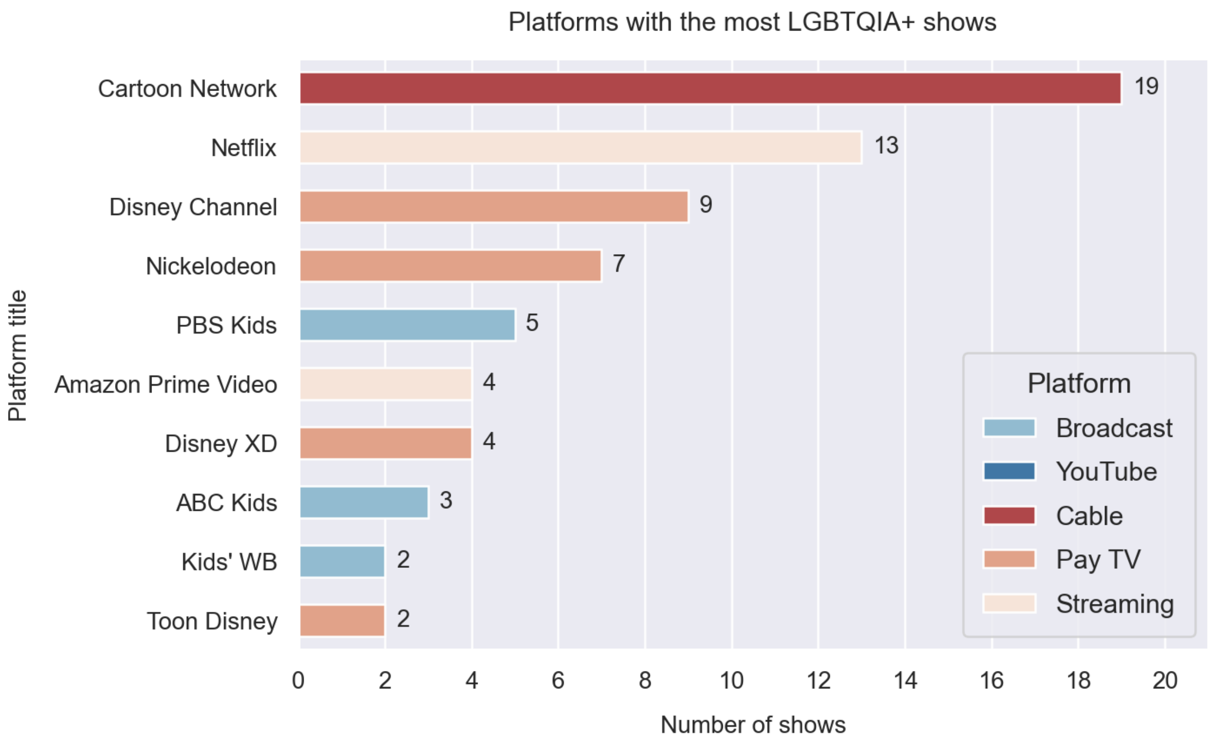
Regarding the number of shows, the cable platform Cartoon Network and Netflix are in the top places.

In [113…

```
# Compute the number of shows on each platform
dat = df_plat.dropna(subset=['platform_id'])
dat = dat.astype({"platform_id": int})
dat['platform_num'] = dat['platform_num'].astype(
    pd.CategoricalDtype(categories=['platform_id1', 'platform_id2'], ordered=
dat.sort_index(inplace=True)
datx = dat.value_counts(sort=False, subset='platform_id')
datx.name = 'show_count'
df_platform_shows = df_platforms.join(datx, on=datx.index)
```

In [114…

```
fig, ax = plt.subplots(figsize=(7.5, 5))

dat = df_platform_shows.sort_values('show_count', ascending=False).iloc[:10]
sns.barplot(ax=ax, data=dat,
            y='Name',
            orient='h',
            order = ['Cartoon Network', 'Netflix', 'Disney Channel', 'Nickelo
            hue='Service',
            hue_order = ['Broadcast', 'YouTube', 'Cable', 'Pay TV', 'Streamin
            x='show_count',
            dodge=False,
            palette=cols)

# Customising the height of the bars
# Resource: (jsgounot & Jed, 2019)
locs = ax.get_xticks()
new_value = 0.55
for i,patch in enumerate(ax.patches):
    current_height = patch.get_height()
    diff = current_height - new_value

    # Change the bar width
    patch.set_height(new_value)

    # Recenter the bar
    patch.set_y(patch.get_y() + diff * .5)

ax.set_title('Platforms with the most LGBTQIA+ shows', fontsize=12, pad=13)
ax.set_xlabel('Number of shows', fontsize=11, labelpad=10)
ax.set_ylabel('Platform title', fontsize=11, labelpad=10)
ax.set_xticks([0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20])
ax.set_xlim(right=21) # adjust xlim to fit labels

# Set bar labels
for i in range(0, len(ax.containers), 1):
    ax.bar_label(ax.containers[i], padding=5, fontsize=11)

ax.legend(title='Platform',
          fontsize=12, title_fontsize=13, borderpad=0.75, labelspacing=0.75)

plt.show()
```



Looking at how the diversity of the top three has changed over time, a sharp increase is seen in the below time series, especially for Netflix. The platform has caught up to Cartoon Network, overtaking it in terms of the number of characters in the last few years (an exception being 2021).

In [133…

```
df_shows_with_plat = df_characters.join(df_plat, on=['show_id'], how='right')
df_shows_with_plat.sort_index(inplace=True)
dat = df_shows_with_plat.reset_index(drop=False)
dat = dat.dropna(subset=['platform_id'])

dat = dat.astype({"platform_id": int})
dat['Year Confirmed'] = dat['Year Confirmed'].dt.year

dat = dat.set_index(['Year Confirmed'], drop=True).sort_index()
dat = dat.set_index(['platform_id'], append=True).sort_index().drop(axis=1, l

dat = dat.join(df_platforms['Name'], on='platform_id', rsuffix='_platform')
dat = dat.set_index(['character_id'], append=True).sort_index()
```

In [134…

```
dat = dat.reset_index('character_id') \
        .groupby(['Name_platform', 'Year Confirmed']).count()[['character_id'
        .loc[(['Netflix', 'Cartoon Network', 'Disney Channel'], slice(None))]
        .rename(mapper=dict(character_id = 'character_count'), axis=1) \
        .unstack(level=0).droplevel(axis=1, level=0)
```

In [135…

```
dat.reset_index(drop=False, col_level=1, inplace=True)
dat.sort_index(inplace=True)
dat.index = list(dat.index)
dat.index.name = 'index'
dat.rename_axis(None, axis=1, inplace=True) # Remove column index name
```

In [136…

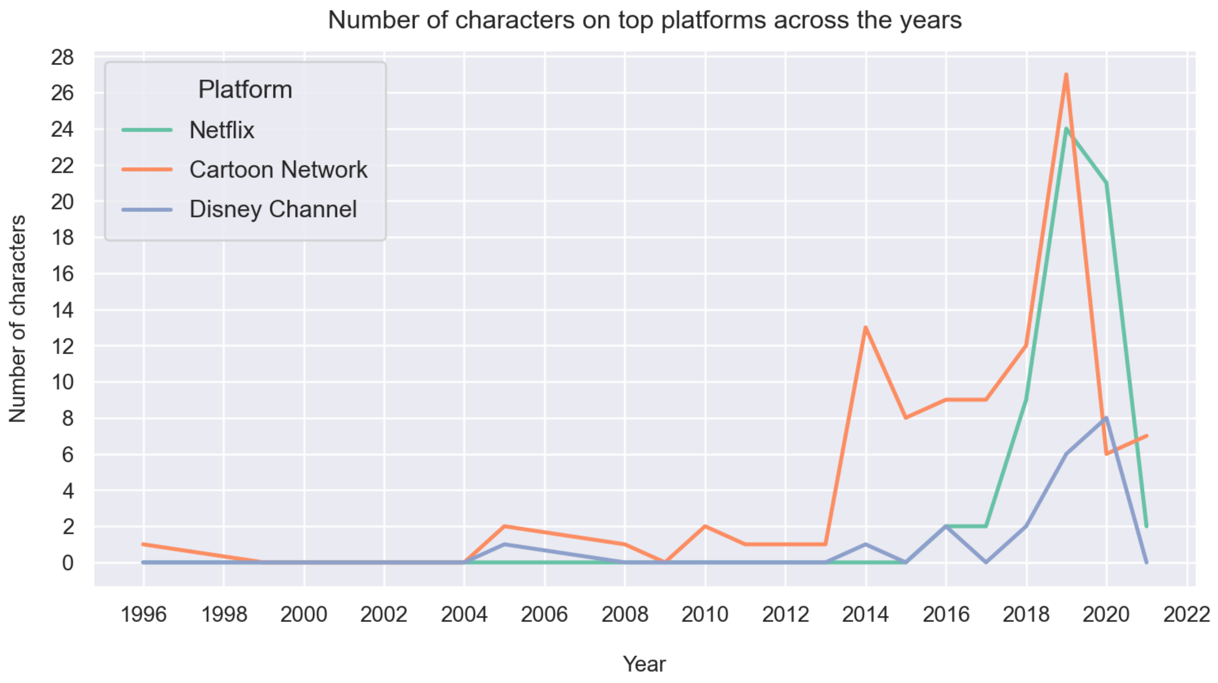
```
dat.head(5)
```

Out[136…

index	Year Confirmed	Netflix	Cartoon Network	Disney Channel
0	1996	0	1	0
1	1999	0	0	0
2	2004	0	0	0
3	2005	0	2	1
4	2008	0	1	0

In [142…

```
sns.set_theme(palette=sns.color_palette("Set2", 3))

fig, ax = plt.subplots(figsize=(10, 5))

for column in dat.drop('Year Confirmed', axis=1):
    x = dat['Year Confirmed']
    y = dat[column]
    ax.plot(dat['Year Confirmed'], dat[column],
            marker='', linewidth=2, label=column)

    # Smooth curve
    #Resource: (Devashish, 2021)
    X_Y_Spline = make_interp_spline(x, y)
    # Returns evenly spaced numbers
    # over a specified interval.
    X_ = np.linspace(x.min(), x.max(), 300)
    Y_ = X_Y_Spline(X_)
    #ax.plot(X_, Y_, marker='', linewidth=2, label=column)

ax.legend(title='Platform', labels=['Netflix', 'Cartoon Network', 'Disney Cha
ax.set_title('Number of characters on top platforms across the years', pad=11
ax.set_xlabel('Year', fontsize=11, labelpad=15)
ax.set_ylabel('Number of characters', fontsize=11, labelpad=10)
ax.set_xticks(np.arange(dat['Year Confirmed'].min(), dat['Year Confirmed'].ma
ax.set_yticks(np.arange(0, 30, 2))
plt.show()
```

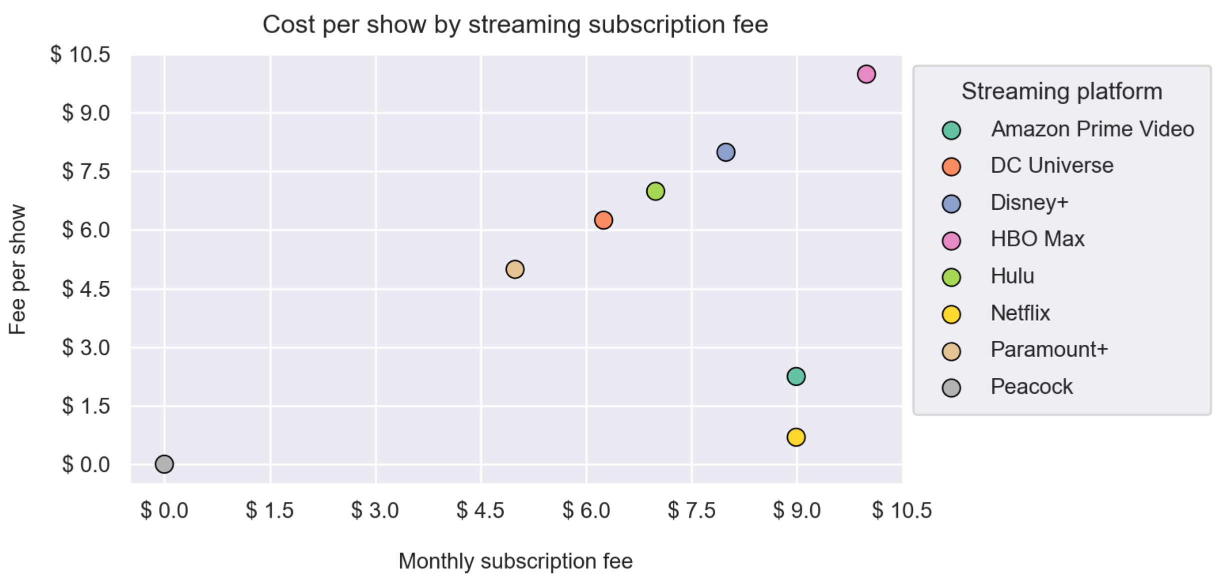


Lastly, an interesting aspect to explore are the subscription fees of the streaming platforms. Since the specific fees vary, a way of comparing the platforms is calculating the fee per show. Then, this "cost per show" is visualised on a scatterplot, clearly communicating a pattern - the higher the subscription, the higher the cost per show, or, in other words: more expensive streaming platforms tend to have less children's programming that includes LGBTQIA+ storylines. Netflix and Amazon Prime Video are an exception, standing out as having more LGBTQIA+ inclusive animation based on their price.

In [216…

```
fig, ax = plt.subplots(figsize=(7, 4))

# Calculate the price per show (monthy fee-show count ratio)
x = df_platform_shows['min_monthly_streaming_fee']
y = df_platform_shows['min_monthly_streaming_fee']/df_platform_shows['show_co

sns.scatterplot(ax=ax, x=x,
                     y=y,
                     hue=df_platform_shows['Name'],
                     hue_order=['Amazon Prime Video', 'DC Universe', 'Disney+
                     s=80, palette=sns.color_palette("Set2", 8), edgecolor='b
               )

ax.set_title('Cost per show by streaming subscription fee', pad=11, fontsize=
ax.set_xlabel('Monthly subscription fee', fontsize=11, labelpad=15)
ax.set_ylabel('Fee per show', fontsize=11, labelpad=10)

# Update ticks and tick labels, adding $ to each
Y_ticks = np.arange(y.min(), y.max()+2, 1.5)
Y_tick_labels = []
for tick in Y_ticks:
    Y_tick_labels.append("$ "+ str(tick))
ax.set_yticks(Y_ticks)
ax.set_yticklabels(Y_tick_labels)

X_ticks = np.arange(x.min(), x.max()+1, 1.5)
X_tick_labels = []
for tick in X_ticks:
    X_tick_labels.append("$ " + str(tick))
ax.set_xticks(X_ticks)
ax.set_xticklabels(X_tick_labels)
```

```python
# Changing the legend marker edge color
# Resource: (Alex, 2021)
# Get the legend handles
handles, labels = ax.get_legend_handles_labels()

# Iterate through the handles and update their attributes
for handle in handles:
    handle.set_edgecolor("black")
    handle.set_linewidth(0.75)
    handle.set_sizes([80])

# Use `ax.legend` to set the modified handles and labels
lgd = ax.legend(
    handles,
    labels,
    title='Streaming platform',
    bbox_to_anchor=(1.0, 1.0),
    fontsize=11, title_fontsize=12, borderpad=0.75, labelspacing=0.75
)

plt.show()
```



Conclusion

Based on the conducted exploration, it is clear that children't US Television is more diverse now than ever, with creators incorporating LGBTQIA+ storylines right at the beginning and being (mostly) open about it. Yet, some groups are still underrepresented, sometimes included in only one way (e.g. nonbinary characters are almost exclusevly non-human). Moreover, these shows are not widely accessible; most require a payment/subscription. Largely, these findings correspond to other reports in terms of race, dissability, trans, bi+ and asexual identities representation, with nonbinary identities being an exception of whom there are more in childern's TV than in the general media.

Possible implications and impact of this exploration, in addition to those mentioned in the introduction, would be further analysis of the topic - analysing the characters from other perspectives. For instance, seeing whether the characters are heroes or villains, how old they are in the story and whether the creators of the shows are queer too. Moreover, this sort of analysis could be performed on non-american and non-english-speaking shows.

Regarding the process of the exploration, I learned a lot through trial and error. Scraping data from a dynamic web application, which I've never done before, was a bit tricky, but satisfying in the end. I believe the data was explored thoroughly from many angles and from multiple sources. I considered human visual perception when constructing the plots such that most don't require that much visual processing and lean on first-order features such as shape, motion and colour. Thus, they are read intuitively and effective at communicating the findings. The colour choices were carefully considered, too - color schemes are appropriate for the type of data and do not fail at communicating the findings in case of colour blindness.

Parts of the process that could be improved are the data processing; there was some trial and error when grouping observations and (string, data type) manipulation, especially in terms of dataframe views vs. copies due to not knowing how some specific parts of the libraries work. Adding multiple levels of variables with hue in Seaborn were also tricky, due to not being aware how the hue parameter affects the grouping of data, at first. All in all, I think all of the aims of the explorations were achieved.

References

Alex. (2021, July 1). *How to add edge line to markers on the legend using matplotlib/seabornn.* Stack Overflow. Retrieved January 16, 2022, from https://stackoverflow.com/questions/68094110/how-to-add-edge-line-to-markers-on-the-legend-using-matplotlib-seabornn

Collins. (2010). Pay television. *Collins dictionary.*

Collins. (2010). Streaming service. *Collins dictionary.*

Craig, S. L., McInroy, L., McCready, L. T., & Alaggia, R. (2015). Media: A catalyst for resilience in lesbian, gay, bisexual, transgender, and Queer Youth. *Journal of LGBT Youth, 12*(3), 254–275. https://doi.org/10.1080/19361653.2015.1040193

Devashish. (2021, February 2). *How to plot a smooth curve in matplotlib?* GeeksforGeeks. Retrieved January 16, 2022, from https://www.geeksforgeeks.org/how-to-plot-a-smooth-curve-in-matplotlib/

GLAAD Media Institute. (2021). (rep.). *Where We Are on TV.* Retrieved January 12, 2022, from https://www.glaad.org/sites/default/files/GLAAD%20-%20202021%20WHERE%20WE%20ARE%20ON%20TV.pdf.

Hall, W. J., Dawes, H. C., & Plocek, N. (2021). Sexual orientation identity development milestones among lesbian, gay, bisexual, and queer people: A systematic review and meta-analysis. *Frontiers in Psychology, 12.* https://doi.org/10.3389/fpsyg.2021.753954

Hunter, J., Dale, D., Firing, E., Droettboom, M., & The Matplotlib development team. (n.d.). *Matplotlib 3.5.1 documentation.* Matplotlib documentation - Matplotlib 3.5.1 documentation. Retrieved January 13, 2022, from https://matplotlib.org/stable/index.html

Hunter, J., Dale, D., Firing, E., Droettboom, M., & The Matplotlib development team. (n.d.). *Choosing colormaps in matplotlib.* Choosing Colormaps in Matplotlib - Matplotlib 3.5.1

documentation. Retrieved January 13, 2022, from https://matplotlib.org/stable/tutorials/colors/colormaps.html

Hunter, J., Dale, D., Firing, E., Droettboom, M., & The Matplotlib development team. (n.d.). *Boxplots.* Boxplots - Matplotlib 3.5.1 documentation. Retrieved January 14, 2022, from https://matplotlib.org/stable/gallery/statistics/boxplot_demo.html#sphx-glr-gallery-statistics-boxplot-demo-py

Jones, A. (2020, November). How to Scrape Dynamic Web pages with Selenium and Beautiful Soup. Retrieved December 10, 2021, from https://towardsdatascience.com/how-to-scrape-dynamic-web-pages-with-selenium-and-beautiful-soup-fa593235981.

jsgounot, & Jed. (2019, February 1). *Changing width of bars in bar chart created using seaborn.factorplot.* Stack Overflow. Retrieved January 17, 2022, from https://stackoverflow.com/questions/34888058/changing-width-of-bars-in-bar-chart-created-using-seaborn-factorplot

Montz, B., & Solomon, M. (2021, September 20). *Acronyms explained*. OutRight Action International. Retrieved January 17, 2022, from https://outrightinternational.org/content/acronyms-explained

Mwaskom. (2016). *Does Boxplot Support Alpha / transparency? · issue #979 · MWASKOM/Seaborn.* GitHub. Retrieved January 14, 2022, from https://github.com/mwaskom/seaborn/issues/979

Pandas documentation. pandas documentation - pandas 1.3.5 documentation. (n.d.). Retrieved January 12, 2022, from https://pandas.pydata.org/docs/index.html

The selenium browser automation project. Selenium. (n.d.). Retrieved December 15, 2021, from https://www.selenium.dev/documentation/

Sexuality: A few definitions. Brook. (2021, February 17). Retrieved January 17, 2022, from https://www.brook.org.uk/your-life/sexuality-a-few-definitions/

Shahnoza. (2021, February 1). *Python - how to change autopct text color to be white in a pie chart?* Stack Overflow. Retrieved January 14, 2022, from https://stackoverflow.com/questions/27898830/python-how-to-change-autopct-text-color-to-be-white-in-a-pie-chart/54149203

Statistical Data Visualization. seaborn. (n.d.). Retrieved January 13, 2022, from https://seaborn.pydata.org/index.html

TV Parental Guidelines Monitoring Board. (n.d.). *Understanding the TV Ratings and Parental Controls.* Retrieved January 12, 2022, from http://www.tvguidelines.org/resources/TV_Parental_Guidelines_Brochure.pdf.

Webdriver for Chrome - downloads. ChromeDriver. (n.d.). Retrieved December 10, 2021, from https://chromedriver.chromium.org/downloads

Webster, N. (1977). Broadcast. The Merriam-Webster Dictionary. Pocket Books.

White, A., & Chik, K. (2021). *We created the first-ever searchable database of 259 LGBTQ characters in cartoons that bust the myth that Kids Can't Handle Inclusion.* Insider Database: 259 LGBTQ characters in kids' cartoons. Retrieved December 27, 2021, from https://www.insider.com/lgbtq-cartoon-characters-kids-database-2021-06

Zelazny7. (2020, November 11). *Creating a new column based on if-elif-else condition.* Stack Overflow. Retrieved January 13, 2022, from https://stackoverflow.com/questions/21702342/creating-a-new-column-based-on-if-elif-else-condition

Appendices

Webscraping script

Also available at Gitlab notebook and in the ZIP files.

Please **don't** run this code here!

Environment setup

In [ ]:

```
# Import libraries
from bs4 import BeautifulSoup
import os
import time
import pandas as pd

from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.chrome.options import Options
# Instantiate an Options object
# and add the "--headless" argument
opts = Options()
opts.add_argument("--headless")
opts.add_argument("--start-maximized")

# Set the path to the Brave browser’s location
opts.binary_location= '/Applications/Brave Browser.app/Contents/MacOS/Brave B

# Set the location of the webdriver
chrome_driver = './chrome-driver/chromedriver'

# Instantiate a webdriver and get the website
driver = webdriver.Chrome(options=opts, executable_path=chrome_driver)
driver.set_window_size(1440, 1167)
driver.get('https://www.insider.com/lgbtq-cartoon-characters-kids-database-20

# Check window size
width = driver.get_window_size().get("width")
print(f"Width: {width}")
height = driver.get_window_size().get("height")
print(f"Height: {height}")
```

Displaying all characters

In [ ]:
```
 Find and locate the "to-down" button
to_down = WebDriverWait(driver, timeout=5).until(EC.visibility_of_element_loc
driver.execute_script("arguments[0].scrollIntoView(true);", to_down)

# Wait until it is clickable
to_down_el = WebDriverWait(driver, timeout=5).until(EC.element_to_be_clickabl
# Click the button
webdriver.ActionChains(driver).move_to_element(to_down_el).click().perform()

# Find and locate the "VIEW CHARACTERS" button
view_chars = WebDriverWait(driver, timeout=5).until(EC.visibility_of_element_

# Move the mouse to the element and click the button to display all character
webdriver.ActionChains(driver).move_to_element(WebDriverWait(driver, timeout=
```

Getting character data

In [ ]:
```
#Get characters
all_characters = driver.find_elements_by_class_name('character')
if len(all_characters) == 259:
    print(f"All {len(all_characters)} characters found.")

#Get all character buttons
all_char_buttons = []
for char in all_characters:
    btn = char.find_element_by_tag_name('button')
    all_char_buttons.append(btn)
print(f"All {len(all_char_buttons)} character buttons found.")
```

Helper functions

In [ ]:
```
def open_character_pop_up(character_btn):
    '''
        Clicks the character button to open a character pop up. Returns the p
    '''
    print(f"Moving to {character_btn.text} button.")
    action = ActionChains(driver)
    action.move_to_element(character_btn).click().perform()

    # Optional saving of screenshot
    #driver.save_screenshot(f"screen_{str(character_btn.text)}.png")

    char_page_source = driver.page_source
    char_soup_page = BeautifulSoup(char_page_source)
    time.sleep(5) # Sleep for 5 seconds so that soup passes the page
    char_wrapper = char_soup_page.find_all(class_="character-pop-up-wrapper")

    # Click anywhere in the pop up to close it
    #wrapper = WebDriverWait(driver, timeout=5).until(EC.element_to_be_clicka
    close_el = WebDriverWait(driver, timeout=5).until(EC.visibility_of_all_el
    print("Closing the pop up.")
    close_el[0].click()

    if(len(char_wrapper) > 0):
        print(f"Returning {character_btn.text} pop up.")
        print()
        return char_wrapper[0]
    else:
        print(f"No character pop up found. Returning 0.")
        return 0

def get_character_data(character_popup):
    '''
        Gets all the character data from the pop up. Returns a dictionary of 
    '''
    char_data = dict() # Create an empty dictionary for the character data
    name = character_popup.find('h3').text
    print(f"Adding data for {name}.")
    char_data['Name'] = name
    show = character_popup.find('h5').text
    char_data['Show'] = show
    stats = character_popup.find_all(class_='stat')
    for stat in stats:
        txt = stat.text.split(':')
        txt[1] = txt[1].strip()
        # append things to the dict
        char_data[txt[0]] = txt[1]
        #print(txt)
    print(f"Returning {name} dict.")
    print()
    print()
    return char_data

def collect_data(char_btns):
    '''
        Iterates through a list of all character buttons, calling helper func
        collect the data and adding it to the main data dictionary.

        The character data gets written into a csv in batches of 10 (or as se
    '''
    # Create an empty dictionary to which we will add all character data
    data_dict = {}
    batch = 9

    # As we iterate through the character pop ups, we add the collected data
    # for char_btn in char_btns:
    #     pop_up = open_character_pop_up(char_btn)
    #     if(pop_up != 0):
    #         char_dict = get_character_data(pop_up)
    #         data_dict[char_btns.index(char_btn)] = char_dict

    for i, char_btn in enumerate(char_btns):
        pop_up = open_character_pop_up(char_btn)
        if(pop_up != 0):
            char_dict = get_character_data(pop_up)
            data_dict[i] = char_dict
            if(i != 0 and i%batch == 0): # Save every batch
                print(f"{i%9}th batch: Adding next {batch} characters...")
                write_data(data_dict)
            elif(i == (len(char_btns)-1)):
                print(f"Writting the last version of the data.")
                write_data(data_dict) # Save the last version of the datafram

    print(f"Collected all character data. Quitting the browser.")
    # Close the browser
    driver.quit()

def write_data(_dict):
    '''
        Converts the dictionary to a dataframe. Writes the dataframe to a csv
        Manages overwritting old csv files with new data.
    '''
    if not os.path.exists('./raw_data'):
        os.mkdir('./raw_data')
    csv_path = './raw_data/data.csv'
    if not os.path.exists(csv_path): # If we don't already have the file
```

```
        print('Creating ' + csv_path + 'and the initial dataframe...')
        df = pd.DataFrame(_dict)
        df = df.transpose()

        df.to_csv(csv_path, index=True)
    else:
        print(f"Overwriting {csv_path} with new characters...")
        df = pd.DataFrame(_dict)
        df = df.transpose()
        df.to_csv(csv_path, index=True)
    print(f"Number of characters collected: {len(df)}")
    return df
```

Running the data collection

In [ ]:

```
df = collect_data(all_char_buttons)
```

Word Count

The following code will count the number of words in Markdown cells. Code cells are not included.

- `Main word count` is the number of words in the main body of the text, *excluding* references or appendices.
- `References and appendices word count` is the number of words in any references or appendices.

Only `Main word count` relates to the assignment word limit. There is no limit to the number of words that can be included in references or appendices. Please note that appendices should only be used to provide context or supporting information. *No marks will be directly awarded for material submitted in appendices*.

Important:

- Please do not modify the word count code!
- To exclude references from your word count **you must** have a cell that starts with the text `## References`. Everything below this cell will not count towards the main word count.
- If you are submitting additional material as appendices **you must** have a cell that starts with the text `## Appendices`. Everything below this cell will not count towards the main word count. If you do not have any appendices you can delete the `## Appendices` cell.
- Code comments should only be used to explain details of the implementation, not for discussing your findings. All analysis commentary **must** be written in Markdown cells. *No marks will be awarded for analysis discussion submitted as comments in code cells*.

In [1]:

```
%%js

// Run this cell to update your word count.

function wordcount() {
    let wordCount = 0
    let extraCount = 0
    let mainBody = true

    let cells = Jupyter.notebook.get_cells()
    cells.forEach((cell) => {
        if (cell.cell_type == 'markdown') {
            let text = cell.get_text()
            // Stop counting as main body when get to References or Appendice
            if (text.startsWith('## References') ||
                text.startsWith('## Appendices')) {
                mainBody = false
            }
            if (text.startsWith('## Word Count')) {
                text = ''
            }
            if (text) {
                let words = text.toLowerCase().match(/\b[a-z\d]+\b/g)
                if (words) {
                    let cellCount = words.length
                    if (mainBody) {
                        wordCount += cellCount
                    } else {
                        extraCount += cellCount
                    }
                }
            }
        }
    })
    return [wordCount, extraCount]
}

let wc = wordcount()
element.append(`Main word count: ${wc[0]} (References and appendices word cou
```