Japanime

Data Science Project

Ben Nadav & Amit Aviv

Introduction

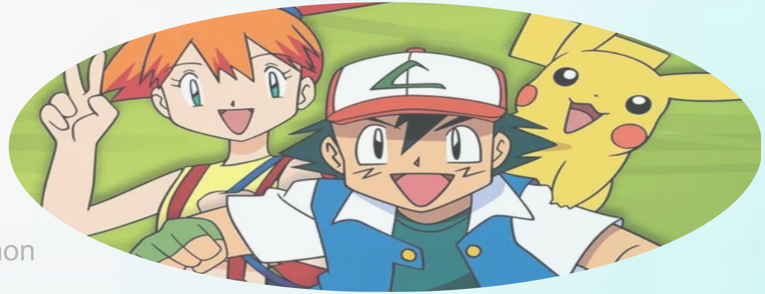
Senku Ishigami, Dr. Stone

In this project we will introduce the topic of Anime.

Anime, as it sounds, is an animated series originated in Japan, and it is extremely popular there. The anime has become a global phenomenon for all ages, due to its unique animations, designs, sounds, and creative stories.

There are many genres of anime, such as Action, Drama, Comedy, Romance, Adventure and so on…

Our purpose in this project, is to collect data from existing Anime, and use it to build a learning model that can analyze new shows and decide whether or not they are prone to success.



Misty, Ash & Pikachu, Pokemon

External Resources

- Anilist - The site we got our data from.
- StackOverflow - we used a few lines of code (mostly for Selenium), and adjusted them to our needs.
- Google Pictures - for the picture decorations in this presentation.
- Our Code (Github) - Amit Aviv , Ben Nadav

Megumi Fushiguro & Yuji Itadori & Satoru Gojo, Jujutsu Kaisen

Yugi Mutou, Yu Gi Oh!

Steps - Our Process

0. Forming a research question (as explained in our introduction).

1. Getting our data through Crawling (Slides 5-9).
2. Data Handling (Slides 10-13).
3. Exploratory Data Analysis [EDA] (Slides 14-25).
4. Building and Training a Machine Learning model (Slides 26-27).

karasuno, Haikyuu!!

Step 1 - Crawling

Our Dataframe:

aniNames = The titles of the shows.

formats = TV, Movie, Special, etc…

number of episodes = The length of the show(1 = Movie/Special).

episode duration = A single episode’s time span.

status = Finished, On Going, etc..

studios = The producing studio.

favorites= Amount of users who ranked the series as their favorite.

popularity= Sum of Completed,Current,Planning,Paused,Dropped.

genres = Comedy, Romance, Action, and many more..

mean score = A provided means of the users score distribution.

source = Some series are based on Manga/Light Novels, etc…

creator = The author/writer of the original story.

completed = Amount of users who finished watching/Caught up.

current = Amount of users who are currently watching the series.

planning = Amount of users who intend to watch the series.

paused = Amount of users who put their watching on hold.

dropped = Amount of users who quit watching the series.

17 columns overall

Monkey D. Luffy, One Piece

Step 1 - Crawling

Our Process:

- We began by creating a function ('load_soup_object') that returns a soup object, and a **list of empty lists**. Each of these lists is a dataset which will later become a column in our dataframe.

- Next, we built our main **crawling function** called 'Scrap', in which we used **BeautifulSoup** to gain a soup object of the main site, extract our relevant links of each Anime, and use our useful 'load_soup_object' to gain a soup object for each of these specific links in a For loop. Inside this same loop, we extracted all of our data. At the end of each iteration, we checked the size of all our lists to see if they're the same. If not, this means there has been a **missing value**, and so we filled these slots with NaN values.

- The reason we used a scraping function, was because we ran into an issue with the website we chose. As it turned out, the site had a dynamic self updating, which means we could only scrape up to 20 Anime at a time, and the next 20 only appear if we scrolled down the page. For this reason we used **Selenium**'s web driver on a self-defined time limit, to automatically scroll down the page and load it fully, so we could later use our scraping function on the entire page instead of just the 20 first shows.

- Finally, we checked the final length of each of the lists sizes, to make sure they are all even and ready to be put in a Panda's Dataframe. We printed our final Dataframe, and saved it to a **CSV file**.

Our Crawling Function

```python
def scrap(url):
    #ani_soup = load_soup_object('https://anilist.co/search/anime/popular')
    ani_soup = BeautifulSoup(url)

    count = 0
    mtag = ani_soup.find_all("div",attrs={"class":"media-card"})
    for t in mtag:
        count+=1
        anime_name = t.find('a',attrs={"class":"title"}).get_text().replace('\n','').replace('\t','')
        aniNames.append(anime_name)

        ani_url = 'https://anilist.co' + t.find('a',attrs={"class":"title"}).get('href')
        soup = load_soup_object(ani_url)
        anime_tag = soup.find_all("div",attrs={"class":"data-set"})
        for x in anime_tag:
            try:
                div_type = x.find("div",attrs={"class":"type"}).get_text()
                div_value = x.find("div",attrs={"class":"value"}).get_text()
            except:
                continue

            if(div_type == 'Format'):
                formats.append(div_value.replace('\n',''))

            if(div_type == 'Episodes'):
                number_of_episodes.append(div_value)

            if('Duration' in div_type):
                episode_duration.append(div_value)

            if(div_type == 'Status'):
                status.append(div_value)

            if(div_type == 'Mean Score'):
                mean_score.append(div_value.replace('%',''))
```

Appending lists for our DataFrame columns

```python
            if(div_type == 'Popularity'):
                popularity.append(div_value)

            if(div_type == 'Favorites'):
                favorites.append(div_value)

            if(div_type == 'Studios'):
                studios.append(x.find("a").get_text().replace("\n",""))

            if(div_type == 'Source'):
                source.append(div_value)

            if(div_type == 'Genres'):
                generes_string = ''
                for g in x.find_all("span"):
                    generes_string += g.get_text() + '|'
                generes_string = generes_string[:-1]
                genres.append(generes_string)


        status_tag = soup.find_all("div",attrs={"class":"status"})
        for l in status_tag:
            status_name = l.find("div",attrs={"class":"name"}).get_text()
            status_value = l.find("div",attrs={"class":"amount"}).get_text().replace("Users","").replace("\n","")

            if('Completed' in status_name):
                completed.append(status_value)

            if('Current' in status_name):
                current.append(status_value)

            if('Planning' in status_name):
                planning.append(status_value)

            if('Paused' in status_name):
                paused.append(status_value)
```

```python
    if('Dropped' in status_name):
        dropped.append(status_value)

roles = soup.find_all('div',attrs={"class":"view-staff"}) #role-card view-staff small
try:
    creator.append(roles[0].find('div',attrs={"class":"name"}).get_text().replace('\n',''))
except:
    pass

#after every anime check if there is any missing value and if it is missing, insert NaN
for i in range(17):
    if((len(arrayOfArrays[i])<count)):
        arrayOfArrays[i].append(np.nan)

time.sleep(1)
```

The end of the crawl function

Lelouch Lamperouge,
Code Geass

```python
#https://stackoverflow.com/questions/53701759/scroll-with-keys-page-down-in-selenium-python
driver=webdriver.Chrome(ChromeDriverManager().install())
driver.get("https://anilist.co/search/anime/popular")
timeout = time.time() + 60*20
while True:
    driver.find_element_by_tag_name('body').send_keys(Keys.PAGE_DOWN)
    time.sleep(3)
    if time.time() > timeout:
        #initialize_arrays()
        scrap(driver.page_source)
        break

#print(len(aniNames), len(formats), len(number_of_episodes), len(episode_duration),len(status),...

data = {'anime_name': aniNames, 'Format':formats, 'Number Of Episodes':number_of_episodes,
        "Episode Duration":episode_duration, "status":status, "Popularity":popularity,
        "favorites":favorites,"Studios":studios, "Genres":genres, "mean_score":mean_score,
        "source":source,"Creator":creator,'Completed':completed,'Current':current,
        'Planning':planning,'Paused':paused,'Dropped':dropped}

df = pd.DataFrame(data)
driver.quit()  # Close the browser
```

Our use of Selenium with the crawl function implementation

	anime_name	Format	Number Of Episodes	Episode Duration	status	Popularity	favorites	Studios	Genres
0	Shingeki no Kyojin	TV	25	24 mins	Finished	523588	47765	Wit Studio	Action|Drama|Fantasy|Mystery
1	DEATH NOTE	TV	37	23 mins	Finished	476020	35757	MADHOUSE	Mystery|Psychological|Supernatural|Thriller
2	Boku no Hero Academia	TV	13	24 mins	Finished	472686	23800	bones	Action|Adventure|Comedy
3	Kimetsu no Yaiba	TV	26	24 mins	Finished	470002	40520	ufotable	Action|Adventure|Drama|Fantasy|Supernatural
4	HUNTER×HUNTER (2011)	TV	148	24 mins	Finished	434820	56940	MADHOUSE	Action|Adventure|Fantasy
5	One Punch Man	TV	12	24 mins	Finished	397841	19414	MADHOUSE	Action|Comedy|Sci-Fi|Supernatural

Todoroki Shoto, My Hero Academia

mean_score	source	Creator	Completed	Current	Planning	Paused	Dropped
85%	Manga	Hajime Isayama	438199	31047	36867	9198	8277
84%	Manga	Tsugumi Ooba	370348	26999	47094	16831	14748
79%	Manga	Kouhei Horikoshi	389491	29221	39180	7503	7291
85%	Manga	Koyoharu Gotouge	363558	41238	48696	9883	6627
90%	Manga	Yoshihiro Togashi	259415	65431	72202	28874	8898
83%	Manga	ONE	333152	15200	35839	7669	5981

A glimpse into our dataframe

Size: 17 X 6080 (before data handling)

Yuno Gasai, Mirai Nikki

Step 2 - Data Handling

We began by examining our data

In [3]:
```
anime_df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6080 entries, 0 to 6079
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   anime_name          6080 non-null   object 
 1   Format              5935 non-null   object 
 2   Number Of Episodes  5780 non-null   float64
 3   Episode Duration    5720 non-null   object 
 4   status              5948 non-null   object 
 5   Popularity          5948 non-null   float64
 6   favorites           5928 non-null   float64
 7   Studios             4923 non-null   object 
 8   Genres              5860 non-null   object 
 9   mean_score          5736 non-null   object 
 10  source              5701 non-null   object 
 11  Creator             5657 non-null   object 
 12  Completed           5948 non-null   float64
 13  Current             5948 non-null   float64
 14  Planning            5948 non-null   float64
 15  Paused              5948 non-null   float64
 16  Dropped             5948 non-null   float64
dtypes: float64(8), object(9)
memory usage: 807.6+ KB
```

In [4]:
```
anime_df_copy.describe()
```

Out[4]:

	Number Of Episodes	Popularity	favorites	Completed	Current	Planning	Paused	Dropped
count	5780.000000	5948.000000	5928.000000	5948.000000	5948.000000	5948.000000	5948.000000	5948.000000
mean	13.279066	20637.395763	575.748650	12415.424344	1139.655683	5811.624916	572.656187	698.034633
std	33.522225	41634.222731	2426.701066	29271.735961	4329.280999	9381.100630	1550.693438	1554.324971
min	1.000000	1225.000000	1.000000	0.000000	2.000000	65.000000	0.000000	0.000000
25%	1.000000	2608.750000	16.000000	1206.750000	64.000000	915.750000	41.000000	38.000000
50%	12.000000	6337.500000	52.000000	3248.500000	169.000000	2064.000000	109.000000	111.000000
75%	13.000000	19124.750000	222.000000	10344.750000	690.000000	6267.500000	461.000000	622.000000
max	1787.000000	522270.000000	56845.000000	436995.000000	171725.000000	100161.000000	41227.000000	26245.000000

```python
#Erasing Rows with more than 5 NaN values.
anime_df_copy.dropna(axis = 0, thresh = 5, inplace=True)
anime_df_copy
```

Which brought us to this: 5948 rows × 17 columns

We figured that 4 or less NaN values
are still manageable for our purposes.

```python
# 1 - TV , 2 - Movie , 3 - Special , 4 - OVA , 5 - TV Short, 6 - ONA , 7 - Music . . . 0 - NaN (added later)
format_replace_map = {'TV':1, 'TV(Chinese)':1, 'TV(South Korean)':1, 'TV ':1,
                      'Movie':2, 'Movie(Chinese)':2, 'Movie(South Korean)':2, ' Movie':2,
                      'Special':3,
                      'OVA':4, 'OVA(South Korean)':4, 'OVA ':4, ' OVA':4,
                      'TV Short':5, 'TV Short(South Korean)':5,
                      'ONA':6, 'ONA(Chinese)':6, 'ONA(South Korean)':6, 'ONA(Doujin)':6, ' ONA':6,
                      'Music':7}

anime_df_copy.replace(format_replace_map, inplace=True)
```

```python
# 1 - Manga , 2 - Light Novel , 3 - Original , 4 - Visual Novel , 5 - Video Game , 6 - Novel , 7 - Other . . . 0 - NaN.
Source_replace_map = {'Manga':1, 'Light Novel':2, 'Original':3,
                      'Visual Novel':4, 'Video Game':5, 'Novel':6,
                      'Multimedia Project':7, 'Web Novel':7, 'Doujinshi':7,
                      'Live Action':7, 'Comic':7, 'Picture Book':7, 'Game':7, 'Anime':7, 'Other':7}

anime_df_copy.replace(Source_replace_map, inplace=True)
anime_df_copy
```

Shota Aizawa (a.k.a Eraser Head), My Hero Academia

We transformed the categorical string values of 'Formats' and 'source' into numbers.

```python
#Replacing String values in columns Mean Score and Episode Duration with Float values.
Episode_Duration_mins=[]
for ep_dur in anime_df_copy["Episode Duration"]:
    mins = 0
    itr = 0
    try:
        str_split = ep_dur.split(',')
        if(len(str_split)>1):
            str_split2 = str_split[itr].split(' ')
            mins += 60*int(str_split2[itr])
            itr += 1
        str_split2 = str_split[itr].split(' ')
        mins += int(str_split2[itr])
        Episode_Duration_mins.append(mins)
    except:
        Episode_Duration_mins.append(0)
#replacing NaNs with 0

anime_df_copy["Episode Duration"] = Episode_Duration_mins
```

cp0, One Piece

```python
# changing mean_score into a binned column. categories are 0,1,2,3,4,5,6,7,8,9,10.

anime_df_copy['mean_score'] = anime_df_copy['mean_score'].str.rstrip('%').astype('float')
anime_df_copy['mean_score'].fillna(0, inplace=True)

bins = [0,10,20,30,40,50,60,70,80,90,100]
labels = [1,2,3,4,5,6,7,8,9,10]

anime_df_copy['mean_score_binned'] = pd.cut(anime_df_copy['mean_score'], bins, labels=labels)

# replacing NaNs with 0 category
anime_df_copy.mean_score_binned = anime_df_copy.mean_score_binned.values.add_categories(0)
anime_df_copy.mean_score_binned = anime_df_copy.mean_score_binned.fillna(0)

# deleting the previous mean_score column (only from the copied df)
anime_df_copy = anime_df_copy.drop('mean_score', axis=1)
```

Our Mean Score and Episode Duration Columns both held strings, so we transformed them to numeric values.

Episode Duration was recalculated to be measured by minutes.

We put the values of mean score into bins 1-10 and added it as a new column to the dataframe, and erased the previous column.

Both columns’ NaN values were replaced with 0.

```python
#replacing remaining NaN values with XX for strings and 0 for Floats.
anime_df_copy.anime_name.fillna('XX', inplace = True)
anime_df_copy.status.fillna('XX', inplace = True)
anime_df_copy.Studios.fillna('XX', inplace = True)
anime_df_copy.Genres.fillna('XX', inplace = True)
anime_df_copy.Creator.fillna('XX', inplace = True)
```

```python
anime_df_copy.Popularity.fillna(0, inplace = True)
anime_df_copy.favorites.fillna(0, inplace = True)
anime_df_copy["Number Of Episodes"].fillna(0, inplace = True)
anime_df_copy.Completed.fillna(0, inplace = True)
anime_df_copy.Current.fillna(0, inplace = True)
anime_df_copy.Planning.fillna(0, inplace = True)
anime_df_copy.Paused.fillna(0, inplace = True)
anime_df_copy.Dropped.fillna(0, inplace = True)
anime_df_copy.Format.fillna(0, inplace = True)
anime_df_copy.source.fillna(0, inplace = True)

anime_df_copy
```

Then, we replaced the NaN values with 'XX' in categorical columns, and 0 in numeric columns.

```python
anime_df_copy.drop_duplicates()
```

Making sure there aren't duplicated rows.

```python
anime_df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5948 entries, 0 to 6079
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   anime_name          5948 non-null   object
 1   Format              5948 non-null   float64
 2   Number Of Episodes  5948 non-null   float64
 3   Episode Duration    5948 non-null   int64
 4   status              5948 non-null   object
 5   Popularity          5948 non-null   float64
 6   favorites           5948 non-null   float64
 7   Studios             5948 non-null   object
 8   Genres              5948 non-null   object
 9   source              5948 non-null   float64
 10  Creator             5948 non-null   object
 11  Completed           5948 non-null   float64
 12  Current             5948 non-null   float64
 13  Planning            5948 non-null   float64
 14  Paused              5948 non-null   float64
 15  Dropped             5948 non-null   float64
 16  mean_score_binned   5948 non-null   category
dtypes: category(1), float64(10), int64(1), object(5)
memory usage: 796.2+ KB
```

Finally we looked at our updated dataframe, and the info:

Naruto Uzumaki, Naruto

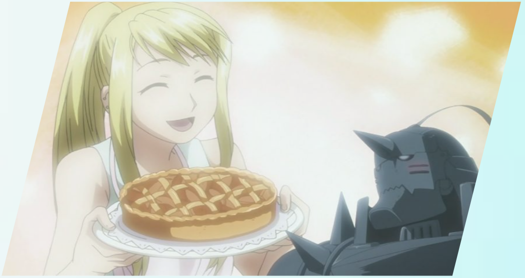
Step 3 - EDA (Visualizations)

We turned our Genres column into a secondary binary Data Frame, then summarized our data’s genres.

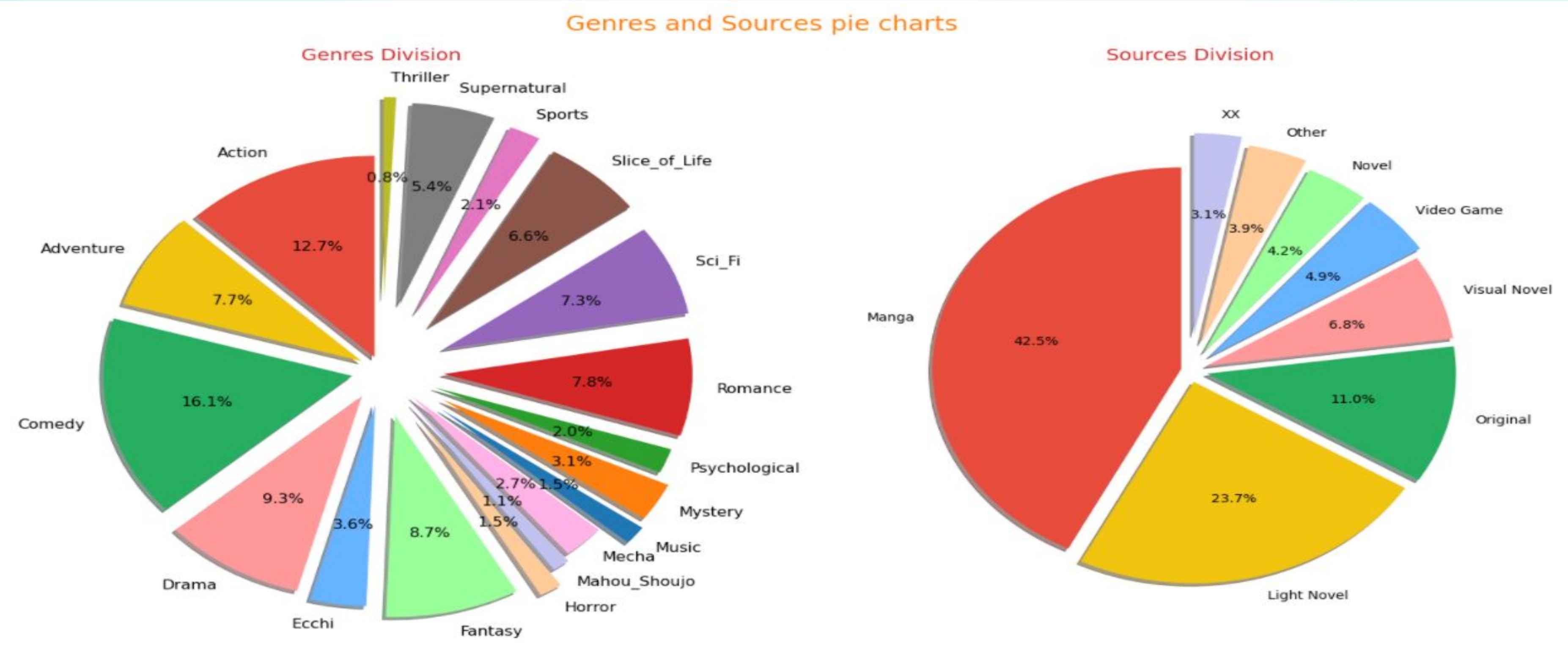
	Action	Adventure	Comedy	Drama	Ecchi	Fantasy	Horror	Mahou_Shoujo	Mecha	Music	Mystery	Psychological	Romance	Sci_Fi	Slice_of_Life	Spo
0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
3	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	
4	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5943	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5944	1	0	0	0	0	0	0	0	1	0	1	1	0	1	0	
5945	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	
5946	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
5947	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	

```
Action           2214
Adventure        1345
Comedy           2815
Drama            1615
Ecchi             634
Fantasy          1518
Horror            264
Mahou_Shoujo      199
Mecha             476
Music             259
Mystery           535
Psychological     344
Romance          1360
Sci_Fi           1269
Slice_of_Life    1156
Sports            359
Supernatural      944
Thriller          139
dtype: int64
```

Winry Rockbell & Alphonse Elric,
Fullmetal Alchemist Brotherhood

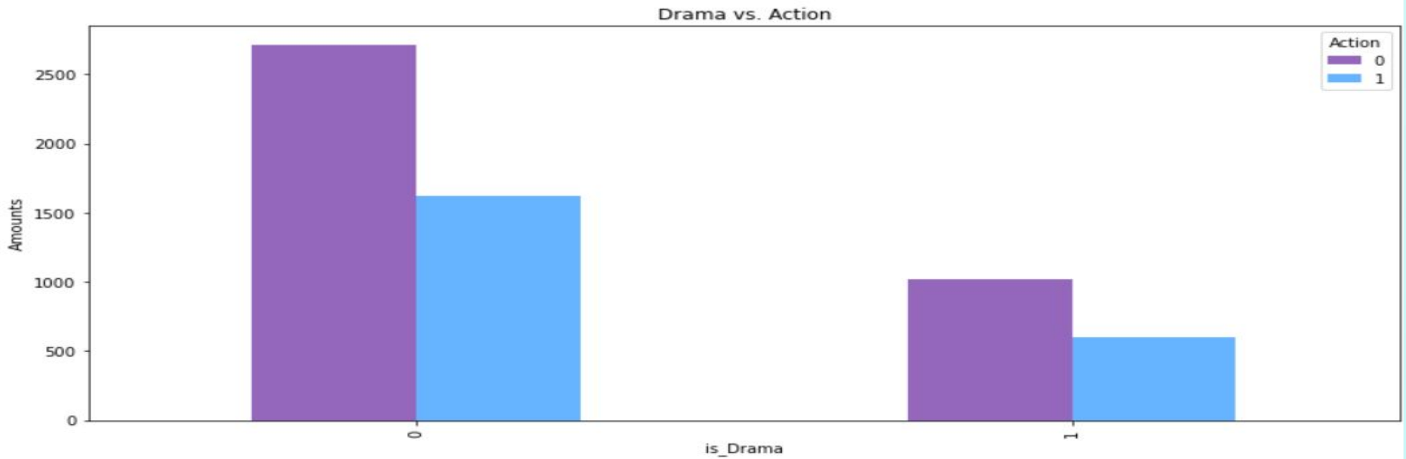


We used it to build our genre Pie chart,
and we also created a Sources pie chart:



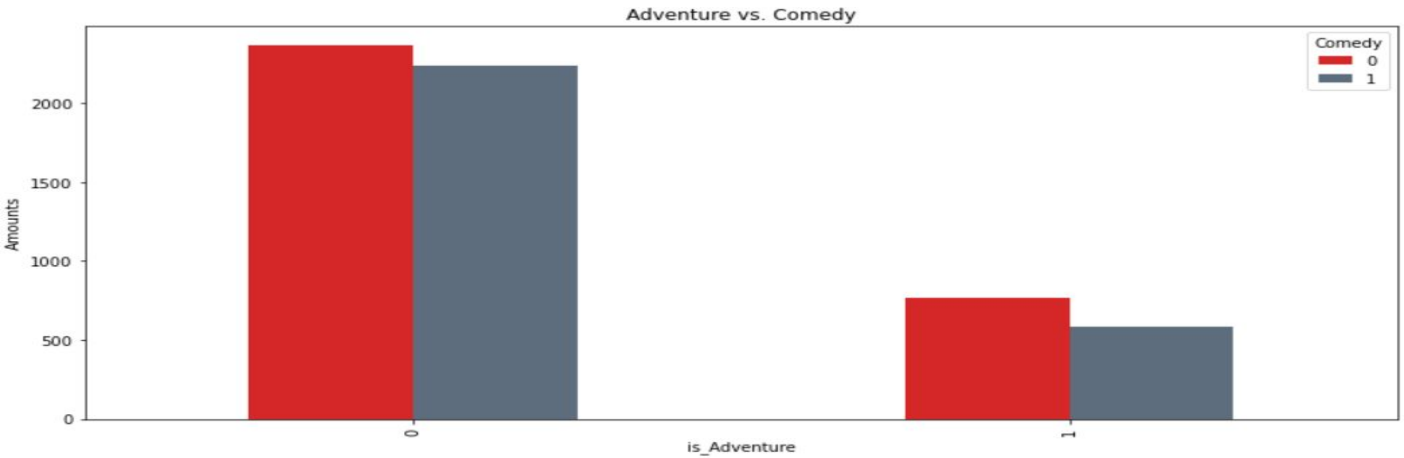
Next, we checked the correlation between genres: Drama vs Action, and Romance vs Comedy

Action	0	1
Drama		
0	2714	1619
1	1020	595



Crosstab

Comedy	0	1
Adventure		
0	2368	2235
1	765	580



```python
# Using Chi squared test to check if assumption H0 is valid,
# meaning both variables are not dependant on eachother.

from scipy.stats import chi2_contingency
chi2_contingency(drama_action_ct)

# According to our p-value - 0.733 > 0.05, H0 is indeed valid.
```

```
(0.11590861765648097,
 0.733514999682193,
 1,
 array([[2720.14492266, 1612.85507734],
        [1013.85507734,  601.14492266]]))
```

P-Value is **higher** than 0.05

```python
# Using Chi squared test to check if assumption H0 is valid,
# meaning both variables are not dependant on eachother.

from scipy.stats import chi2_contingency
chi2_contingency(adventure_comedy_ct)

# According to our p-value - 0.0005 < 0.05, H0 is not valid,
#and the variables ARE dependant on eachother.
```

```
(12.105939160664143,
 0.0005026147518160441,
 1,
 array([[2424.54589778, 2178.45410222],
        [ 708.45410222,  636.54589778]]))
```

P-Value is **lower** than 0.05

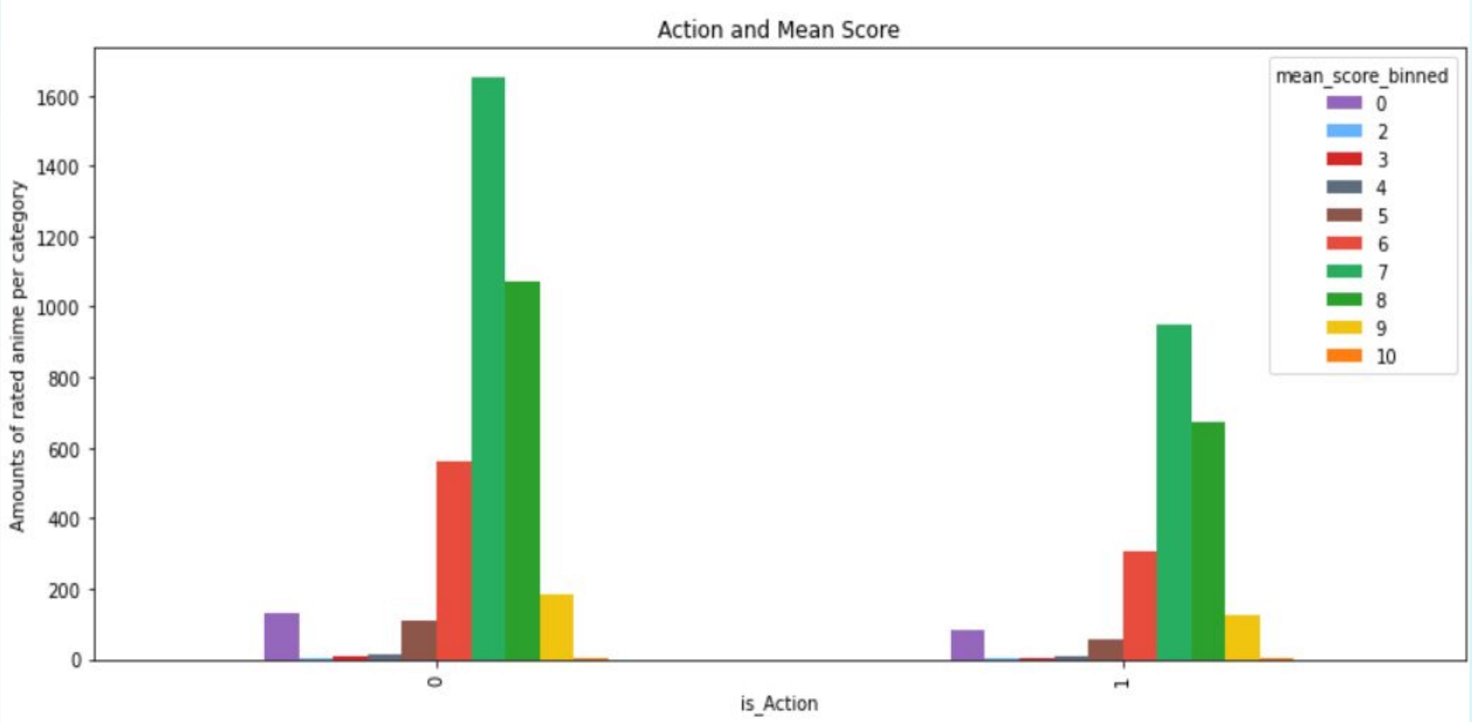
Our results show that Drama and Action are not dependant on each other, while Adventure and Comedy ARE dependant on each other.

Ochaco Uraraka, My Hero Academia

mean_score_binned	0	2	3	4	5	6	7	8	9	10
Action										
0	128	3	7	16	111	562	1652	1072	181	2
1	84	2	2	10	58	307	952	673	125	1

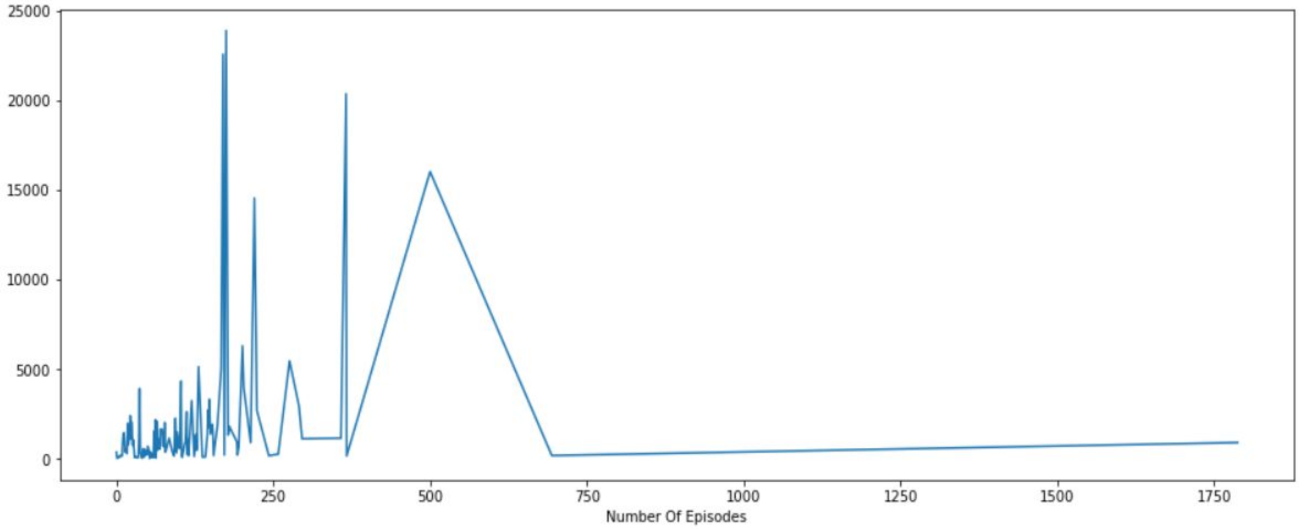
```
(6.950371342868021,
 0.6422865797250843,
 9,
 array([[1.33088097e+02, 3.13887021e+00, 5.64996638e+00, 1.63221251e+01,
         1.06093813e+02, 5.45535642e+02, 1.63472360e+03, 1.09546570e+03,
         1.92098857e+02, 1.88332213e+00],
        [7.89119032e+01, 1.86112979e+00, 3.35003362e+00, 9.67787492e+00,
         6.29061870e+01, 3.23464358e+02, 9.69276395e+02, 6.49534297e+02,
         1.13901143e+02, 1.11667787e+00]]))
```

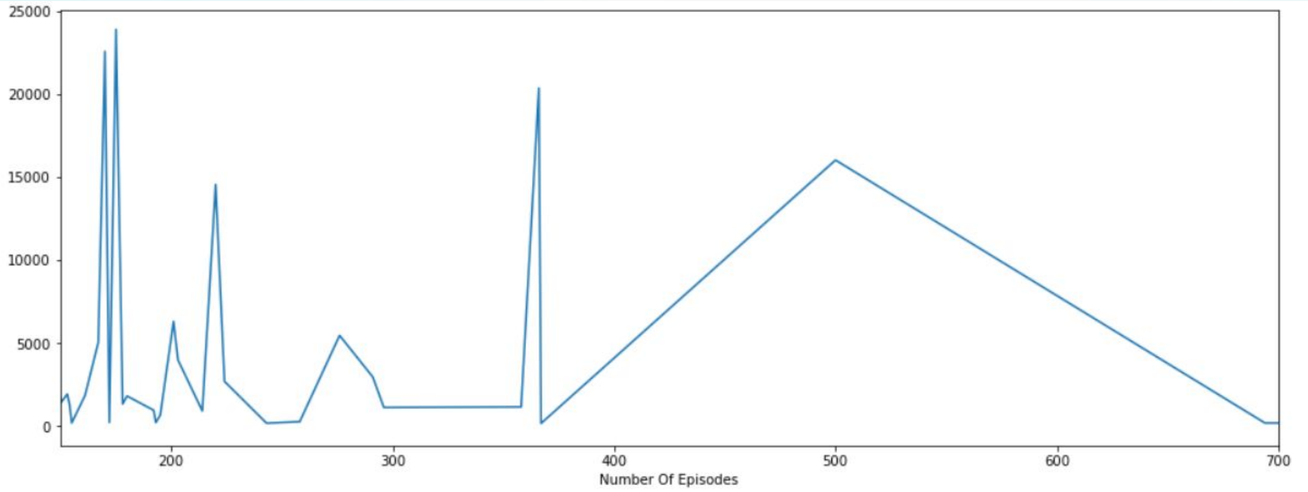
P-value > 0.05



We made one more chi square test between the Action genre and Mean Score to see if there is dependency.

it turns out they don’t depend on each other.

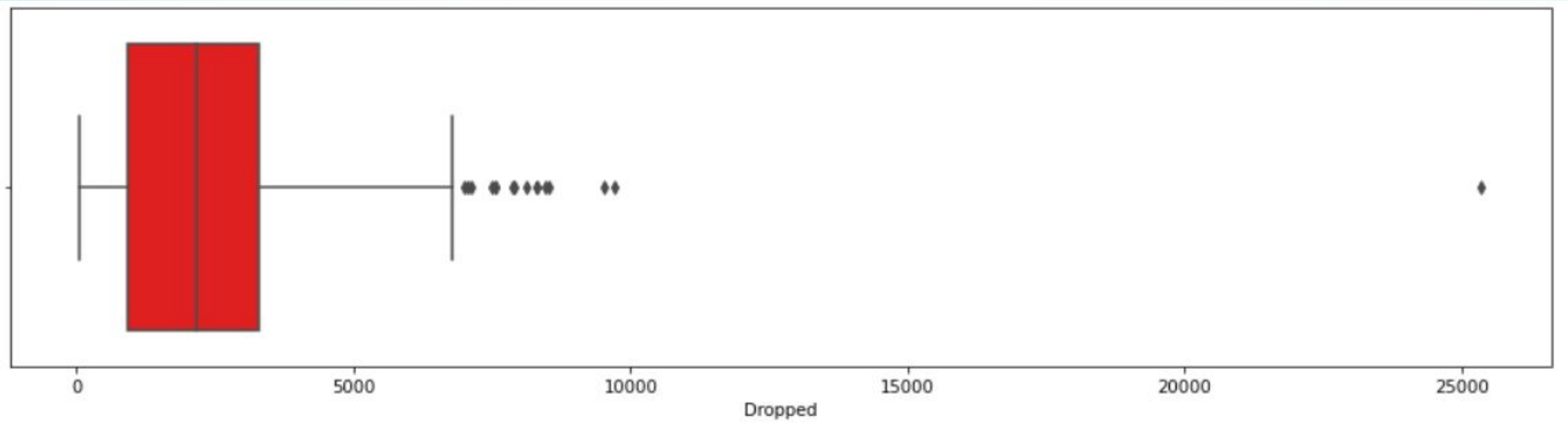




We made a line plot describing the Number of users who dropped the Anime compared to the number of episodes the Anime lasts.

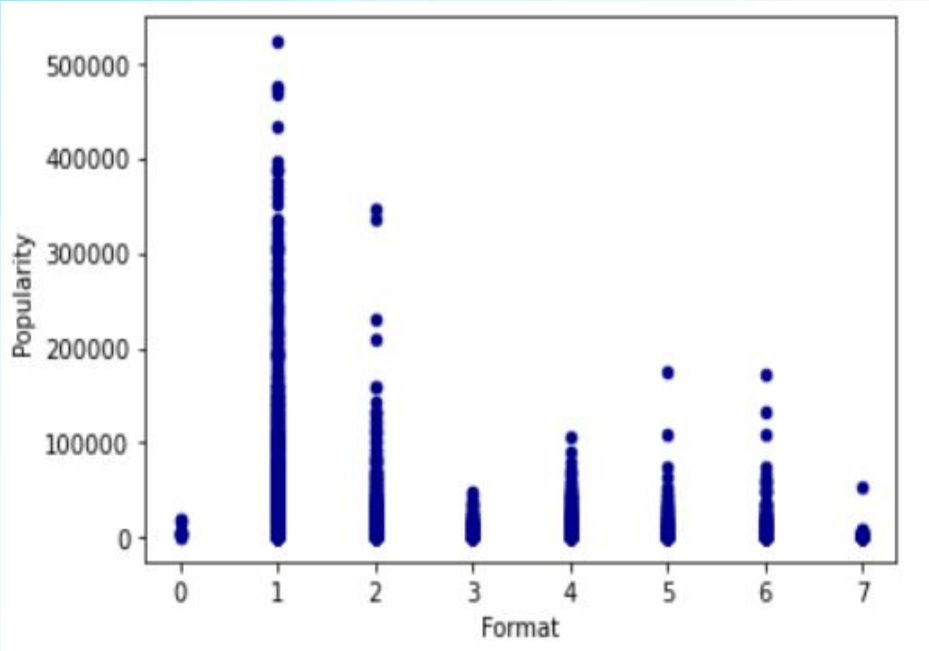
Rem, Re:Zero

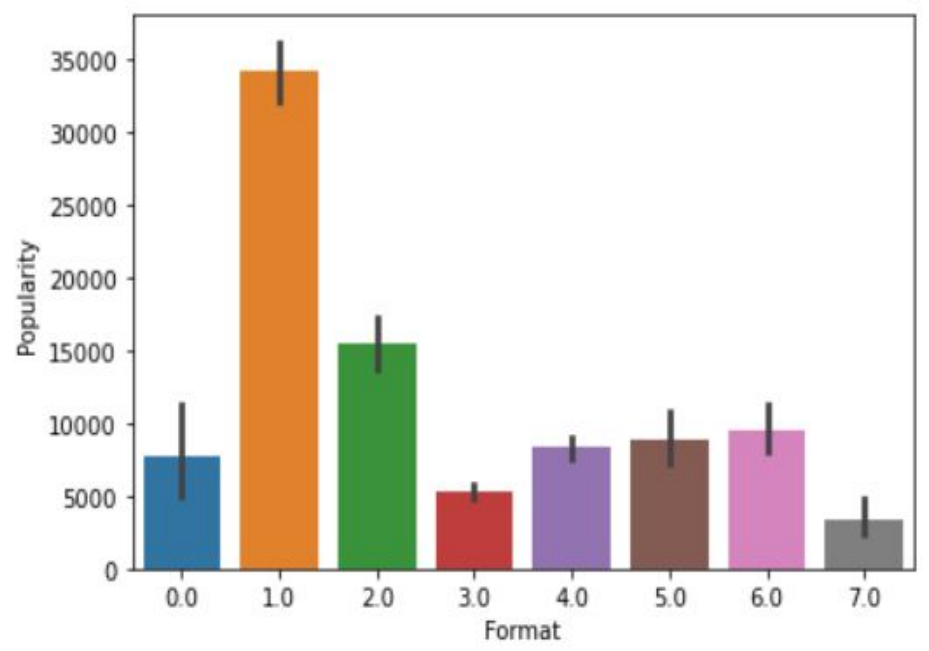
Then, we zoomed into the part that seems to have outliers.



Next, we created the following box plot, describing the Zoomed in part of the line plot, whiskers size is 1.5*IQR.

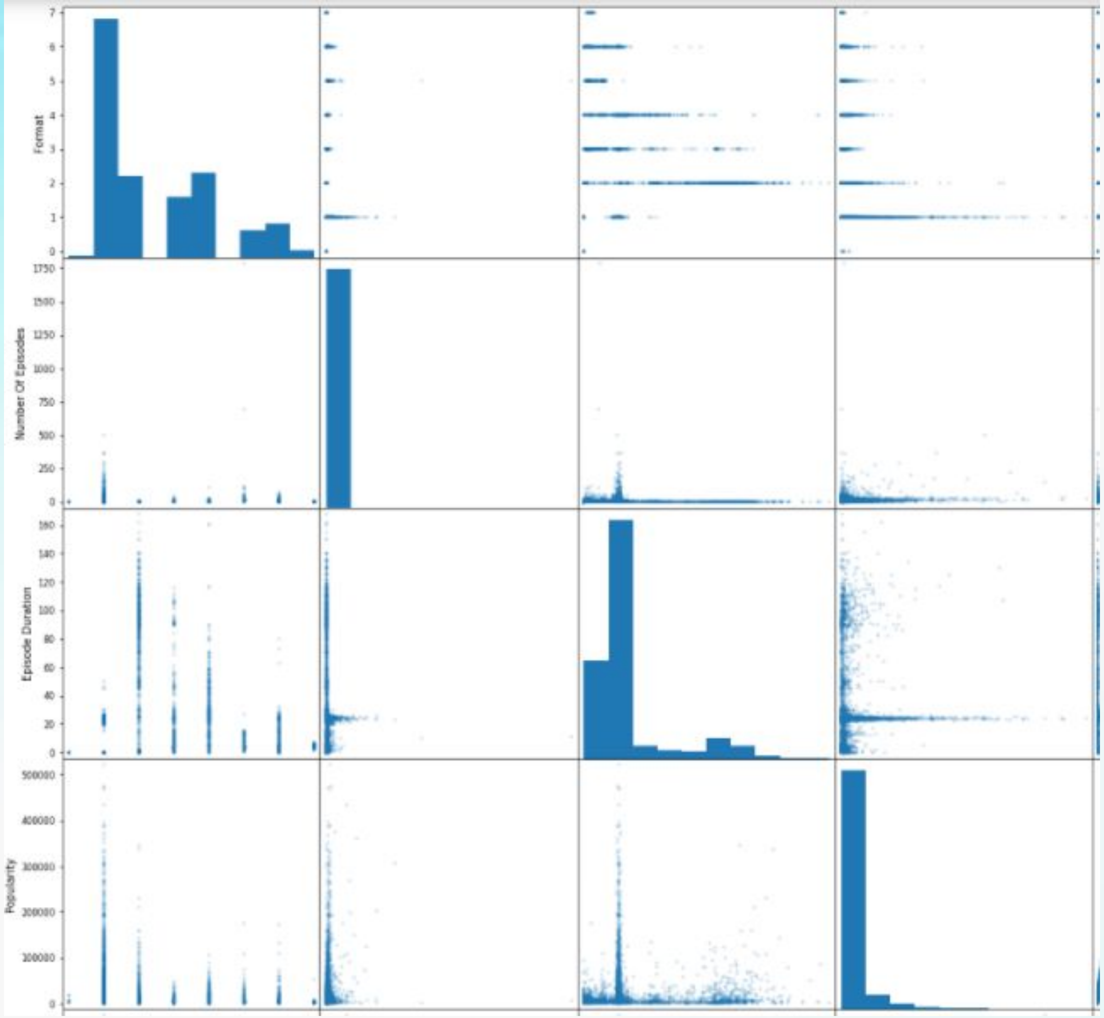
It seems there are about 10-15 outliers, so we checked manually to see if these values are possible, and it seems they are. Our data varies by different Formats, Popularities and Creators, and so some values may seem on a different scale than others, but there is no mistake here, so we decided to let these outliers remain.





These scatter plot and candle plot both describe the exact same thing: Popularity vs. Format.

A quick reminder: 0 - NaN, 1 - TV , 2 - Movie , 3 - Special , 4 - OVA , 5 - TV Short, 6 - ONA , 7 - Music.
The popularity describes the number of users who had voted - Complete/Planning/Dropped etc…



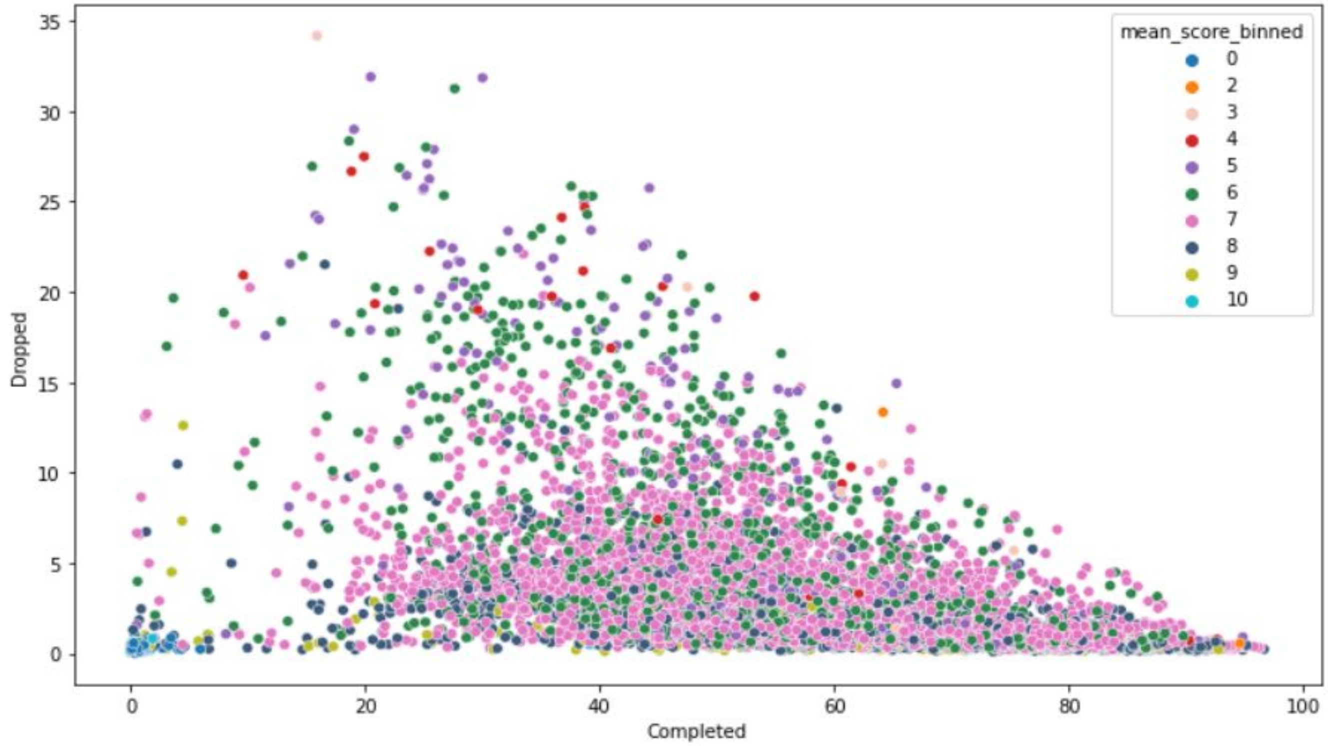
We have also created a scatter matrix of all our data. Of course this is just a very small part of it, as it is too big to showcase here.

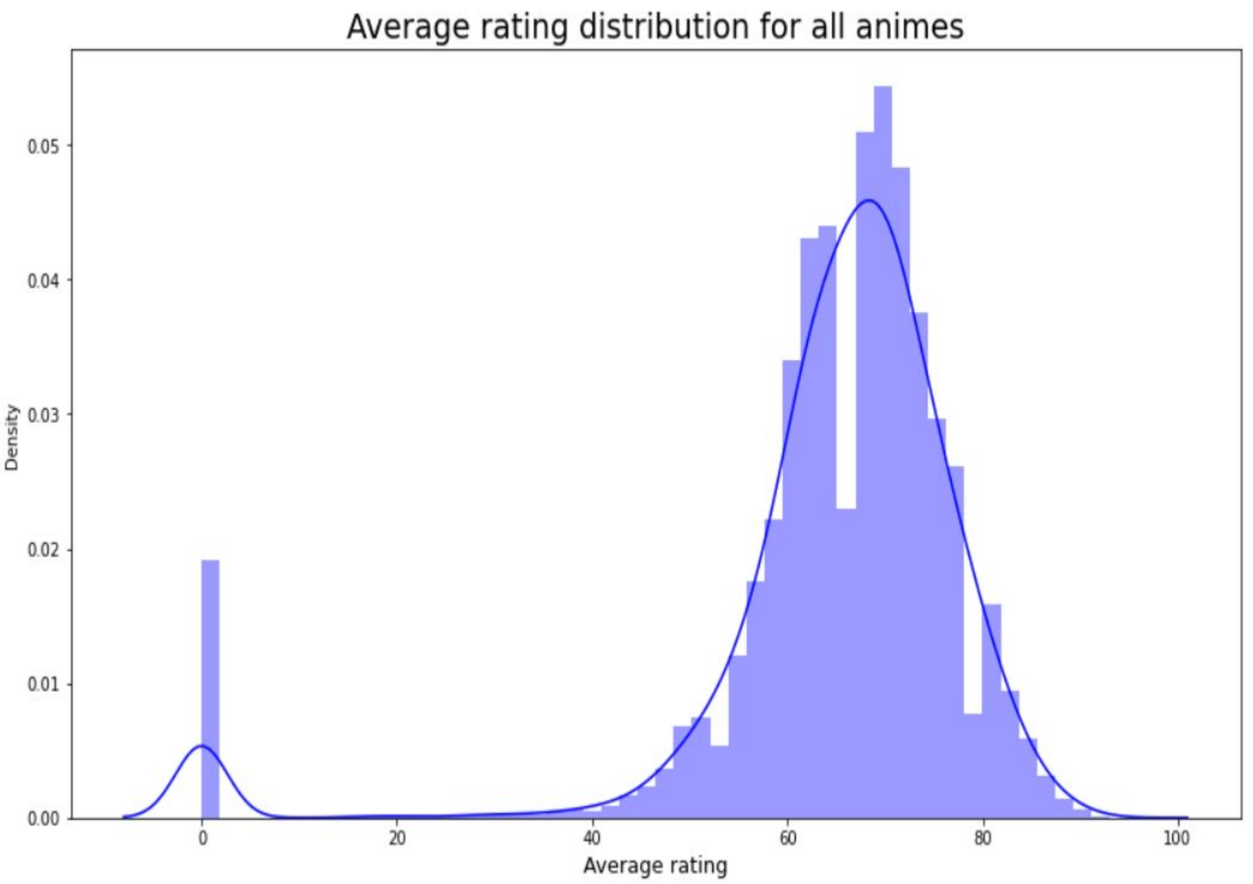
Saitama, One Punch Man

Completed	Current	Planning	Paused	Dropped
83.672238	5.931606	7.054206	1.759818	1.582132
77.784587	5.677630	9.898788	3.539674	3.099321
82.391129	6.184964	8.296654	1.586736	1.540516
77.309314	8.787429	10.386152	2.108940	1.408165
59.640682	15.065035	16.610338	6.639525	2.044420

For a few of the following visualizations, we had to transform these 5 columns into percentage.

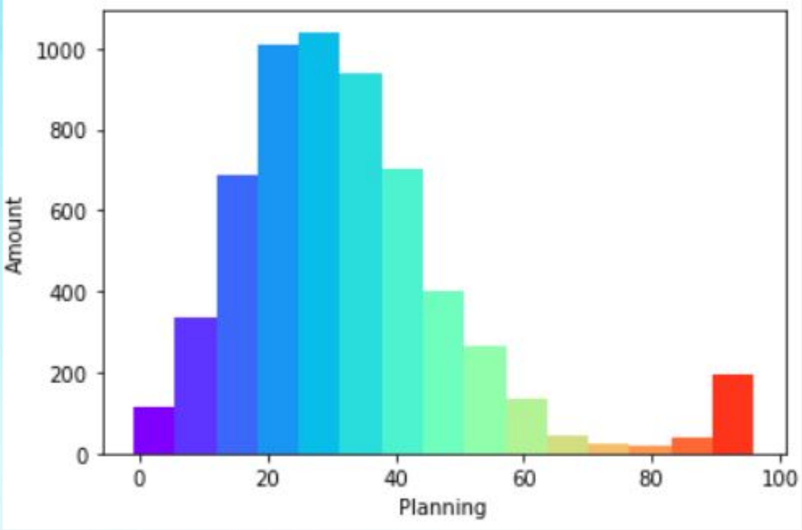
This scatterplot shows the feature's viewer drop rate vs. viewers complete rate. The colors mention the mean score range of the features. We should mention that no Anime has a mean score in category 1 which is why it's missing here.

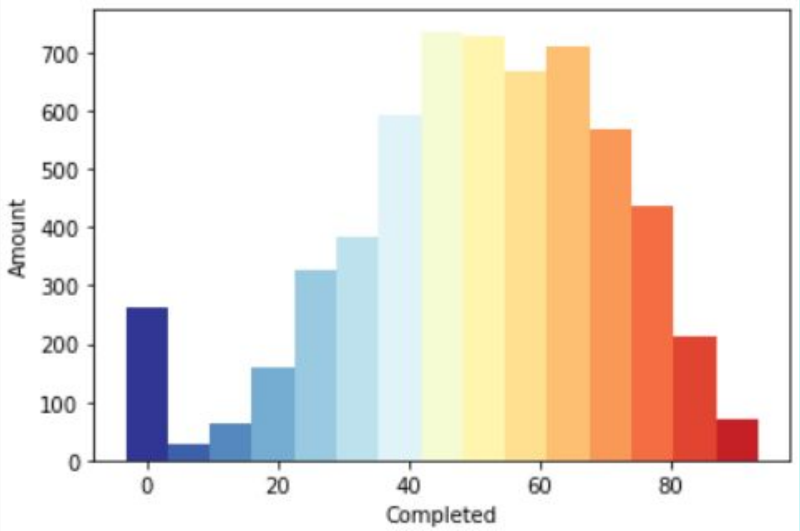




This distplot showcases the distribution of Anime in every possible mean score(Average Ratings Score 1-100).
0 is NaN.

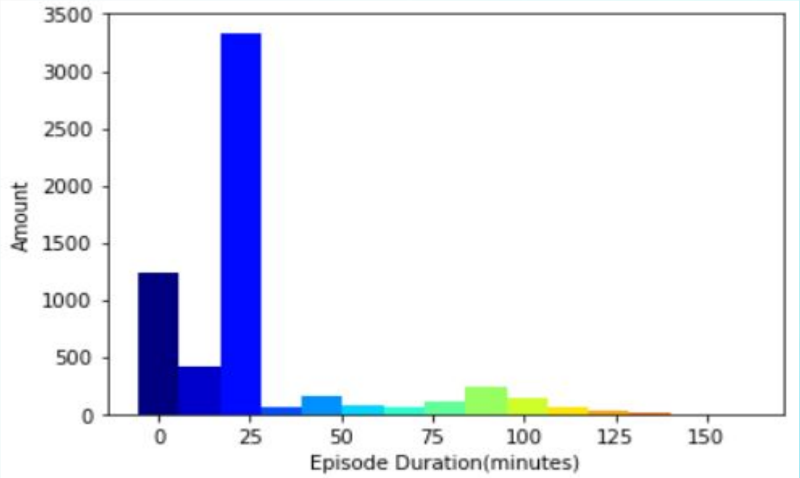
Son Goku & Vegeta, Dragon Ball Super





These histograms show the Anime distribution by the number of users Planning to watch them, Completed them, and by length of the episodes measure by minutes.

We'll remind that 'Completed' and 'Planning' were transformed into percentage out of all the users who had shown interest in some way.



General Franky, One Piece

Step 4 - Machine Learning

Our research question was: **Is it possible to predict Anime's average rating score?**

To check that, we used our mean score column as labels in a few supervised machine learning methods.

Our initial mean score column consisted of continuous numeric values, and we used it for a Linear Regression algorithm, which resulted in a very low accuracy of 0.26 success rate.

Earlier in this project, we created a Binned mean score column, which split the scores into 11 categories (0 = NaN). we decided to use this categorical column as our labels for 2 more algorithms: KNN and Decision Tree.

Our KNN resulted in a 0.54 success rate, which is an improvement, but still not enough.
The decision tree resulted in a 0.70 success rate, which to us seemed like the best result we could achieve with our current data.

The process was similar for all of our learning algorithms:

1. We added our genres secondary data frame to the first one, so we will have more detailed and accurate algorithms.

2. We saved our label column into a ‘y’ variable, and then dropped all of the String columns and both mean score columns (original and binned).

3. We used ‘train test split’ to divide our data to train (80%) and test (20%) sets, while also adding a shuffle element to the function.

4. Finally we used Fit and Predict functions in order to get our Accuracy result.

For our KNN we also used ‘GridSearchCV’ in order to find the best value for K, and the answer we got was that the best value is 29, which makes sense because it is an uneven number, which is smaller than the square root of the number of the features in our test set.

Nico Robin, One Piece

Tatsumaki, One Punch Man