Project Report Stage 2

490398331, 490407356, 480048691

Group: 79

Lab: Friday 01pm Link Building 122

University of Sydney | DATA1902 | November 2019

Table of Contents

Part One: Brief overview

Domain Knowledge

The digital age has been marked by the emergence of smartphones. Its commercial opportunities reflect its immense societal prevalence – Apple was valued at 1 trillion USD with 285 billion USD in annual sales during 2018 (P. Mourdoukoutas, 2018). Therefore, it follows that mobile apps are significant contributors to the current age. Mobile apps are a platform for entertainment, productivity and communication. As of June 2019, there are 2.7 million Google Play Store and 2.2 million Apple App Store apps (Dogtiev, 2019).

With the expansion of the mobile app industry, consumer expectations have heightened (A. Sefferman, 2019). Indeed, the success of an enterprise lies in its average rating. The average rating of an app will determine its discoverability. According to ReviewTrackers (2018), 80% of mobile app users do not trust businesses with an average rating lower than 4.0. Furthermore, app developers and businesses can benefit greatly from receiving feedback from customers and thus improving the application. Therefore, in this project, we wish to learn about the different variables that could affect an app's average rating.

Data Source

The data was sourced from two datasets from Kaggle, Apple App Store (Copyright (c) 2018 Ramanathan Perumal) and Google Play Store. The Apple App Store dataset was scraped from the iTunes API and shared on Kaggle on June 10th, 2018. Similarly, the Google Play Store dataset was scraped from the Google Play Store on September 5th, 2018, it was last updated on February 4th, 2019.

Strengths and Weaknesses

The collection of mobile app data contains no notable biases because its process is automated. Moreover, its variables are not subject to confounding or bias because they are measurements. Therefore, there are no notable weaknesses in the collection method of the data. However, the experience of mobile app users are measured by their ratings. While both the Apple App Store and Google Play Store ratings are from 1 to 5, the fact that they are on different platforms could weaken their comparison. Nevertheless, it is to a small extent to which this weakens our analysis of the dataset.

Data Rights

Kaggle is a public platform that allows users to upload datasets as either public domain or creative commons (requires attribution). The Apple App Store dataset was uploaded under a general public license (GPL-2), meaning it can be used freely by any user but requires acknowledgement of copyright upon modification or transfer. The Google Play Store dataset did not however contain any licensing information, however, as it is uploaded to be publicly available on Kaggle it can be inferred to be within the public domain.

Summary of Analysis

Quick look of the dataset post-cleaning

In [1]:
```
import pandas as pd

full_data = pd.read_csv('cleaned3-5.csv')
full_data = full_data.loc[:, ~full_data.columns.str.contains('^Unnamed')]
full_data.head(n=3)
```

Out[1]:

	Store	App	Rating	Reviews	Price	Size	Content Rating	Genre	Current Version
0	App Store	PAC-MAN Premium	4.0	21292	3.99	100788224.0	Everyone	Games	6.3.5
1	App Store	Evernote - stay organized	4.0	161065	0.00	158578688.0	Everyone	Productivity	8.2.2
2	App Store	WeatherBug - Local Weather, Radar, Maps, Alerts	3.5	188583	0.00	100524032.0	Everyone	Weather	5.0.0

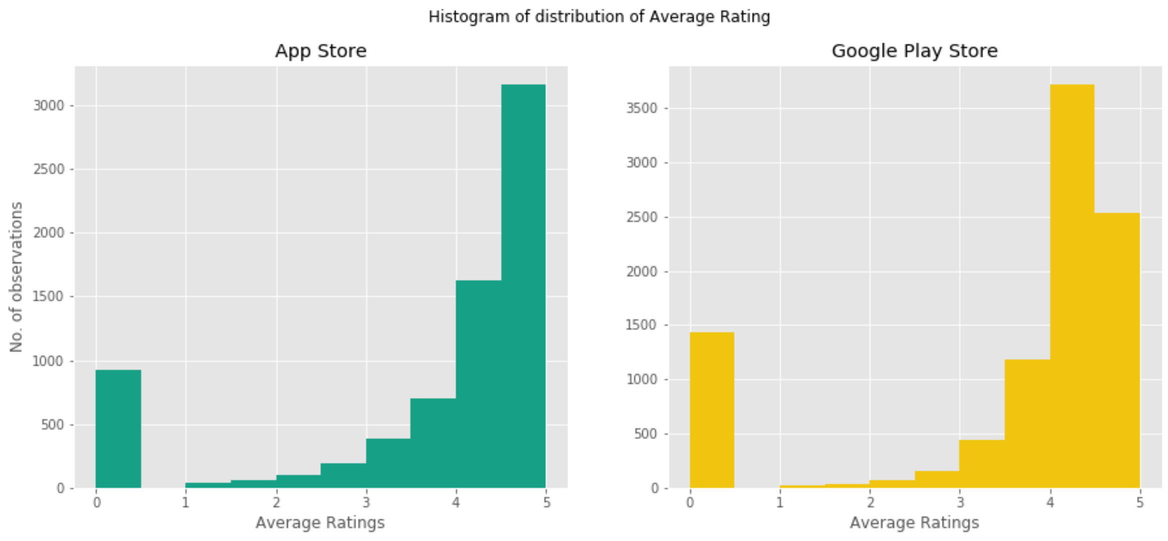
In [2]:
```
full_data.tail(n=3)
```

Out[2]:

	Store	App	Rating	Reviews	Price	Size	Content Rating	Genre	Current Version
16774	Google Play Store	Fingerprint Lock Screen Prank	4.1	10786	0.00	4300000.0	Everyone	Utilities	5.0
16775	Google Play Store	FP NFC Rewrite	0.0	17	0.00	67000.0	Everyone	Utilities	1.1
16776	Google Play Store	Fast Tract Diet	4.4	35	7.99	2400000.0	Everyone	Health and fitness	1.9.3

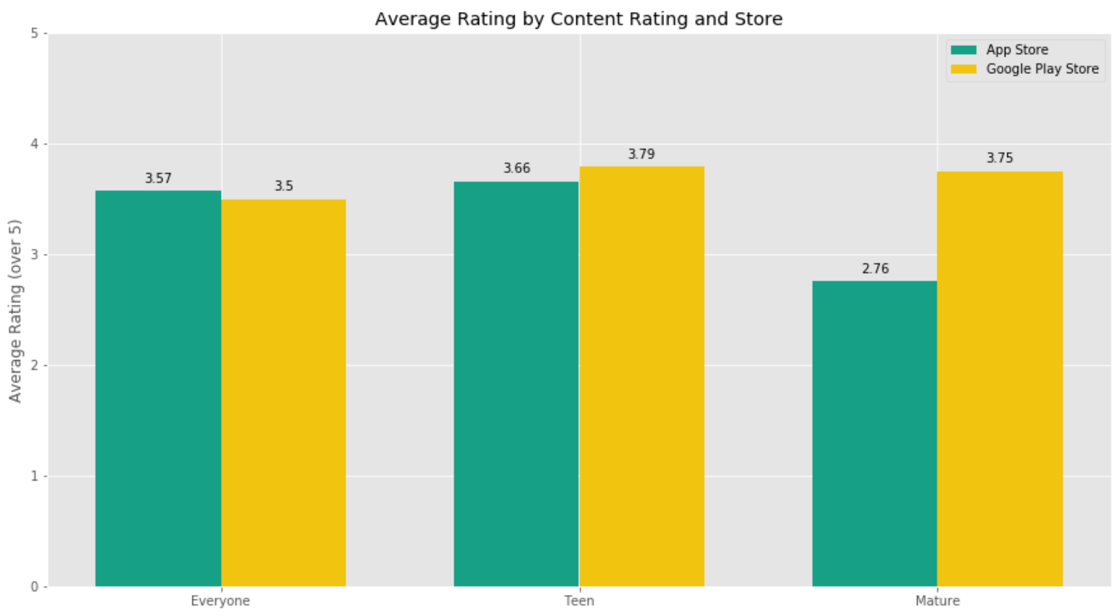
Distribution of apps among app stores

It can be seen that the majority of apps in both app stores both receive positive feedbacks from users. In the Apple App Store, the majority of apps are rated from 4.5 to 5 stars while the range is 4.0 to 4.5 for the Google Play Store. This information, however, does not lead to any definite conclusion since the size of the two application stores are different and many apps were remained unrated at the time of being scraped (whose default value was set to 0). Coverage bias is also at play since those who are willing to give a rating might have stronger opinions than those are not.



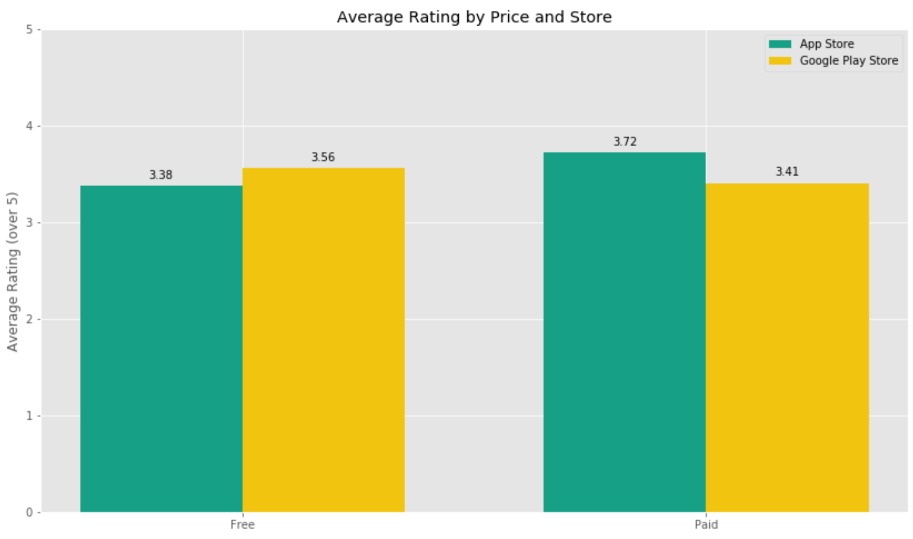
Effect of content rating on app rating

It appears that mobile apps with mature content, accumulatively, received the lowest mean average rating of 2.76. However, this trend only appeared in the Apple App Store. On the other hand, mobile apps with content suitable for everyone received the lowest mean average rating in Google Play Store.



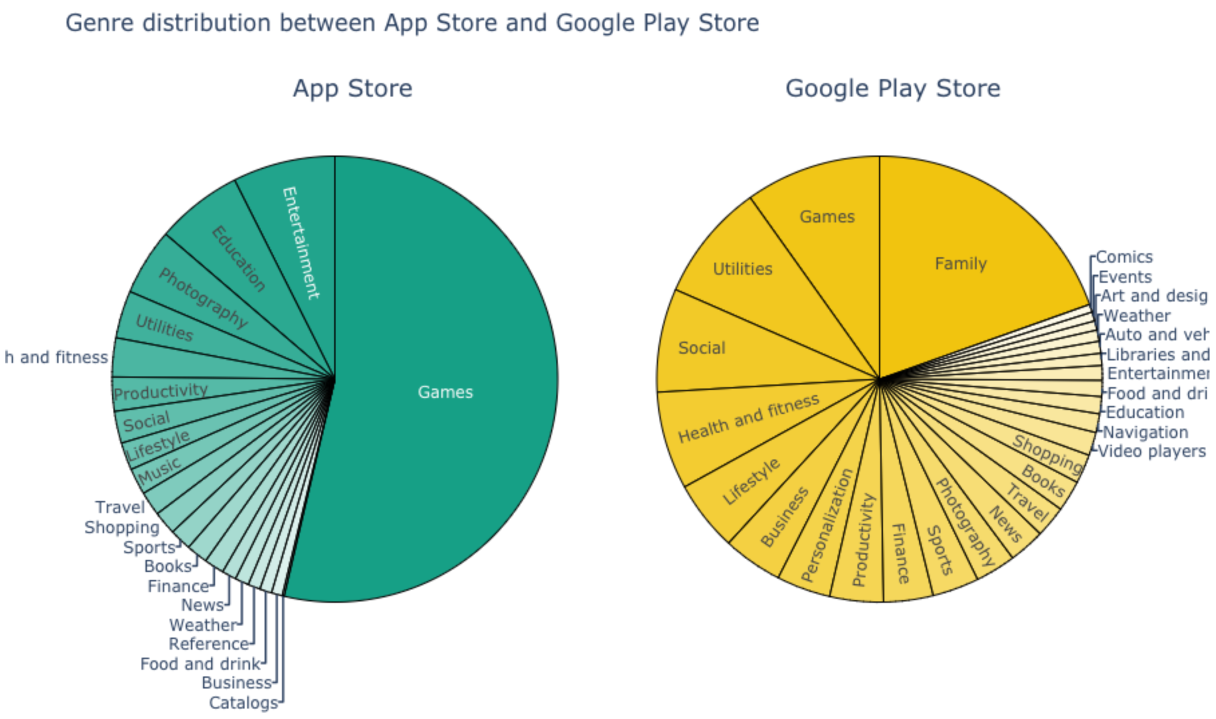
Effect of Price on App Rating

We observed here an interesting disparity between the mean average ratings of free and paid app in both stores. While paid apps, accumulatively, received a higher mean average rating in App Store (3.72 to 3.38), it is reversed in Google Play Store (3.56 to 3.41).

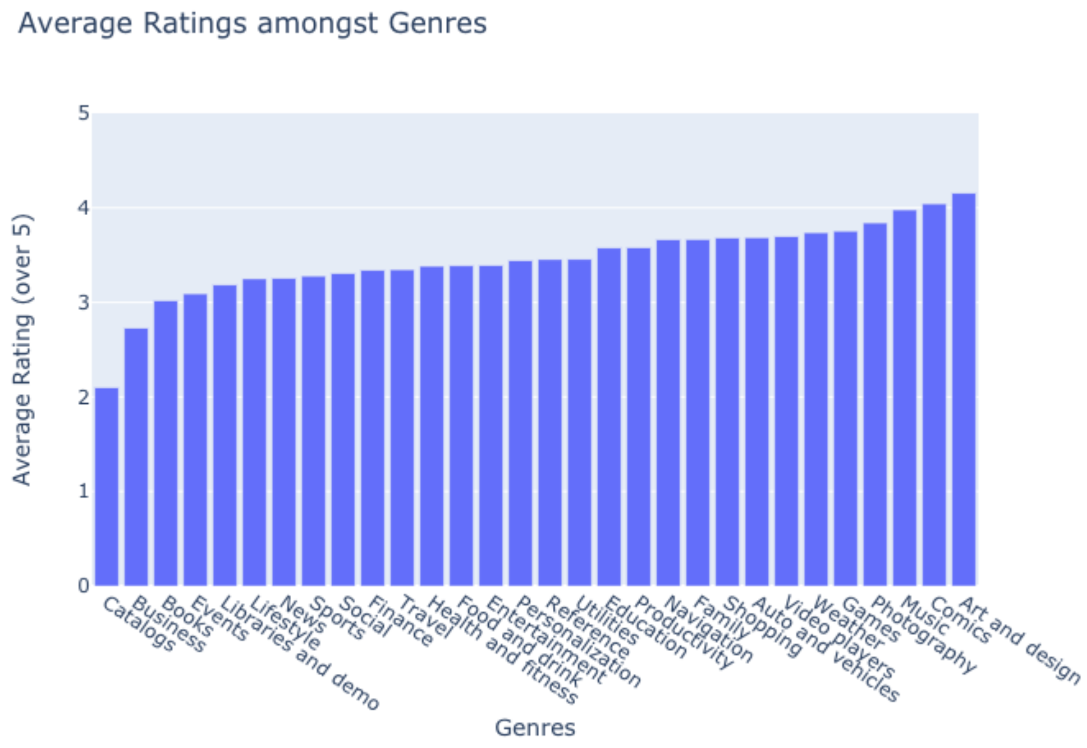


This can be explained in two different point of views. When a user is willing to pay for a mobile app, it usually indicates that the app and the content it offers is that of the user's interest. However, if the app does not behave accordingly to its advertisement, or to the user's expectation, it is now subjected to a lower rating. Therefore, further analysis needs to be carried out before any conclusion is made.

Analysis of Genre Distribution and Rating



According to our datasets, the most prominent genre in the Apple App Store is Games with a proportion of 53.7% and Family with 19.6% in Google Play Store.



It can be seen that the three categories with the highest average ratings are “Art and Design” (4.15), “Comics” (4.04), and “Music” (3.98). It must be mentioned, however, that since “Art and Design” and “Comics” did not receive many user ratings as of the time the dataset being scraped, their average ratings are much more sensitive to any new rating recorded compared to “Games”.

Another aspect explored was the effect of both content rating and genre on the average rating of an app. The combination of content rating and genre which produced the highest average rating was Teen/Art and Design with an average rating of 4.5 (Figure 1), followed by Mature/Weather, Teen/Weather and Everyone/Comics all with average ratings of 4.2.

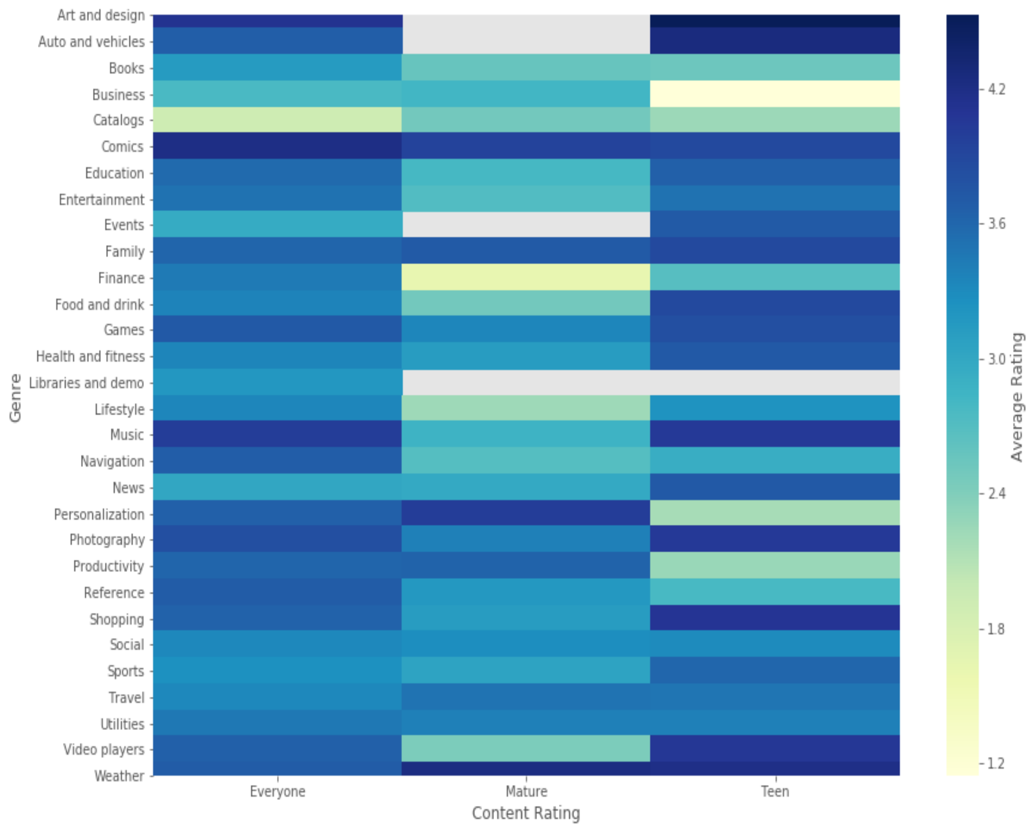
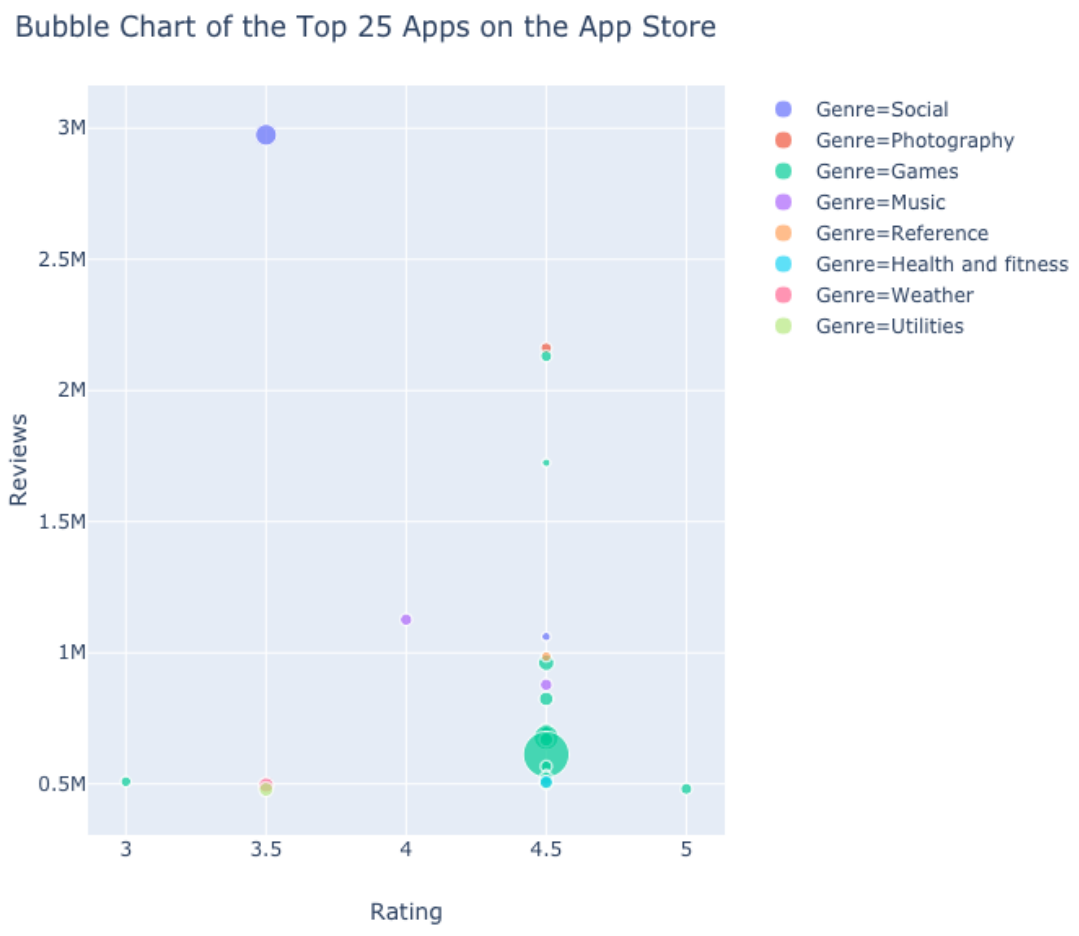


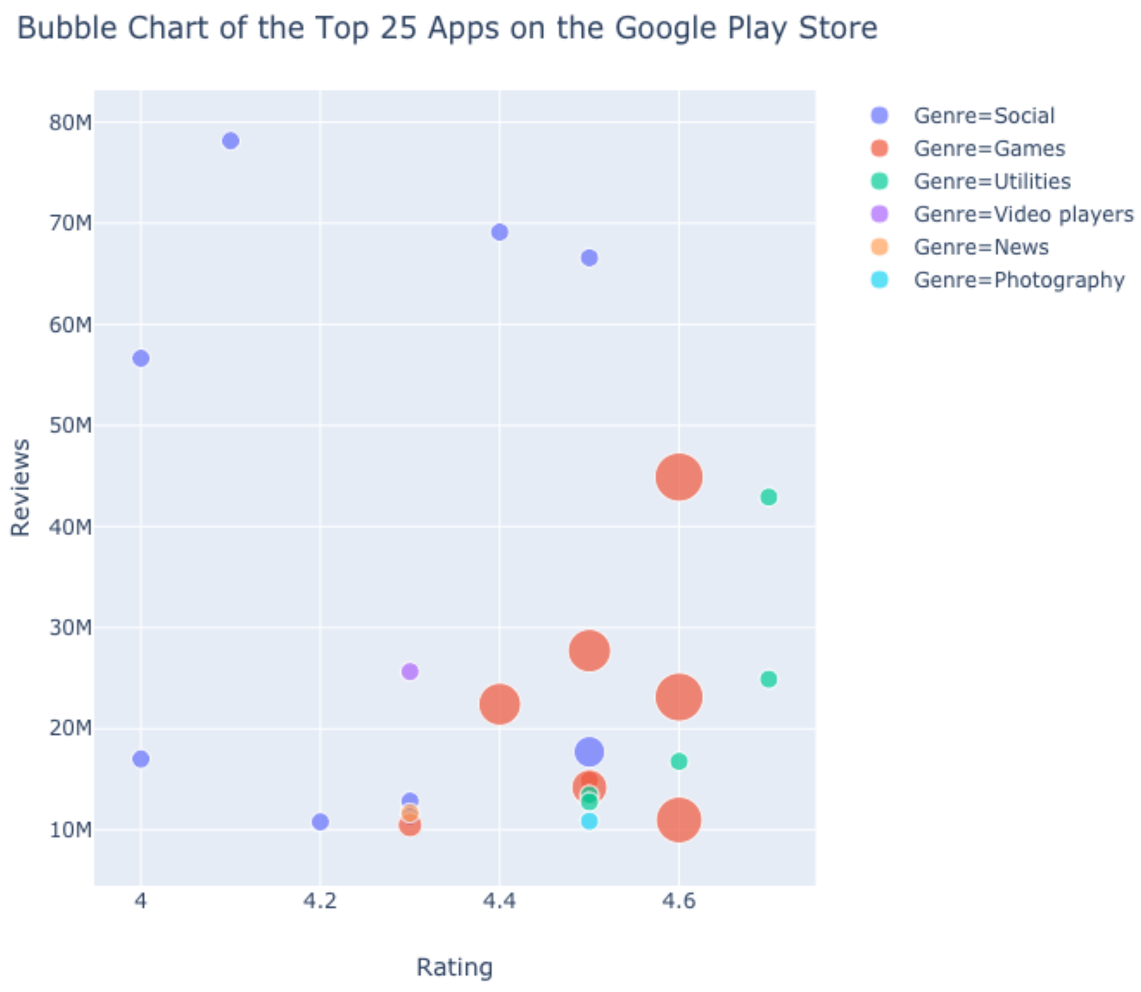
Figure 1: A heatmap of the effect of content rating and genre on the average rating of an app. The x-axis are the content ratings and the y-axis are the genres. The colour of each box represents a value between 0 and 5, with the colours progressively darkening as an average rating of 5 is approached. A box with no colour represents no apps of that combination existing in the dataset.

It also appears that the categories such as ‘Shopping’, ‘Photography’, and ‘Food and Drink’ are more popular amongst Teen, while genres that are popular amongst Everyone are ‘Comics’, ‘Music’ and ‘Games’ and for Mature they are ‘Personalisation’, ‘Weather’, and ‘Family’. On the contrary, categories that are least popular amongst Teens are ‘Productivity’, ‘Business’, and ‘Personalisation’, while they are ‘Catalogs’ for Everyone and ‘Finance’ for Mature.

Analysis of the top 25 apps on both stores

The variables of the top 25 apps in each store can help provide insight into what makes an app popular. A bubble chart was generated for each store, where the x-axis is the app rating, y-axis is the number of reviews, size of the bubble is the physical size of the app in bytes and colour of the bubble is the genre





It can be seen that a majority of the top 25 apps in both stores are fairly uniform in size, while the genres that are appear to be popular on both stores are social, games and utilities. The app with the most reviews on both stores was Facebook, which also had a relatively low rating when compared to the others in the top 50.

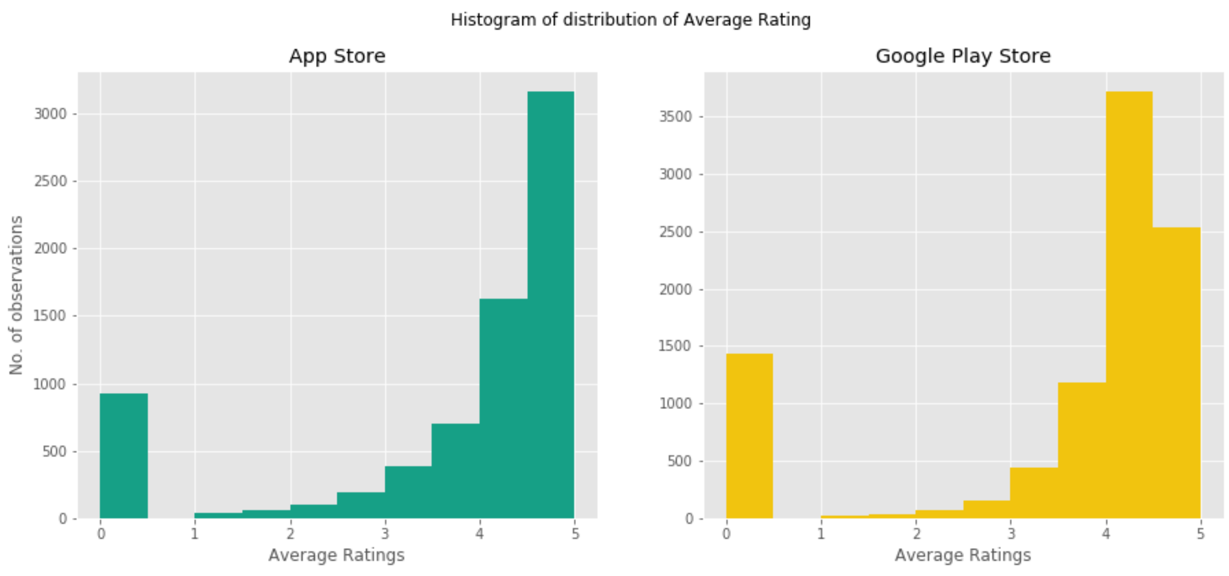
Part Two: Analysis

We will now discuss about the process of how we determined the general relationship between average ratings and each respective variable.

Distribution of Average Ratings amongst the two dataset

The app store dataset was analysed using **pandas** library in python and graphed using the **matplotlib** library. In order to understand the distribution of average ratings among each dataset, we used the matplotlib library to graph two histograms accordingly for each store. The 'ggplot' style was implemented for an effective visualisation. To obtain two subplots, the method **plt.subplots(1, 2)** was used.

```python
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")

full_data = pd.read_csv("cleaned3-5.csv”)
apple_data = full_data.loc[full_data['Store'] == "App Store"]
google_data = full_data.loc[full_data['Store'] == "Google Play Store"]

fig, ax = plt.subplots(1,2, figsize=(15,6))
ax[0].hist(apple_data["Rating"], color = '#16A086')
ax[0].set_xlabel("Average Ratings")
ax[0].set_ylabel("No. of observations")
ax[0].set_title("App Store")

ax[1].hist(google_data["Rating"], color = '#F1C40F')
ax[1].set_title("Google Play Store")
ax[1].set_xlabel("Average Ratings")
fig.suptitle("Histogram of distribution of Average Rating")
plt.show()
```



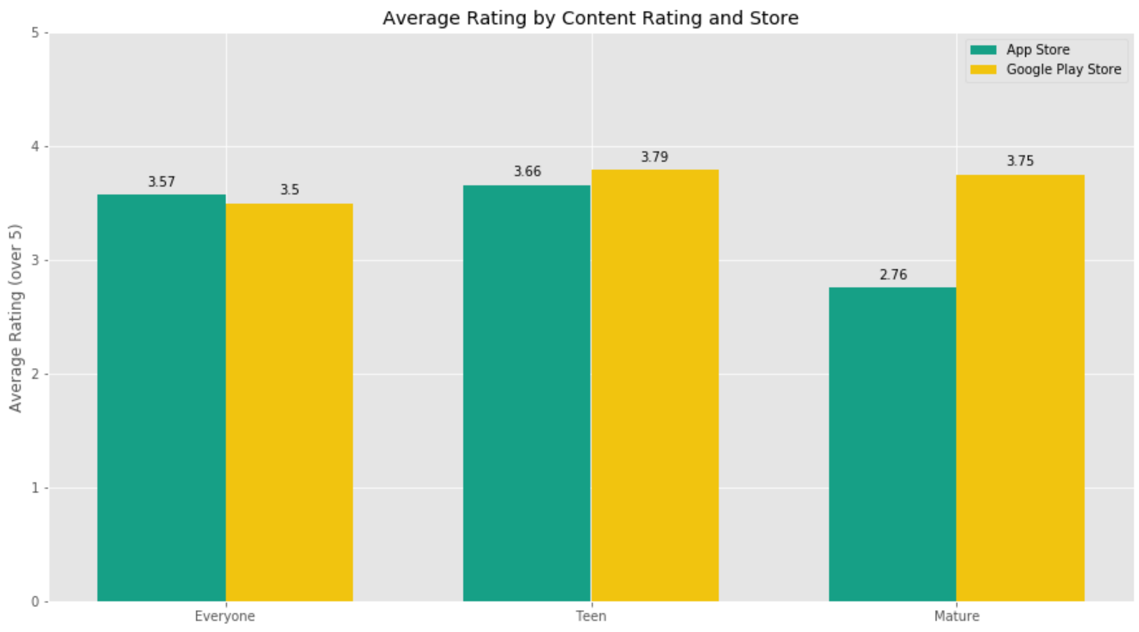
Distribution of Average Ratings amongst Content Ratings

A grouped bar plot was used to compare the average rating for each content rating. Pandas was used to read the csv with column 7 (content rating) as the index and an axis was again created using **plt.subplots**(). The “ggplot” style was once again used.

This method was chosen for the analysis of content ratings as it produced a clear and concise visual summary of the mean of app ratings within each content rating and quick comparison between two app stores.

```python
#prepare data
full_data = pd.read_csv("cleaned3-5.csv", index_col = 8)
apple_data = full_data.loc[full_data['Store'] == "App Store"]
google_data = full_data.loc[full_data['Store'] == "Google Play Store"]

apple_content = apple_data.groupby('Content Rating')
apple_content_dict = apple_content['Rating'].mean().to_dict()
apple_content_values = []

for key in apple_content_dict:
    apple_content_values.append(round(apple_content_dict[key], 2))

apple_vals = [apple_content_values[0], apple_content_values[2], apple_content_values[1]]

google_content = google_data.groupby('Content Rating')
google_content_dict = google_content['Rating'].mean().to_dict()
google_content_values = []

for key in google_content_dict:
    google_content_values.append(round(google_content_dict[key], 2))

google_vals = [google_content_values[0], google_content_values[2], google_content_values[1]]  #changing the order because in the dict, mature comes before teen

#plot
import numpy as np
plt.style.use("ggplot")
fig, ax = plt.subplots(figsize = (15, 8))

N = 3
ind = np.arange(N)
width = 0.35
b1 = plt.bar(ind, apple_vals, width, label = "App Store", color = '#16A086')
b2 = plt.bar(ind + width, google_vals, width, label = "Google Play Store", color = '#F1C40F')

def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height,2)),
                    xy=(rect.get_x() + rect.get_width() / 2, 0.05+height),
                    ha='center', va='bottom', rotation=0)

```

```python
autolabel(b1)
autolabel(b2)

ax.set_ylabel("Average Rating (over 5)")
ax.set_title("Average Rating by Price and Store")
ax.set_ylim([0,5])
plt.xticks(ind + width / 2, ('Everyone', 'Teen', 'Mature'))
ax.legend(loc='best')
plt.show()
```

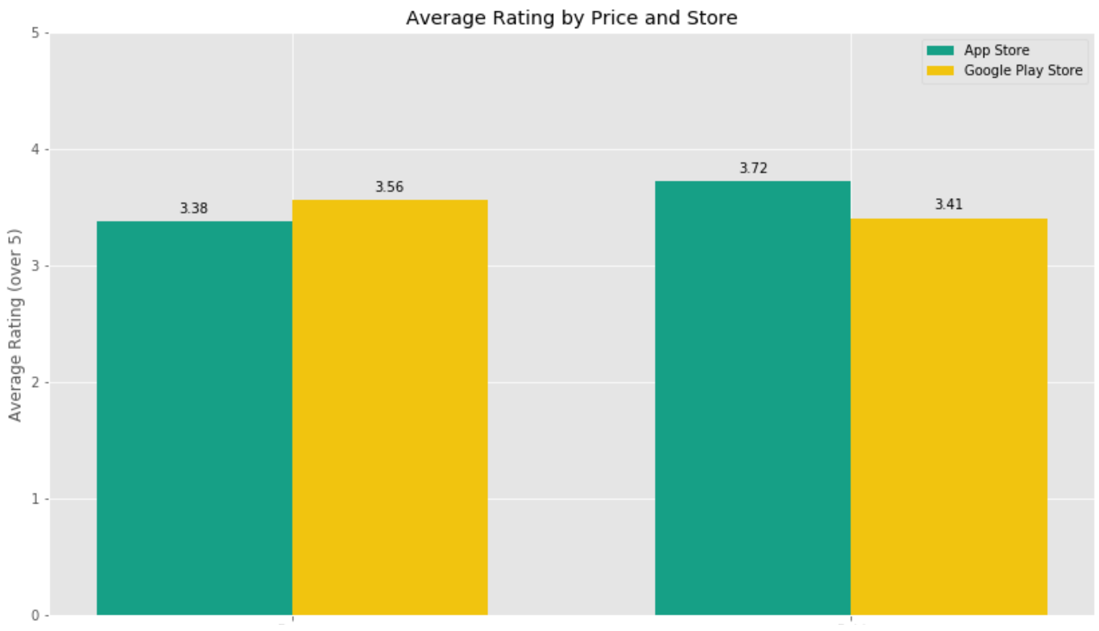


It appears that mobile apps with mature content, accumulatively, received the lowest mean average rating of 2.76. However, this trend only appeared in the Apple App Store. On the other hand, mobile apps with content suitable for everyone received the lowest mean average rating in Google Play Store.

Distribution of Average Ratings amongst Price

The approach above is also used in determining the relationship between an app's average ratings and its price.

```python
df = pd.read_csv('cleaned3-5.csv')

google_app_prices = {"Free": 0, "Paid": 0}
google_app_rating_cumulative = {"Free": 0, "Paid": 0}

apple_app_prices = {"Free" : 0, "Paid": 0}
apple_app_rating_cumulative = {"Free": 0, "Paid": 0}

for index, row in df.iterrows():
    if row['Store'] == "App Store":
        if float(row['Price']) == 0:
            apple_app_prices["Free"] += 1
            apple_app_rating_cumulative["Free"] += float(row['Rating'])
        else:
            apple_app_prices["Paid"] += 1
            apple_app_rating_cumulative["Paid"] += float(row['Rating'])
    else:
        if float(row['Price']) == 0:
            google_app_prices["Free"] += 1
            google_app_rating_cumulative["Free"] += float(row['Rating'])
        else:
            google_app_prices["Paid"] += 1
            google_app_rating_cumulative["Paid"] += float(row['Rating'])

print("Average Ratings for apps in Google Play Store based on their price:")
google_mean_ls = []
for key in google_app_prices:
    mean = google_app_rating_cumulative[key]/google_app_prices[key]
    google_mean_ls.append(mean)
    print("{}: {:.2f}".format(key, mean))

print()

print("Average Ratings for apps in Apple Play Store based on their price:")
apple_mean_ls = []
for key in apple_app_prices:
    mean = apple_app_rating_cumulative[key]/apple_app_prices[key]
    apple_mean_ls.append(mean)
    print("{}: {:.2f}".format(key, mean))
```



Effect of Genre on Average Ratings

Distribution of Genres

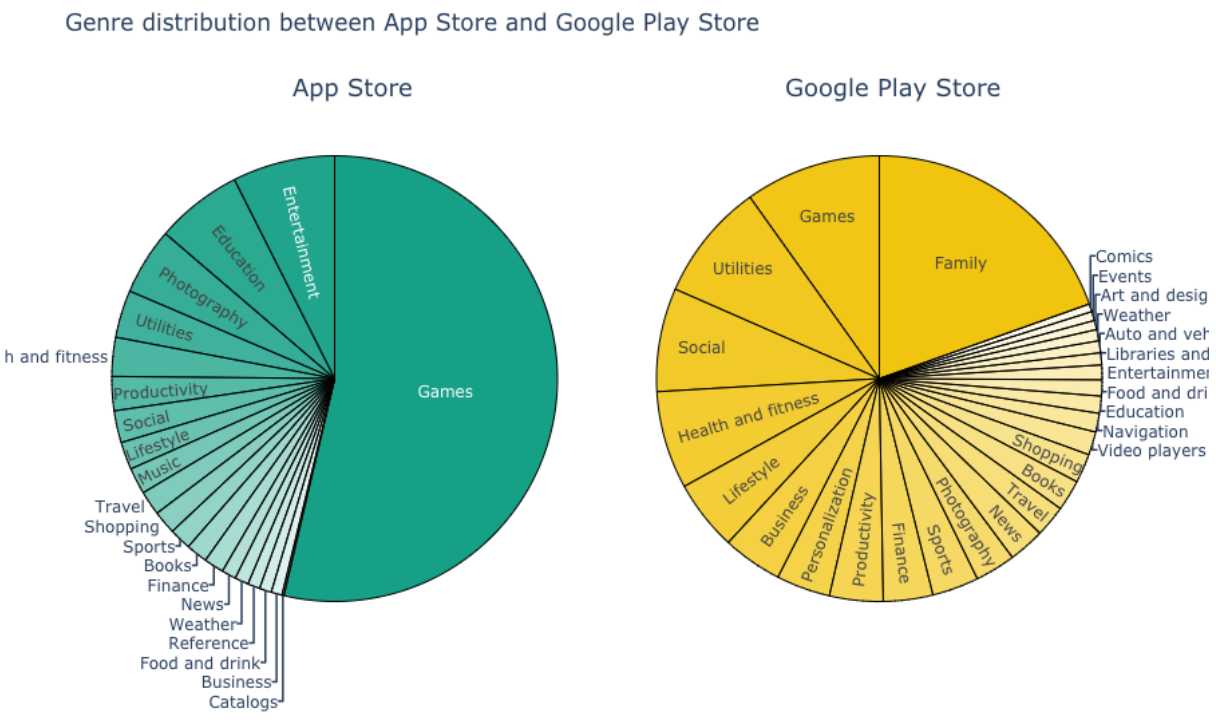
To understand the effect of an app's genre to its average rating, we first explore the distribution of app genres in both mobile application stores. We first used **matplotlib** to present this information by using barcharts. However, given the vast number of genres in both stores, this approach quickly appeared to be not helpful and hard for readers to quickly identify most prominent genres.

Pandas was used in preparing the data. We used the groupby, size, and to_dict functions to create a dictionary of each genre as key and the number of observations as its corresponding value. We then create a sorted list to sort the categories by their number of observations in a descending manner.

```python
full_data = pd.read_csv("cleaned3-5.csv")
apple_data = full_data.loc[full_data['Store'] == "App Store"]
google_data = full_data.loc[full_data['Store'] == "Google Play Store"]

apple_genre_count = apple_data.groupby('Genre').size().to_dict()
google_genre_count = google_data.groupby("Genre").size().to_dict()

apple_genre_sorted = sorted(apple_genre_count.items(), key = lambda item: item[1],
reverse = True)
google_genre_sorted = sorted(google_genre_count.items(), key = lambda item: item[1]
, reverse = True)

labels1 = []
values1 = []

for i in apple_genre_sorted:
    labels1.append(i[0])
    values1.append(i[1])

labels2 = []
values2 = []

for i in google_genre_sorted:
    labels2.append(i[0])
    values2.append(i[1])
```

We then decided to use two pie charts, as shown below. This was done by using the **plotly** graphing module. An interactive version is provided in the attached folder under the name **genre_distribution.html**, which users can learn about each genre's proportion to the entire store by hovering on each slice of the pie.

```python
import plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots

specs = [[{'type':'domain'}, {'type':'domain'}]]
fig = make_subplots(rows=1, cols=2, specs=specs)

fig.add_trace(go.Pie(labels = labels1, values = values1, text = labels1, marker_col
ors = colors1, marker_line = dict(color='#000000', width=1)), 1, 1)
fig.add_trace(go.Pie(labels = labels2, values = values2, text = labels2, marker_col
ors = colors2, marker_line = dict(color='#000000', width=1)), 1, 2)

fig.update_traces(hoverinfo='label+percent', textinfo = "text")
fig.update(layout_showlegend=False)

fig.update_layout(
    height = 600,
    width = 900,
    title_text="Genre distribution between App Store and Google Play Store",
    annotations=[dict(text='App Store', x = 0.18, y = 1.05, font_size = 18, showarrow = False),
                     dict(text='Google Play Store', x = 0.9, y= 1.05, font_size = 18, showarrow = False)])
fig.write_image("genre_distribution.png")
plotly.offline.plot(fig, filename='genre_distribution.html')
fig.show()
```



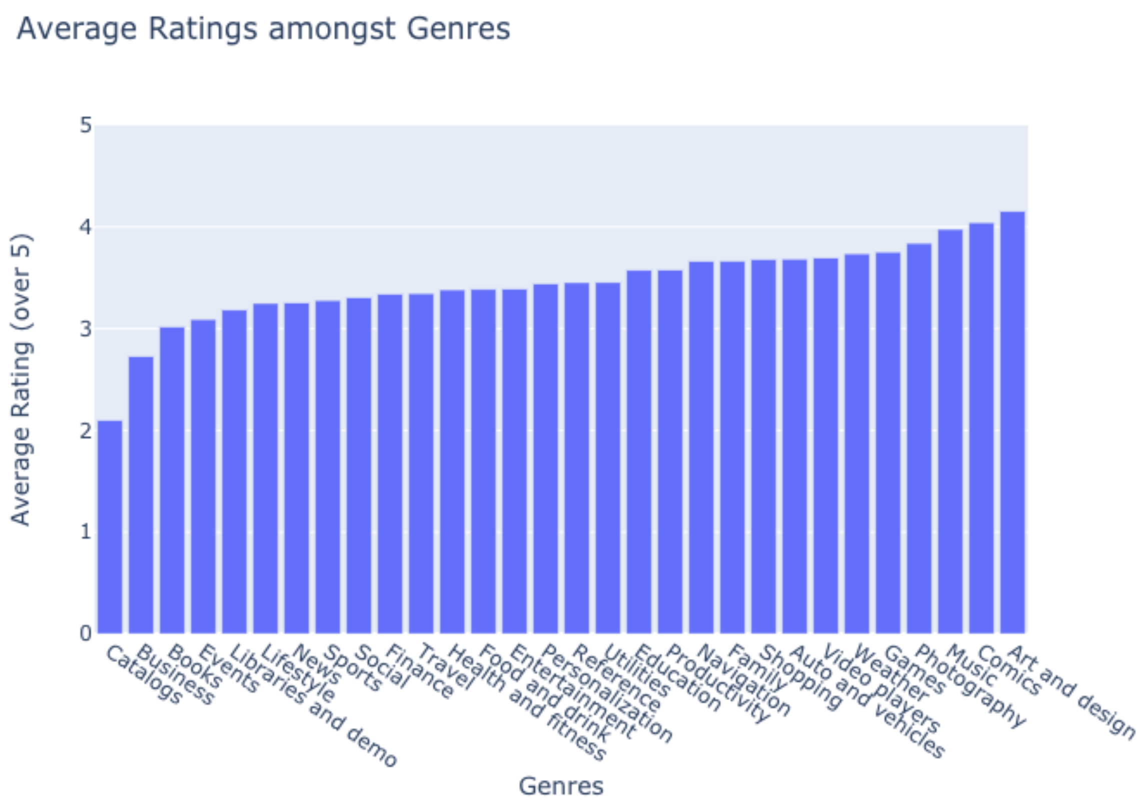
It can be seen that the most 3 popular categories in App Store are Games (53.7%), Entertainment (7.43%), and Education (6.29%), while they are Family (19.6%), Games (9.86%), and Utilities (8.56%) in Google Play Store.

Genres and Average Ratings

Another aspect of the dataset that was analysed using pandas in conjunction with plotly was **the effect of genre on rating**. Pandas was used to first group the apps by genre and then take the mean. This mean was placed into a dictionary as a value with the genre as the key. Once the axis had been created and the appropriate labels put in place, a for loop was created to go through each key in the aforementioned dictionary and produce a bar for the barplot. This resulted in a somewhat hard to interpret barchart with the comparison of genres with close average ratings being difficult.

The barchart was hence ordered by increasing average rating left to right so that information regarding the average rating for a genre could be easily deduced and compared. This was done by mapping each mean rating value in the dictionary to its key and then creating a new sorted dictionary based on mean rating. An interactive version is also provided under the name **‘genre_avg_rating.html’**, in which user can hover on each bar to see the exact average rating of each genre.

```python
import plotly.graph_objects as go

full_data = pd.read_csv("cleaned3-5.csv")

full_sorted = full_data.sort_values('Rating', ascending = False)
full_genre = full_sorted.groupby("Genre")
full_genre_ratings = full_genre['Rating'].mean().to_dict()

sorted_d = sorted(full_genre_ratings.items(), key = lambda item: item[1])

labels = []
full_values = []

for i in sorted_d:
    labels.append(i[0])
    full_values.append(i[1])

fig = go.Figure([go.Bar(x=labels, y=full_values)])
fig.update_layout(title="Average Ratings amongst Genres",
                  xaxis_title="Genres",
                  yaxis_title="Average Rating (over 5)")
fig.update_xaxes(tickangle = 35)
fig.update_yaxes(range=[0, 5])
fig.show()
```



Heatmap comparing genre and Average Rating

For the heatmap exploring combinations of genre and content rating, the pandas library was again used to sort the data. Pandas grouped the data by two qualitative variables, “Content Rating” and “Genre” and then took the mean rating for each of the combinations and placed it in a dictionary using the **mean.todict** function.

```python
from collections import defaultdict
import matplotlib.pyplot as plt
import seaborn as sns
full_data = pd.read_csv("cleaned3-5.csv")

full_data = full_data.groupby(["Content Rating","Genre"])
genre_user = full_data['Rating'].mean().to_dict()
```

However, this creates a dictionary where the key is a tuple containing both content rating and genre (e.g. (Teen, Games)) which was the incorrect format for producing a heatmap. Instead, a new dictionary had to be made where each key was the content rating and within the entry for each key was a nested dictionary containing the genre and average rating of the combination of genre plus content rating.

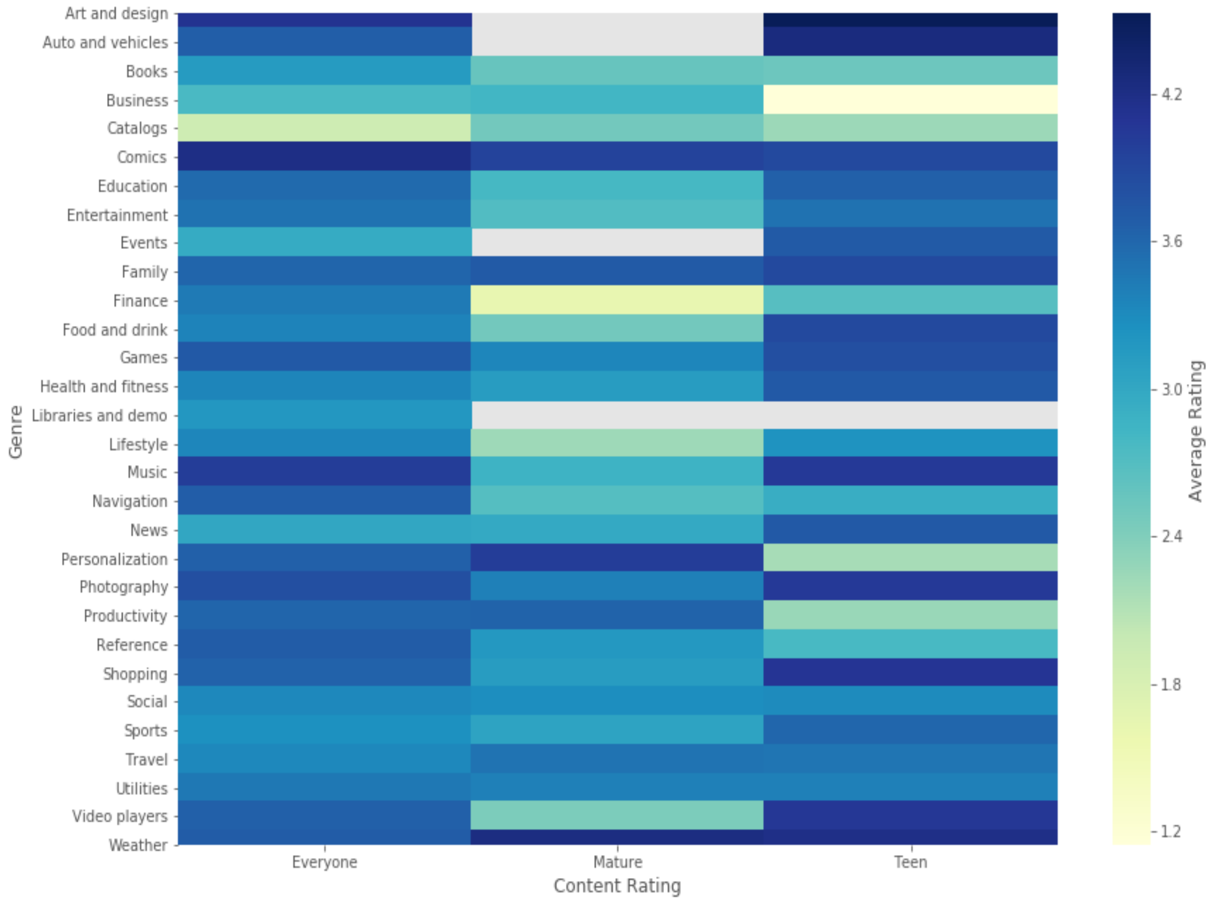
```python
full = defaultdict(dict)

for key in genre_user:
    content_rating = key[0]
    genre = key[1]
    full[content_rating][genre] = {}
    full[content_rating][genre] = genre_user[key]
```

The library collections had to be imported for this step to create a default dictionary from which the nested dictionary could be created as otherwise a KeyError would occur in the for loop which created the nested dictionary.

The nested dictionary was then converted back into a pandas dataframe using **pd.Dataframe.from_dict**() and the axis was created using matplotlib with figure size set to 10 x 10. The seaborn library was used to generate the heatmap from the dataframe. Initially the colour map was set to default, however, this made parts of the heatmap where there was no value appear to be at the top end of the colour scale. Instead, the colour map was set to **YellowGreenBlue** as this provided the greatest visual clarity. Finally, the appropriate labels were added and the figure was generated.

```python
heatmap = pd.DataFrame.from_dict(full)
fig, ax = plt.subplots(figsize=(10,10))

heatmap = pd.DataFrame.from_dict(full)
fig, ax = plt.subplots(figsize=(10,10))

```

```python
ax1 = sns.heatmap(heatmap, cmap = "YlGnBu", ax = ax,cbar_kws = {'label': 'Average R
ating'})
ax1.set(xlabel = 'Content Rating', ylabel = 'Genre')
plt.savefig('heatmap.png')
plt.show()
```



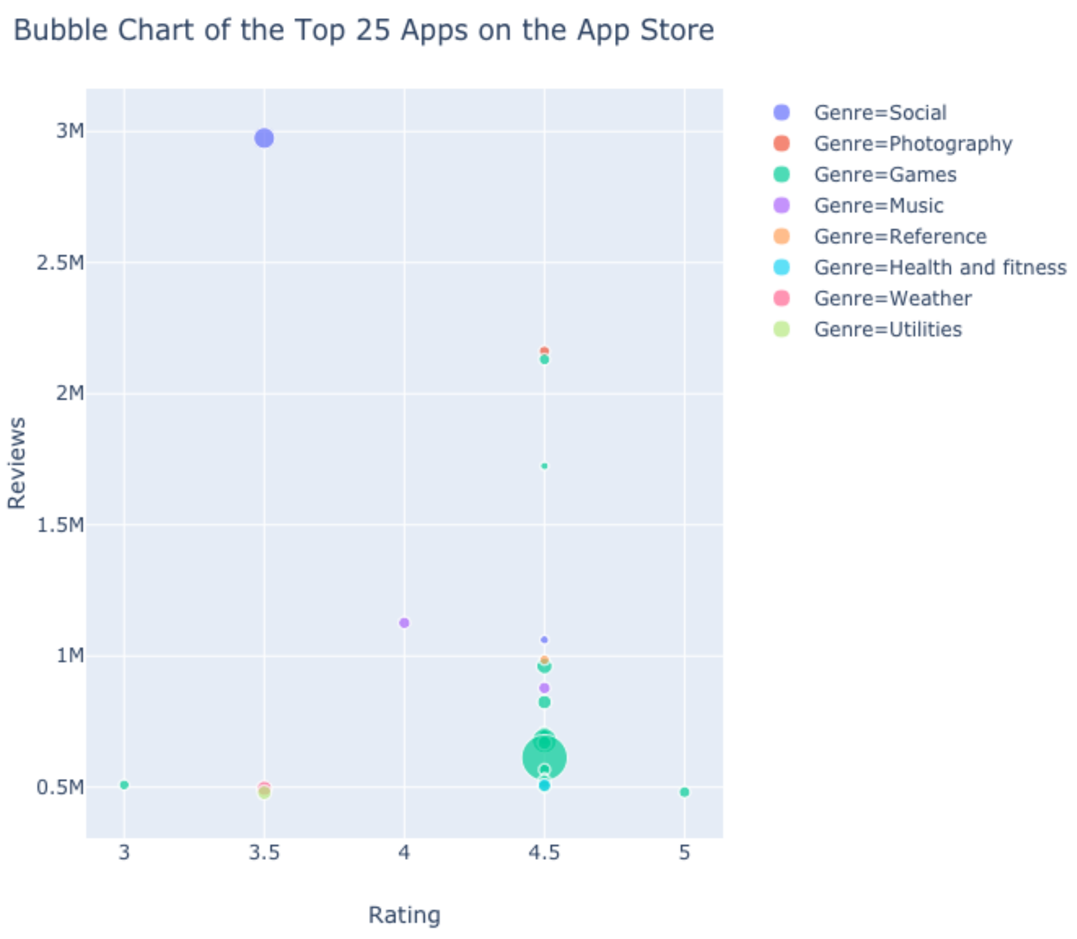
Bubble Chart Exploring the Top 25 Apps on Both Stores

The chart generated in this section used four variables: Reviews, Rating, Size and Genre. Pandas was used to split the dataset into two dataframes, one containing the App Store entries and one containing the Google Play Store entries. Each dataframe was sorted based on total reviews and the top 25 apps were taken into a new dataframe.

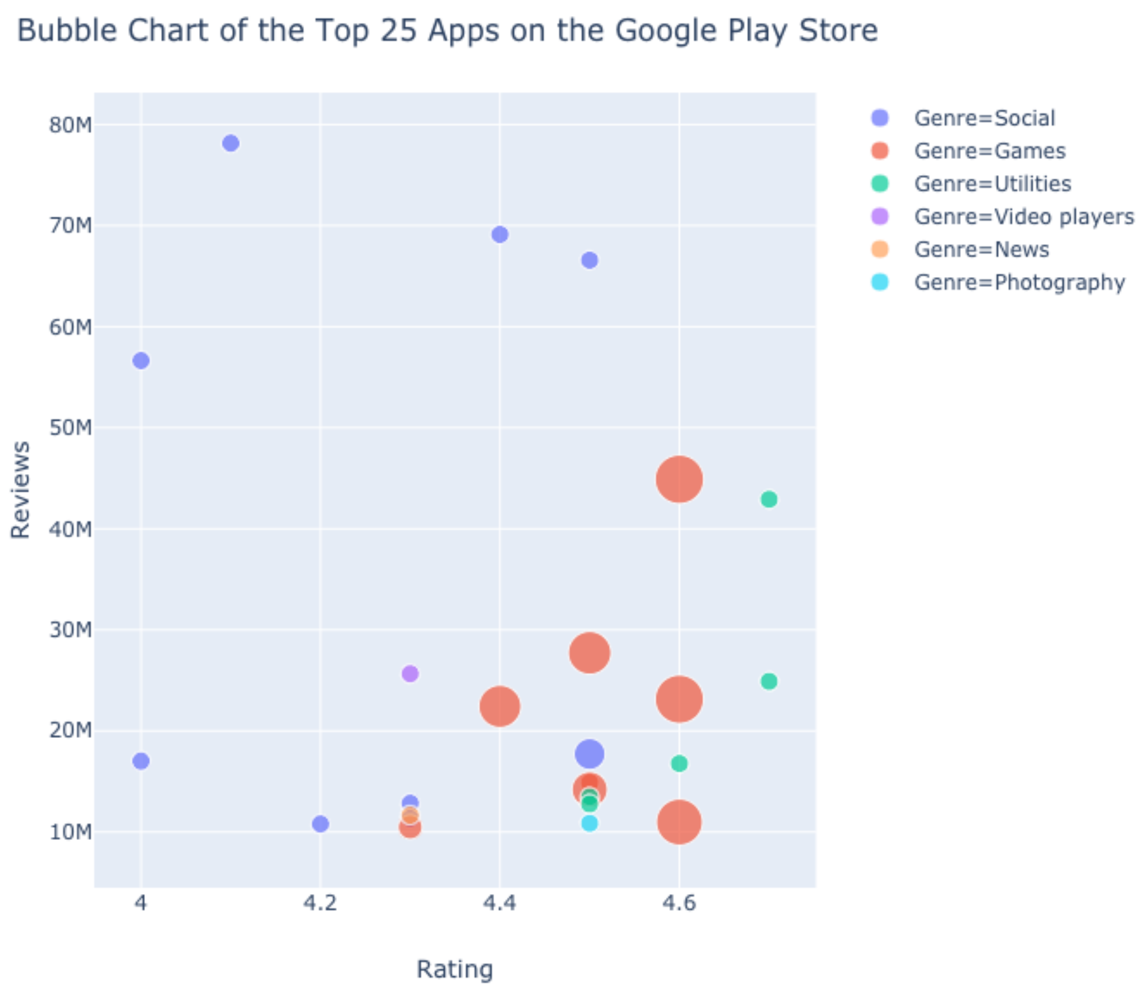
```python
import pandas as pd
full_data = pd.read_csv("cleaned3-5.csv")

apple_data = full_data.loc[full_data['Store'] == "App Store"]
apple_sort_rating = apple_data.sort_values('Reviews', ascending = False)
apple_top25 = apple_sort_rating.iloc[0:25]

google_data = full_data.loc[full_data['Store'] == "Google Play Store"]
google_sort_rating = google_data.sort_values('Reviews', ascending = False)
google_top25 = google_sort_rating.iloc[0:25]
```

The library plotly express was used to generate the bubble charts. A figure was generated with **px.Scatter** with the x-axis being the rating column in the dataframe, y-axis being the number of reviews, size being the app size in bytes and colour corresponding to the genre. The appropriate titles were then applied.

```
#Apple
fig = go.Figure(
    px.scatter(apple_top25, x="Rating", y="Reviews", color="Genre", hover_name="App
", size="Size"))
fig.update_layout(
    title="Bubble Chart of the Top 25 Apps on the App Store")
fig.write_image("bubble_apple_top25.png")
plotly.offline.plot(fig, filename='bubble_apple_top25.html')
fig.show()
```



```python
#Google
fig = go.Figure(
    px.scatter(google_top25, x="Rating", y="Reviews", color="Genre", hover_name="App", size="Size"))
fig.update_layout(
    title="Bubble Chart of the Top 25 Apps on the Google Play Store")
fig.write_image("bubble_google_top25.png")
plotly.offline.plot(fig, filename='bubble_google_top25.html')
fig.show()
```



An interactive version is also provided under the name **'bubble_apple_top25.html'** and **'bubble_google_top25.html**, in which user can hover on each bubble to see the exact values for each parameter of the top 25 apps on both stores.

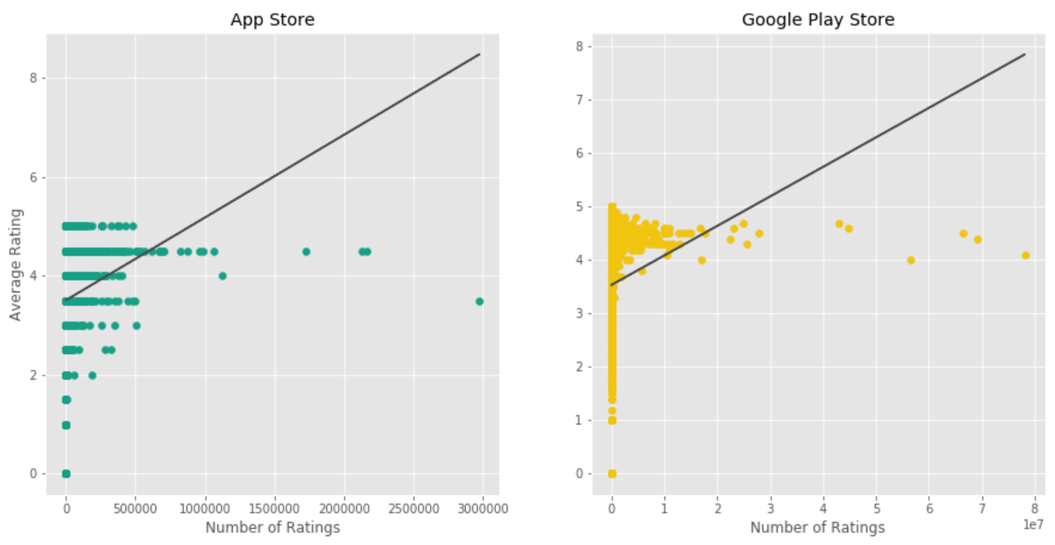
Part Three: Predictive Model

In this part, we introduce two different predictive models: a univariate linear regression model and a multivariate binary classification logistic model. We learned that it is not possible to predict the average rating of an app solely from its number of ratings since there are lurking variables involved. Nevertheless, we managed to predict the "success" of an app by taking into consideration multiple variables including 'Store', 'Genre', 'Content Rating', and 'Price'.

Linear Regression Model predicting average rating from rating count

We wish to predict the average rating of a mobile app from its number of ratings, using a simple univariate linear least-squares regression model. This was done using the **linregress** method from the **scipy.stats** module. This method is useful since we can easily obtain the important values of a linear regression model such as intercept, slope, and correlation coefficients by assigning the method to a variable.

```python
import numpy as np
from scipy.stats import linregress

#Preparing data
y1 = apple_data["Rating"]
x1 = apple_data["Reviews"]

y2 = google_data["Rating"]
x2 = google_data["Reviews"]

#Models
lm_apple = linregress(x1,y1) #(slope, intercept, rvalue, pvalue, stderr)
linear_apple = lm_apple[0]*x1 + lm_apple[1] #regression line

print("Correlation coefficient for App Store datset: {:.2f}".format(lm_apple[2]))

lm_google = linregress(x2,y2) #(slope, intercept, rvalue, pvalue, stderr)
linear_google = lm_google[0]*x2 + lm_google[1] #regression line

print("Correlation coefficient for Google Play Store: {:.2f}".format(lm_google[2]))

#Plot
fig, ax = plt.subplots(1,2, figsize=(15, 7))
ax[0].scatter(x1,y1, color = '#16A086')
ax[0].plot(x1, linear_apple, color = '0.3')

ax[0].set_xlabel("Number of Ratings")
ax[0].set_ylabel("Average Rating")
ax[0].set_title("Figure 3.1.1. App Store")

ax[1].scatter(x2,y2, color = '#F1C40F')
ax[1].plot(x2, linear_google, color = "0.3")
ax[1].set_xlabel("Number of Ratings")
ax[1].set_title("Figure 3.1.2. Google Play Store")

plt.show()
```



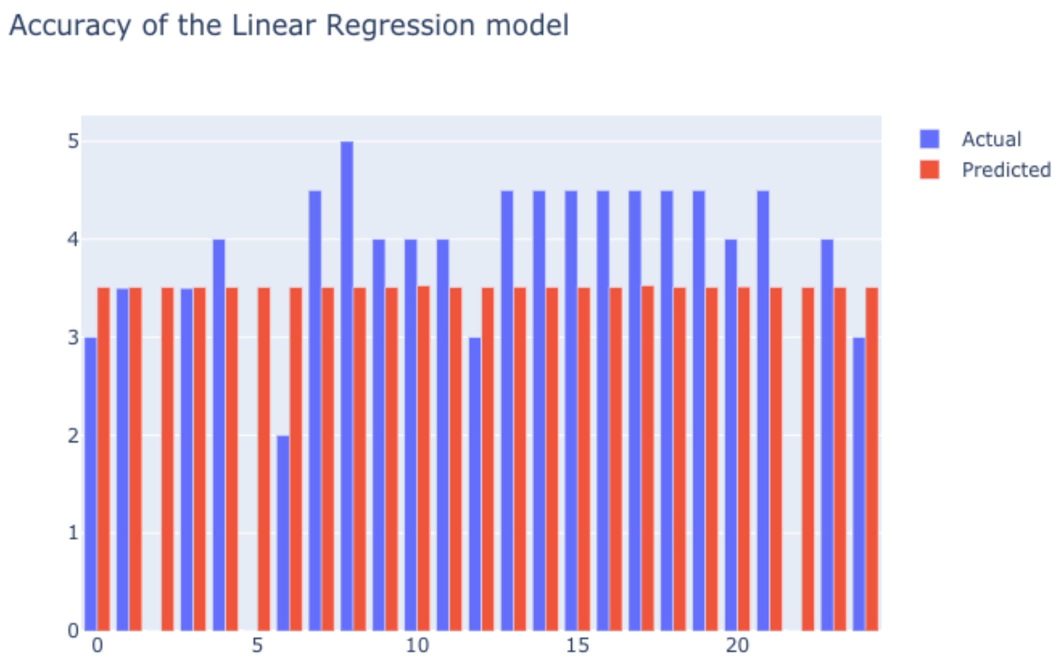
The scatterplots above show the relationship between the number of ratings and the average rating in both application stores. It can be seen that the higher the number of ratings, the higher average rating an app can get. However, the correlation coefficients, which is 0.08 and 0.06 for App Store and Google Play Store, respectively, indicate that there exists a **very weak linear association** between the two variables, and this trend applied to both stores.

Indeed, when implementing the Linear Regression algorithm from the **sklearn** module to evaluate the accuracy of the model for App Store, we quickly learned that the R-squared score for the model is 0.006, given our chosen random state. This information suggests that only 0.6% of the variance found in average rating can be explained by a change of one unit in the number of ratings. There might also exist confounding variables that affect the relationship of the two variables such as user experience, in-app purchase, etc. Therefore, no definite conclusion can be made.

```python
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import train_test_split

X = apple_data["Reviews"].values.reshape(-1,1)
y = apple_data["Rating"].values.reshape(-1,1)

X_train , X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_
state = 44)
regr = linear_model.LinearRegression().fit(X_train, y_train)

y_pred = regr.predict(X_test)

# R-squared score: 1 is perfect prediction
print('R-squared score:', metrics.r2_score(y_test, y_pred))
```

The inaccuracy of model can once again made clear by using **grouped barplot** to visualise the difference between the actual and predicted average ratings of the first 25 apps, whose exact values can be seen by hovering on each bar in our interactive version of the chart under the name **'accuracy-linregress.html'**. It can be seen above that our predicted values do not fluctuate but stay within the range of 3.5. Thus, it can be concluded that it is not possible to predict the average rating of an app from its rating count.

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
df1 = df.head(25)

fig = go.Figure(data = [ go.Bar(name = 'Actual', y = df1['Actual']),
                         go.Bar(name = 'Predicted', y = df1['Predicted'])])

fig.update_traces(hoverinfo='y')
fig.update_layout(title_text = "Accuracy of the Linear Regression model")
fig.write_image("accuracy_linregress.png")
plotly.offline.plot(fig, filename='accuracy_linregress.html')
fig.show()
```



We will now move on to the second model.

Logistic Regression Model for predicting successful mobile application

We now wish to predict app success using Logistic Regression model for binary classification, with our outcome values of **2 being a successful mobile application, and 1 being otherwise**. For simplicity, we consider an app to be successful by having an average rating of 4.0 or over and the number of ratings 10,000 or over. Our feature vector includes 'Store', 'Price', 'Genre', and 'Content Rating'.

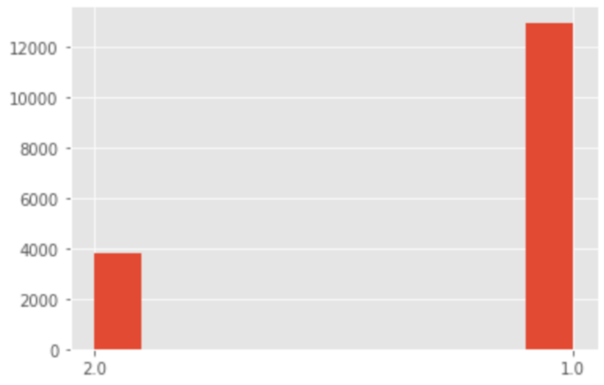
Firstly, for each mobile application in the full dataset, we construct a new column 'Classification' with a Boolean variable of 1 or 2 based on whether or not the app satisfies our requirements of being successful.

```python
df = pd.read_csv("cleaned3-5.csv")
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

df.insert(df.shape[1], "Classification", np.ones(df.shape[0]))

for index, row in df.iterrows():
    if float(row['Rating']) >= 4.0 and int(row['Reviews']) >= 10000:
        df.at[index, "Classification"] = 2

df_train = df[['Store', 'Price', 'Genre', 'Content Rating']]

target = df['Classification']
print(target.value_counts())

a = target.astype(str).hist()
```

We then use **matplotlib** to graph a histogram to visualise the number of 'successful' mobile apps in our dataset.

It appears that there are 3,791 apps satisfy our requirements to be considered successful, note that this includes duplicates of the same mobile app in two stores.

```
1.0     12986
2.0      3791
Name: Classification, dtype: int64
```



Given that the features 'Store', 'Genre', and 'Content Rating' are of categorical values, we need to transform these data into numerical values so that the **linear_model.LogisticRegression** algorithm can process them. We do this by mapping each value with dictionary comprehension (replace method was adopted from this website). For instance, since there are 30 genres in total in the full dataset, each genre will be associated to an integer from 1 to 30. The same method applies to 'Store' and 'Content Rating'.

```python
#Replace values
replace_app = {'Store': {'App Store': 1, 'Google Play Store': 2}}

labels_genre = df['Genre'].astype('category').cat.categories.tolist()
replace_genre = {'Genre' : {k: v for k,v in zip(labels_genre,list(range(1,len(label
s_genre)+1)))}}

labels_content = df['Content Rating'].astype('category').cat.categories.tolist()
replace_content = {'Content Rating': {k: v for k,v in zip(labels_content,list(range
(1,len(labels_content)+1)))}}

df_replace = df_train.copy()

df_replace.replace(replace_app, inplace = True)
df_replace.replace(replace_genre, inplace = True)
df_replace.replace(replace_content, inplace = True)
```

```

#Preprocess data
df_train.head(n=5)
#Post-process data
df_replace.head(n=5)
```

Our training data pre-processed (right) and post-processed (left) can be seen as follows:

	Store	Price	Genre	Content Rating
0	App Store	3.99	Games	Everyone
1	App Store	0.00	Productivity	Everyone
2	App Store	0.00	Weather	Everyone
3	App Store	0.00	Shopping	Teen
4	App Store	0.00	Reference	Everyone

	Store	Price	Genre	Content Rating
0	1	3.99	13	1
1	1	0.00	22	1
2	1	0.00	30	1
3	1	0.00	24	3
4	1	0.00	23	1

We can now assign our dataframes to the independent variable X and response variable y in preparation for our machine learning model. We then split this data into a training set and a test set for future validation.

```
X = df_replace.values
y = target.values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_s
tate = 42)
```

The following process for Logistic Regression was adopted from Grok Learning, 'Common Algorithms in Machine Learning' presented by professor Alan Fekete. We used the **LogisticRegression** method from the **linear_model** module to fit our training data.

```
import random
clf = linear_model.LogisticRegression(solver='liblinear').fit(X_train, y_train)
```

To validate our model, we use the predict method on our clf object to predict the classification of an app using our test data. Note that the model has not seen the test set before, thus we will be able to assume whether or not our model generalise.

Below is a sample case from taken from our testing data. This case was chosen randomly by using the **randint** method from **random** module. For each feature value given, our model produces the outcome values in the form of a list of length 2. Within the list is a probability of an app to be of class 1 (not successful) or 2 (successful), whichever closest to one will be classified as such class.

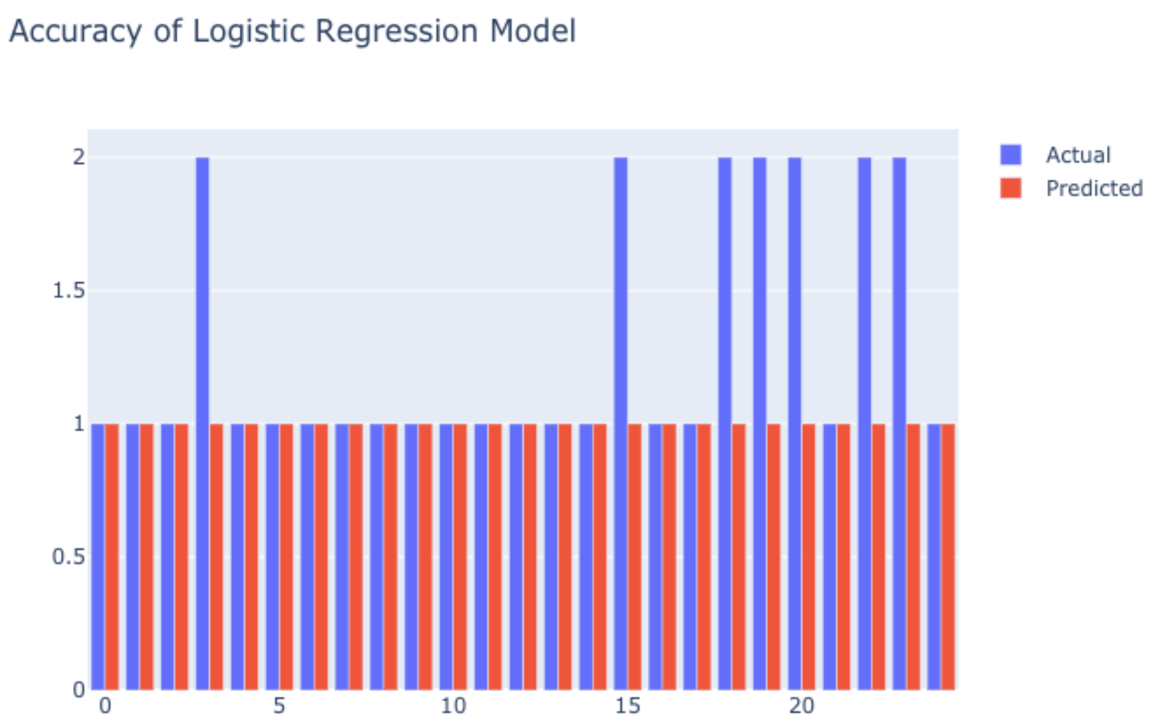
```
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)

```

```
print('----- Sample case -----')
num = random.randint(0, len(y_test))

sample = X_test[num]
for column, value in zip(list(df_replace), sample):
    print(column + ': ' + str(value))

last_sample_proba = y_pred_proba[num]
print('Probability of class 1:', last_sample_proba[0])
print('Probability of class 2:', last_sample_proba[1])
print('Actual class:', int(y_test[num]))
print('where 2 means a successful app, 1 means otherwise')
print('-----------------------')
# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
----- Sample case -----
Store: 2.0
Price: 0.0
Genre: 11.0
Content Rating: 3.0
Probability of class 1: 0.611368456196625
Probability of class 2: 0.38863154380337495
Actual class: 1
where 2 means a successful app, 1 means otherwise
-----------------------
Accuracy: 0.7699642431466031
```

By using the **accuracy_score** method from the **metrics** module to evaluate our model, we can see that the performance has improved considerably compared to our previous model. Once again, we can visualise our actual and predicted classification of each app (for the first 25 apps) using grouped bar chart. Two red and blue bars of the same height is a correct prediction. An interactive version is also provided under the name **'accuracy_logistic.html'**.

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})

df1 = df.head(25)

fig = go.Figure(data = [ go.Bar(name = 'Actual', y = df1['Actual']), go.Bar(name = 
'Predicted', y = df1['Predicted'])])
fig.update_layout(title_text = "Accuracy of Logistic Regression Model")
fig.write_image("accuracy_logistic.png")
plotly.offline.plot(fig, filename='accuracy_logistic.html')
fig.show()
```



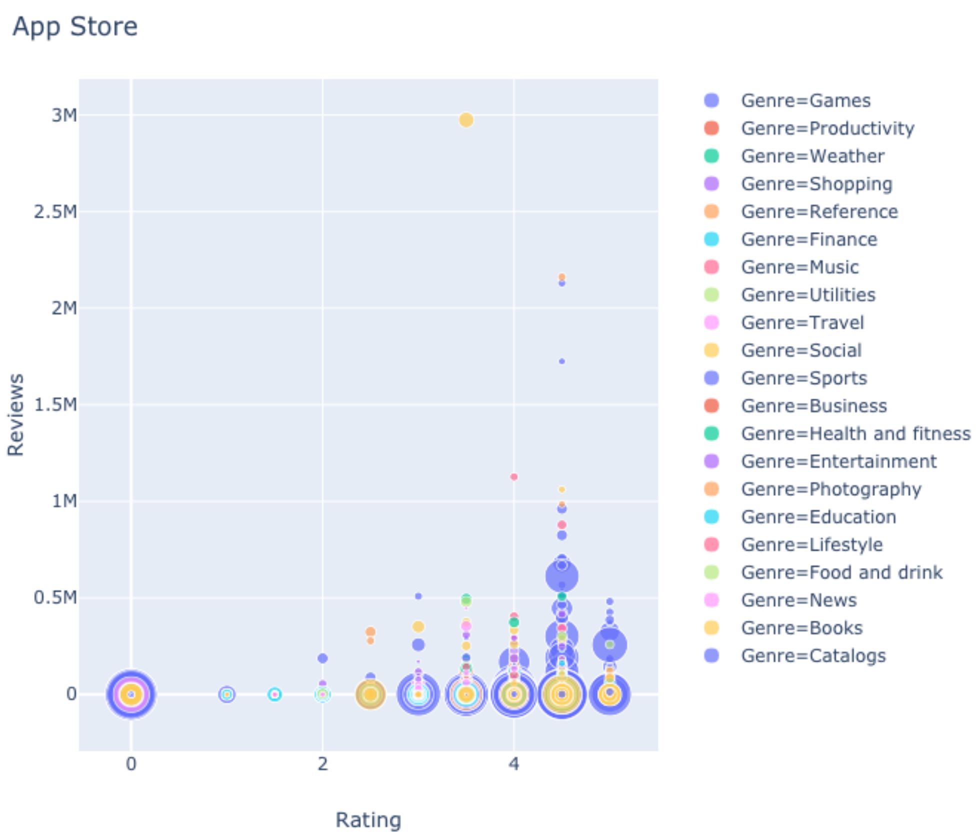
For our particular experience, we learned that by adding more relevant variables to our model, our predictive analysis improved greatly. This assumption can easily be scaled to the real word as one variable is rarely dependent on just one another variable. For our domain of interest, we can conclude that a mobile app's average rating is not dependent on its number of ratings alone. There are many other factors at play such as user experience, the number of bugs, frequency of ads, etc. However, our initial statement is a very general assumption and may not apply to other circumstances.

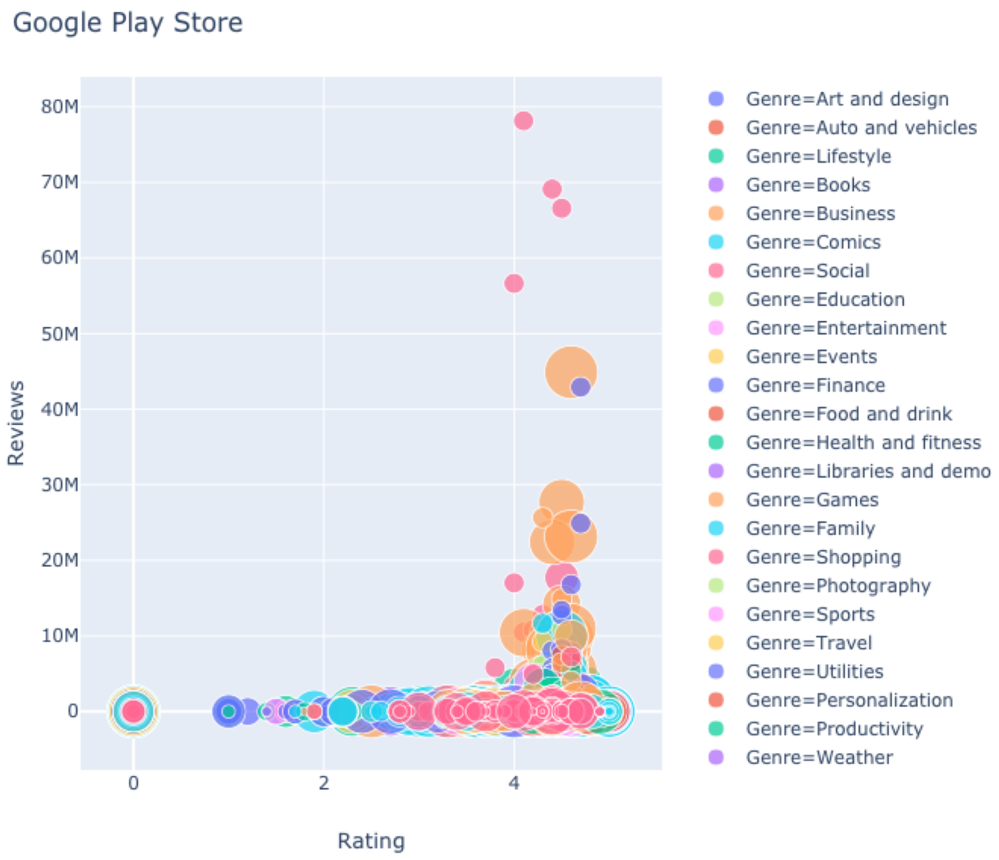
Part Four: Interactive Visualisation

For our first interactive visualisation, we invite users to explore the relationship between four variables including 'Average Rating', 'Number of Ratings', 'Size', and 'Genre', separately for each store. This was done quite conveniently using the module **plotly.express.**

Each dot represents an app, with its x coordinate corresponds to its average rating, y to the number of ratings, size to the physical size of the app, and colour to its genre. Users can play around with the dataset by zooming in to certain section or hover over the dots to see the information about each app. Interactive files are attached under the name **'google_interactive.html'** and **'apple_interactive.html'**.

```python
 import plotly.express as px

#Apple:
fig = px.scatter(apple_data, x="Rating", y="Reviews", color="Genre",size="Size", hover_name="App",hover_data=None)
fig.update_layout(title_text = "App Store")
fig.write_image("apple_interactive.png")
plotly.offline.plot(fig, filename='apple_interactive.html')
fig.show()

#Google:
fig = px.scatter(google_data, x="Rating", y="Reviews", color="Genre",size="Size", hover_name="App")
fig.update_layout(title_text = "Google Play Store")
fig.write_image("google_interactive.png")
plotly.offline.plot(fig, filename='google_interactive.html')
fig.show()
```


App Store
Reviews
3M
2.5M
2M
1.5M
1M
0.5M
0
Rating
0
2
4
Genre=Games
Genre=Productivity
Genre=Weather
Genre=Shopping
Genre=Reference
Genre=Finance
Genre=Music
Genre=Utilities
Genre=Travel
Genre=Social
Genre=Sports
Genre=Business
Genre=Health and fitness
Genre=Entertainment
Genre=Photography
Genre=Education
Genre=Lifestyle
Genre=Food and drink
Genre=News
Genre=Books
Genre=Catalogs

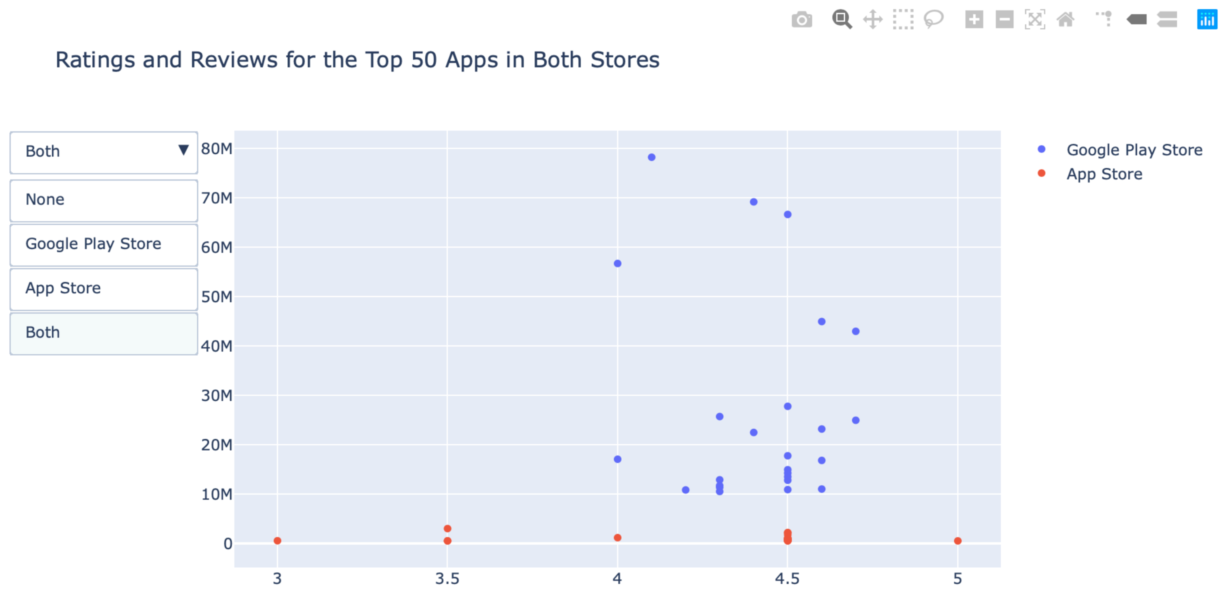

Google Play Store
Reviews
80M
70M
60M
50M
40M
30M
20M
10M
0
Rating
0
2
4
Genre=Art and design
Genre=Auto and vehicles
Genre=Lifestyle
Genre=Books
Genre=Business
Genre=Comics
Genre=Social
Genre=Education
Genre=Entertainment
Genre=Events
Genre=Finance
Genre=Food and drink
Genre=Health and fitness
Genre=Libraries and demo
Genre=Games
Genre=Family
Genre=Shopping
Genre=Photography
Genre=Sports
Genre=Travel
Genre=Utilities
Genre=Personalization
Genre=Productivity
Genre=Weather

For our second interactive plot, we want users to explore the average rating and number of ratings in the top 50 apps in each dataset and both at once. Users can choose specifically to show one or both datasets. Datasets were prepared using the **pandas** module. After separating the App Store and Google Play Store dataset, each was then sorted by the number of reviews, in a descending manner. By doing that, we can then quickly extract the first 50 columns to get the top 50 most popular apps in each dataset.

```python
full_data = pd.read_csv("cleaned3-5.csv")
apple_data = full_data.loc[full_data['Store'] == "App Store"]
google_data = full_data.loc[full_data['Store'] == "Google Play Store"]

apple_sort_rating = apple_data.sort_values('Reviews', ascending = False)
apple_top50 = apple_sort_rating.iloc[0:50]

google_sort_rating = google_data.sort_values('Reviews', ascending = False)
google_top50 = google_sort_rating.iloc[0:50]
```

For graphing utilities, we used the **plotly.graph_objects** module. The **updatemenus** method was utilised to create a dropdown menu that allows users to modify the scatterplot. Interactive version is attached under the name **'both_interactive.html'**.

```python
import plotly.graph_objects as go
import pandas as pd
fig = go.Figure()

fig.add_trace(
    go.Scatter(x=google_top50["Rating"], y=google_top50["Reviews"], text = google_top50["App"], name = "Google Play Store", mode="markers"))
fig.add_trace(
    go.Scatter(x=apple_top50["Rating"], y=apple_top50["Reviews"], text = apple_top50["App"], name = "App Store", mode="markers"))
fig.update_layout(
    updatemenus=[
        go.layout.Updatemenu(
            active = 0,
            buttons = list([
                dict(label="None",
                    method = "update",
                    args=[{"visible":[False,False]},
                         {"title": "Ratings and Reviews"}]),
                dict(label="Google Play Store",
                    method = "update",
                    args=[{"visible":[True,False]},
                         {"title": "Google Play Store Ratings and Reviews"}]),
                dict(label="App Store",
                    method = "update",
                    args=[{"visible":[False,True]},
                         {"title": "App Store Ratings and Reviews"}]),
                dict(label="Both",
                    method = "update",
                    args=[{"visible":[True,True]},
                         {"title": "Ratings and Reviews for the Top 50 Apps in Both Stores"}])
            ]))
    ])
plotly.offline.plot(fig, filename='both_interactive.html')
fig.show()
```



The approach used in designing the interactive visualisation was chosen as it provided charts which could display many variables at once in a visually appealing manner. The interactive bubble charts contain many useful features such as being able to zoom and pan as well as hover over each bubble to view what it represents. Users can also filter subsets of the data by clicking on the genre(s) they wish to analyse. The bubble chart also compares a multitude of variables and is hence rich in information from the dataset. The scatter plot on the other hand also contains functionality such as being able to zoom and pan while providing detail on hover. Extra functionality in the choice of store was added and can be made here using a simple drop-down menu. This allows the user to analyse the top 50 apps in the Google Play Store and App Store either together or individually.

References

Dogtiev, A. (2019, September). *App Store List (2019).* Business of Apps. Retrieved from: https://www.businessofapps.com/guide/app-stores-list/

Gupta, L. (2018, September). *Google Play Store Apps.* Kaggle. Retrieved from: https://www.kaggle.com/lava18/google-play-store-apps

Pathak, M. (2018, May). *Handling categorical data in Python.* Datacamp. Retrieved from: https://www.datacamp.com/community/tutorials/categorical-data

Ramanathan. (2018, June). *Mobile App Statistics (Apple iOS app store).* Kaggle. Retrieved from: https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps

ReviewTrackers. (n.d.). *2018 ReviewTrackers Online Reviews Survey.* ReviewTrackers. Retrieved from: https://www.reviewtrackers.com/reports/online-reviews-survey/

Sefferman, A. (2016, June). *Mobile Ratings: The Good, the Bad, and the Ugly.* Apptentive. Retrieved from: https://www.apptentive.com/blog/2016/06/23/mobile-ratings-good-bad-ugly/