DATS6401

VISUALIZATION OF COMPLEX DATA

Dr. Reza Jafari

Final Term Project

Nusrat Nawshin

NN

03/29/2022

Table of Content

Table of Figures and Tables

Abstract:

The objective of this project is to apply course learning objectives to visualize complex data using Python and an interactive web-based dashboard.

Introduction:

Do you like to look at enormous data tables and then decipher the information, or do you prefer to look at a data visualization that delivers the information in an easy-to-understand visual format? Most of you, on the other hand, would like data visualization! That is because data visualization is incredibly helpful in comprehending data and gaining valuable insights. It can help you acquire a quick comprehension of the data that you would not be able to get by looking at rows of data in a table. That is why it is so crucial for Data Scientists! Data visualization's major purpose is to make it simpler to see patterns, trends, and outliers in massive data sets. Information graphics, information visualization, and statistical graphics are all terms that are frequently used interchangeably with one other.

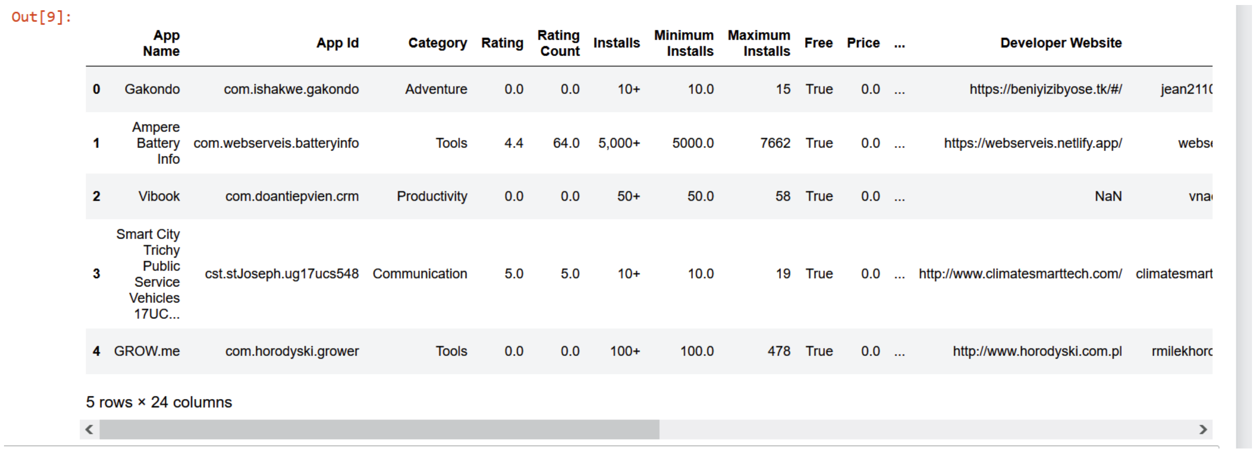
In this project, I will use a dataset from Kaggle named 'Google Play Store Apps'. At first, I preprocess the dataset for better visualization. Then checked that if there is any outlier in the dataset and then remove the outlier. After the data cleaning, I have completed the following tasks named, Principal Component Analysis (PCA) for checking dimensionality reduction, Normality test for gaussian dataset or not, Heatmap and Pearson correlation coefficient matrix for scanning the relationship among the numerical variables, and few statistics about the datasets. After inspecting the dataset, I will visualize the dataset using Seaborn. Here I have visualized this dataset using line plot, bar plot, scatter plot, cat plot, Displot, Pairplot, Kernel Density estimate plot, etc. I would also visualize data statistics using subplots for more effective comparison.

Farther develop a web-based app using a dash package so that I can visualize the dataset interactively. The dash app contains 5 tabs. The first tab is statistics, I showed some statistics about our dataset with some filters so that user can change and filter the dataset interactively. In the second tab, user can compare two apps, and based on some points I calculated the comparison result and decide which app is better. In the third tab, user would find the apps list with some filtering options. User can also download the filtered dataset in csv format and see the details of a single app. In the fourth tab, I am visualizing the data with filter option of specific graph type, and finally in the fifth tab, there are normality tests options on the numeric columns. The app is deployed on Google Cloud Platform, you can visit the website on this link: Dash (dashapp-s4afy3ytoq-rj.a.run.app)

Description of the dataset:

Every year, a lot of mobile applications are released in different app stores. It is really difficult to keep track of which type of application has already developed and which category apps are becoming popular now-a-days. From this aspect I selected a dataset which includes all the apps which were released on the google play store and will visualize this dataset so that we can understand the current condition of the app store world.

The dataset is collected from Kaggle [1]. The Author collected the data by the help of python scripts and collected it in the month of June 2021. The name of the dataset is “Google Play Store Apps”, and it contains almost 2.3+ million applications information (Figure 1) and out of 25 columns 19 columns are categorical and 5 columns are numerical. The column list and their datatype are shown below (figure 2).

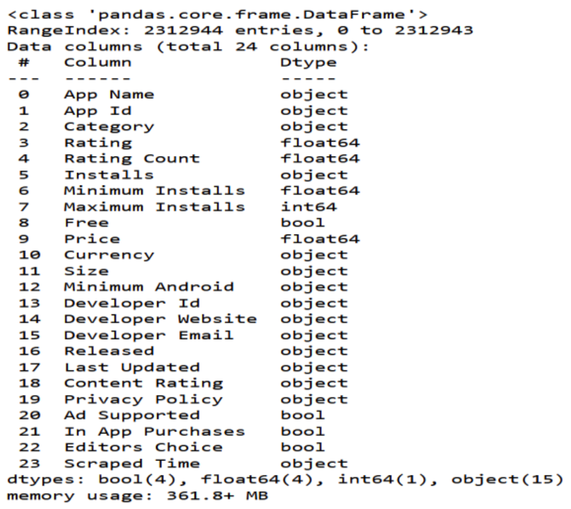


Out[9]:

	App Name	App Id	Category	Rating	Rating Count	Installs	Minimum Installs	Maximum Installs	Free	Price	...	Developer Website	
0	Gakondo	com.ishakwe.gakondo	Adventure	0.0	0.0	10+	10.0	15	True	0.0	...	https://beniyizibyose.tk/#/	jean211
1	Ampere Battery Info	com.webserveis.batteryinfo	Tools	4.4	64.0	5,000+	5000.0	7662	True	0.0	...	https://webserveis.netlify.app/	webs
2	Vibook	com.doantiepvien.crm	Productivity	0.0	0.0	50+	50.0	58	True	0.0	...	NaN	vna
3	Smart City Trichy Public Service Vehicles 17UC...	cst.stJoseph.ug17ucs548	Communication	5.0	5.0	10+	10.0	19	True	0.0	...	http://www.climatesmarttech.com/	climatesmart
4	GROW.me	com.horodyski.grower	Tools	0.0	0.0	100+	100.0	478	True	0.0	...	http://www.horodyski.com.pl	rmilekhorc

5 rows × 24 columns

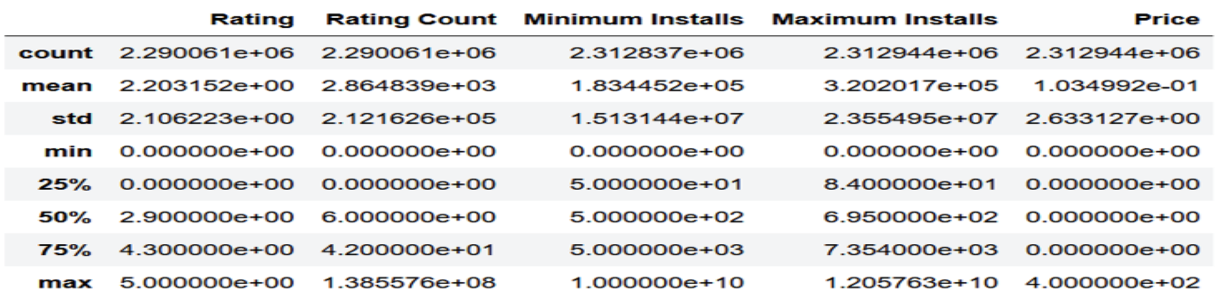
Figure 1: 1st five records of the dataset



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2312944 entries, 0 to 2312943
Data columns (total 24 columns):
 #   Column             Dtype
---  ------             -----
 0   App Name           object
 1   App Id             object
 2   Category           object
 3   Rating             float64
 4   Rating Count       float64
 5   Installs           object
 6   Minimum Installs   float64
 7   Maximum Installs   int64
 8   Free               bool
 9   Price              float64
 10  Currency           object
 11  Size               object
 12  Minimum Android    object
 13  Developer Id       object
 14  Developer Website  object
 15  Developer Email    object
 16  Released           object
 17  Last Updated       object
 18  Content Rating     object
 19  Privacy Policy     object
 20  Ad Supported       bool
 21  In App Purchases   bool
 22  Editors Choice     bool
 23  Scraped Time       object
dtypes: bool(4), float64(4), int64(1), object(15)
memory usage: 361.8+ MB
```

Figure 2: Dataset column data type

In figure 3 there are the statistical information of numerical columns.

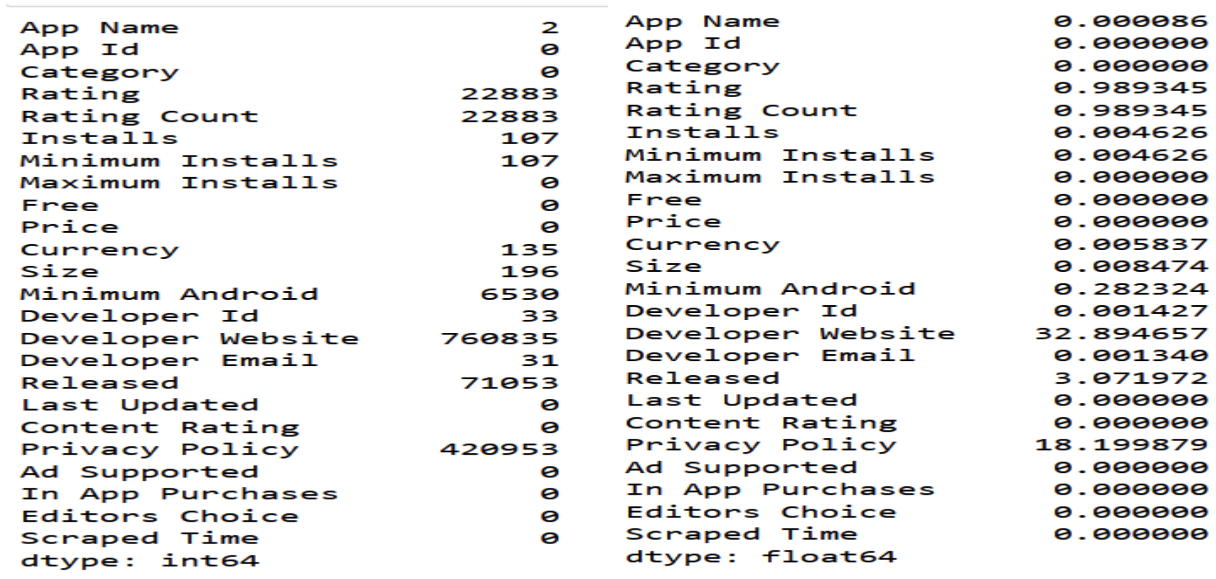


	Rating	Rating Count	Minimum Installs	Maximum Installs	Price
count	2.290061e+06	2.290061e+06	2.312837e+06	2.312944e+06	2.312944e+06
mean	2.203152e+00	2.864839e+03	1.834452e+05	3.202017e+05	1.034992e-01
std	2.106223e+00	2.121626e+05	1.513144e+07	2.355495e+07	2.633127e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	5.000000e+01	8.400000e+01	0.000000e+00
50%	2.900000e+00	6.000000e+00	5.000000e+02	6.950000e+02	0.000000e+00
75%	4.300000e+00	4.200000e+01	5.000000e+03	7.354000e+03	0.000000e+00
max	5.000000e+00	1.385576e+08	1.000000e+10	1.205763e+10	4.000000e+02

Figure 3: Statistics of Numerical Columns Data

Preprocessing Dataset:

Preprocessing the dataset is one the most important tasks in the data visualization and analysis. Raw data collected from a source is rarely perfect. There are always either some missing values in the dataset, or some data are inconsistent than other data. So, for better visualization, it is particularly important to preprocess the data by some techniques. For example, remove the data from the dataset, replace the data set with a common (mean or median) value or drop the whole column. In this dataset, I calculated and found the total percentage of missing value and the value is 2.35%. From figure 4 we can see that 'Developer Website' and 'Privacy Policy' have a lot of missing values. In the percentage it is almost 32% and 18%.

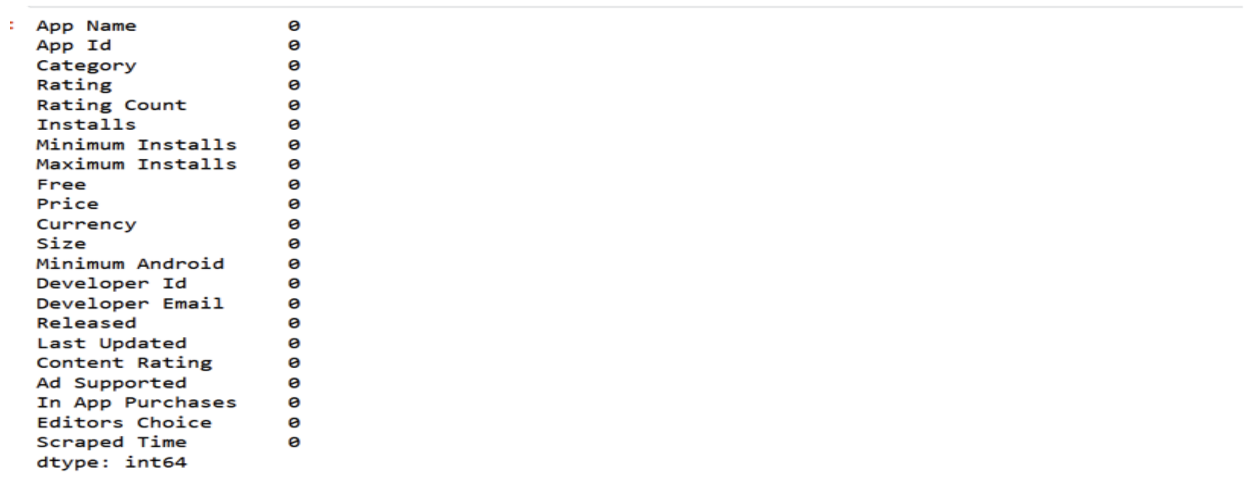


```
App Name                      2
App Id                        0
Category                      0
Rating                    22883
Rating Count              22883
Installs                    107
Minimum Installs            107
Maximum Installs              0
Free                          0
Price                         0
Currency                    135
Size                        196
Minimum Android            6530
Developer Id                 33
Developer Website        760835
Developer Email              31
Released                  71053
Last Updated                  0
Content Rating                0
Privacy Policy           420953
Ad Supported                  0
In App Purchases              0
Editors Choice                0
Scraped Time                  0
dtype: int64
```

```
App Name              0.000086
App Id                0.000000
Category              0.000000
Rating                0.989345
Rating Count          0.989345
Installs              0.004626
Minimum Installs      0.004626
Maximum Installs      0.000000
Free                  0.000000
Price                 0.000000
Currency              0.005837
Size                  0.008474
Minimum Android       0.282324
Developer Id          0.001427
Developer Website    32.894657
Developer Email       0.001340
Released              3.071972
Last Updated          0.000000
Content Rating        0.000000
Privacy Policy       18.199879
Ad Supported          0.000000
In App Purchases      0.000000
Editors Choice        0.000000
Scraped Time          0.000000
dtype: float64
```

Figure :4 (left) Column wise total missing values and (right) the percentage of missing values

For this high percentage of missing values, I am dropping these two columns from the dataset. Now the percentage of total missing values drops from 2.35 % to 0.24 %. In the next step, I replaced the numerical missing data with their mean value. For the categorical data I have dropped the corresponding row value. In the end there are no missing values in the dataset (figure5).



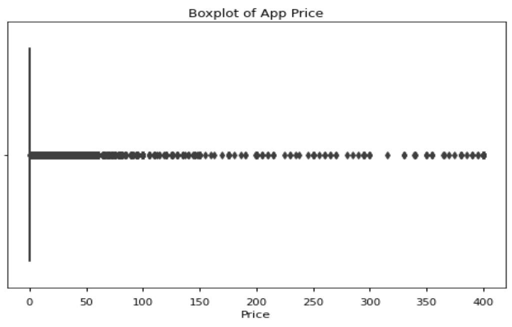
```
: App Name              0
  App Id                0
  Category              0
  Rating                0
  Rating Count          0
  Installs              0
  Minimum Installs      0
  Maximum Installs      0
  Free                  0
  Price                 0
  Currency              0
  Size                  0
  Minimum Android       0
  Developer Id          0
  Developer Email       0
  Released              0
  Last Updated          0
  Content Rating        0
  Ad Supported          0
  In App Purchases      0
  Editors Choice        0
  Scraped Time          0
  dtype: int64
```

Figure 5: Column missing value counts

For the web-based dashboard app, I have cleaned the dataset further. At first, I convert the 'Installs' column object to int64 so that I can visualize it more effectively. Then I filtered the data based on the number of installs. I am considering the popular apps by considering the apps that has been installed more than 1,00,000 times. Farther I have filtered the dataset based on rating count, if the 'Rating Count' is more than 4500 then I am only considering those apps. After that without dropping any rows, I am checking if any categorical column has missing values, then replaced those values by 'No Information' and numerical values we are replaced by 0.0. I also drop two unnecessary columns from the dataset. those columns are 'Scraped Time' and 'Last Update'

Outlier detection and removal:

Outliers are data points that are far from other data points or unusual values in a dataset. Outliers are problematic for many statistical analyses because they can cause tests to either miss significant findings or distort actual results. Unfortunately, there are no strict statistical rules for definitively identifying outliers. Finding outliers depends on subject-area knowledge and an understanding of the data collection process. Visualizing using boxplot is one of the graphical ways of detecting outliers. After visualizing the distribution using boxplot (Figure 6), here we can see that if we remove the outliers from price column it will have only zero prices but considering this dataset these values are not outliers as there are very few apps that are paid in the play store. On the other side, App Rating column does not have any outliers but its imbalanced.



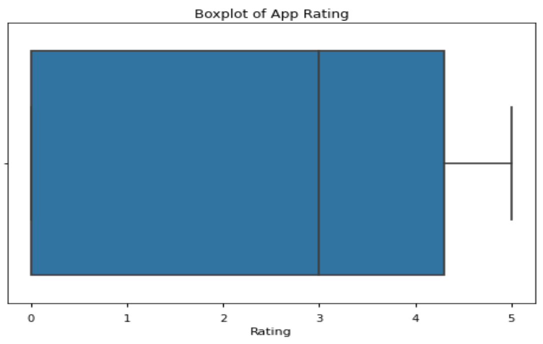


Figure 6: Boxplot of App Price and App Rating

After removing outliers from the other three numeric columns (figure 7), there is still some outliers present. But removing more data will remove too much data.

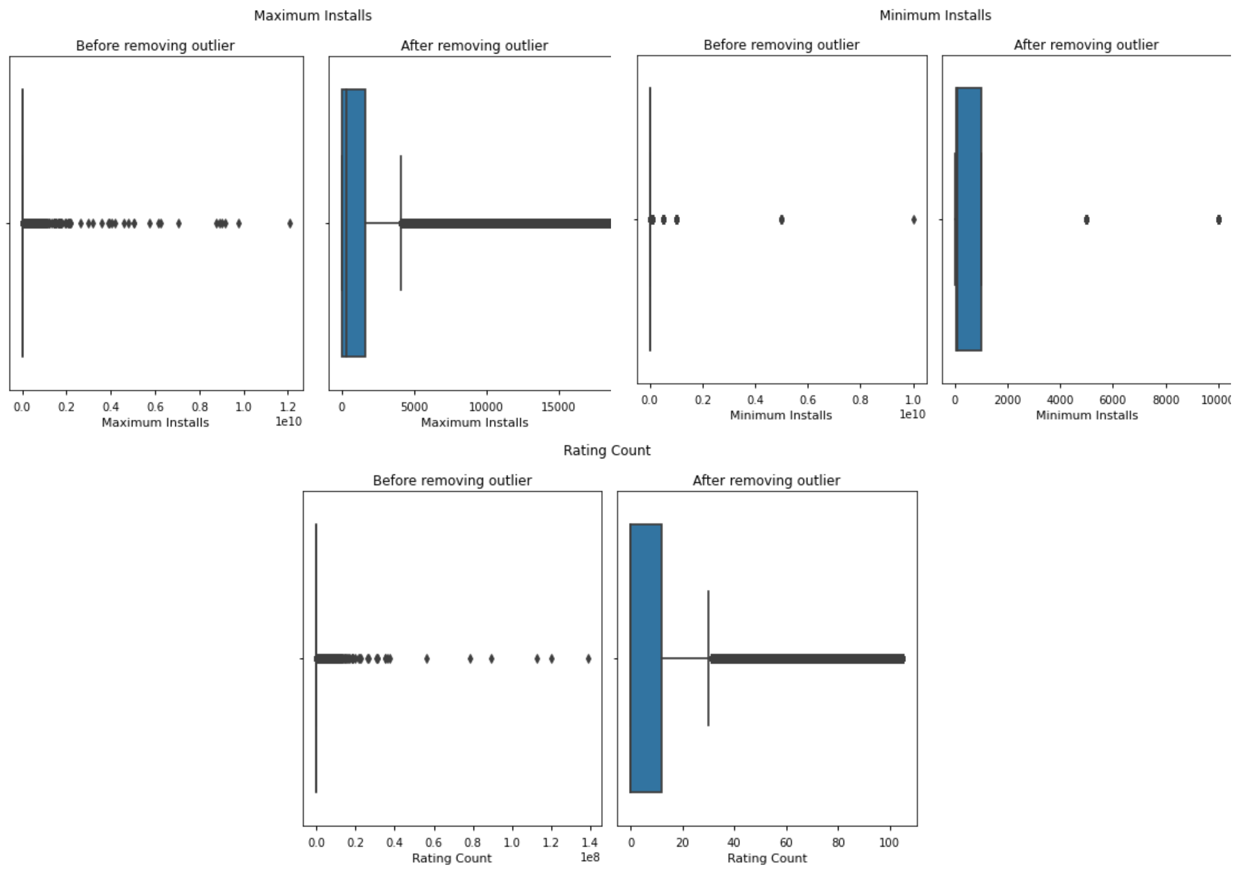
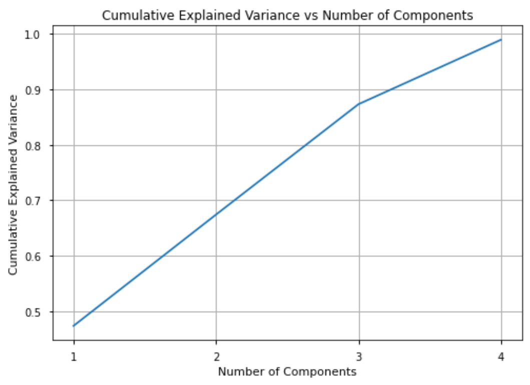


Figure 7: Boxplot of before and after removing outliers (Maximum Installs, Minimum Installs, Rating Count)

After removing the outliers, the dataset now consists of 17,82,652 number of rows and 22 columns.

Principal Component Analysis (PCA):

Principal component analysis, or PCA, is a statistical procedure that summarize the information content in large data tables by means of a smaller set of "summary indices" that can be more easily visualized and analyzed [1]. To perform PCA analysis on this dataset only the five numeric columns are used and using python SKlearn package PCA () function with 'mle' as n_component argument it reduces the feature space from 5 to 4. But from the cumulative explained variance vs number of component graph (figure 8) it can be observed that even with only three features we can get almost 90% explained variance. So, I am removing one more feature and the reduced feature space is now with three features and with almost 90% explained variance.



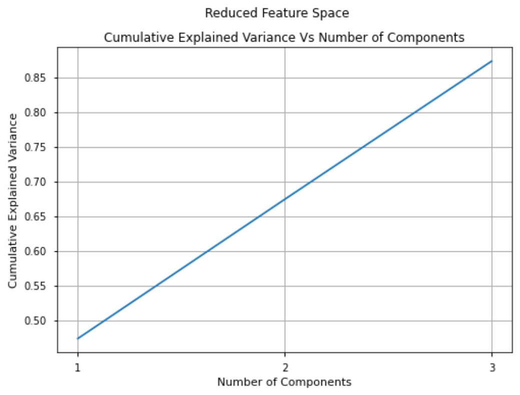
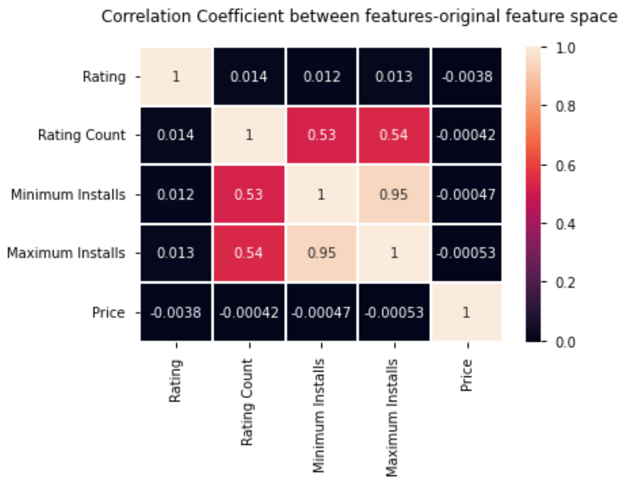
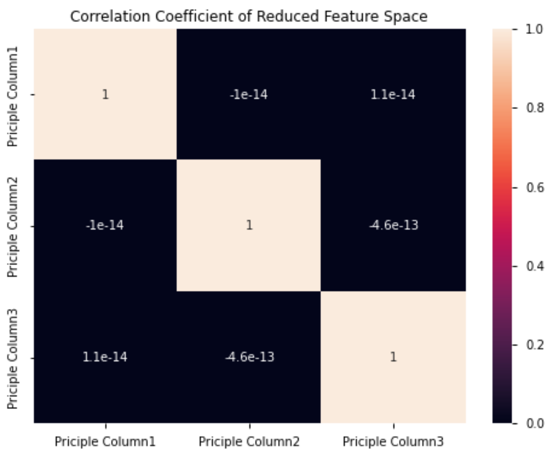


Figure 8: PCA-Cumulative Explained Variance vs Number of Components

From the correlation matrix of the original feature space and the PCA reduced feature space (figure 9), we can see that there were two medium correlated features and PCA removed that on the reduced feature space.



	Rating	Rating Count	Minimum Installs	Maximum Installs	Price
Rating	1	0.014	0.012	0.013	-0.0038
Rating Count	0.014	1	0.53	0.54	-0.00042
Minimum Installs	0.012	0.53	1	0.95	-0.00047
Maximum Installs	0.013	0.54	0.95	1	-0.00053
Price	-0.0038	-0.00042	-0.00047	-0.00053	1



	Priciple Column1	Priciple Column2	Priciple Column3
Priciple Column1	1	-1e-14	1.1e-14
Priciple Column2	-1e-14	1	-4.6e-13
Priciple Column3	1.1e-14	-4.6e-13	1

Figure 9: PCA-Correlation matrix of original feature space and reduced feature space

Data transformation:

As the column 'Install' was as string, I am converting it to numeric by removing the '+' or '.'

Normality Test:

There are two primary methods to find normal distribution on a dataset. Graphical method and statistical test. Histogram and QQ plot are two popular and effective method to identify normal distribution. Shapiro-test, Kolmogorov-Smirnov-test, D'Agostino-Pearson Test are the three popular statistical methods to identify normal distribution. All three test considers null hypothesis on normal distribution.

Now the dataset has 6 numeric columns, Install, Rating, Rating Count, Minimum Installs, Maximum Installs, and price. Test results of all normality tests indicated the distribution of these 5 columns aren't normal as the p value were significant (reject the null hypothesis) (table 1).

Table 1: Normality Test

Column Name	Test Name	Test Statistics	P-Value	Comment
Minimum Installs	Shapiro-test	0.029	0	Sample doesn't look Gaussian (reject H0)
	Kolmogorov-Smirnov	1	0	Sample doesn't look Gaussian (reject H0)
	D'Agostino-Pearson	162034.7	0	Sample doesn't look Gaussian (reject H0)
Maximum Installs	Shapiro	0.036	0	Sample doesn't look Gaussian (reject H0)
	Kolmogorov-Smirnov	1	0	Sample doesn't look Gaussian (reject H0)
	D'Agostino-Pearson	153416.5	0	Sample doesn't look Gaussian (reject H0)
Rating	Shapiro	0.898	0	Sample doesn't look Gaussian (reject H0)
	Kolmogorov-Smirnov	0.985	0	Sample doesn't look Gaussian (reject H0)
	D'Agostino-Pearson	15099.6	0	Sample doesn't look Gaussian (reject H0)
Rating Count	Shapiro	0.036	0	Sample doesn't look Gaussian (reject H0)
	Kolmogorov-Smirnov	1	0	Sample doesn't look Gaussian (reject H0)
	D'Agostino-Pearson	173807.9	0	Sample doesn't look Gaussian (reject H0)
Price	Shapiro	0.026	0	Sample doesn't look Gaussian (reject H0)
	Kolmogorov-Smirnov	0.5	0	Sample doesn't look Gaussian (reject H0)
	D'Agostino-Pearson	134385.4	0	Sample doesn't look Gaussian (reject H0)
Installs	Shapiro	0.029	0	Sample doesn't look Gaussian (reject H0)
	Kolmogorov-Smirnov	1	0	Sample doesn't look Gaussian (reject H0)
	D'Agostino-Pearson	162034.7	0	Sample doesn't look Gaussian (reject H0)

In the graphical tests, both the QQ-plot (figure 10) and histogram (figure 11) are indicating non normality in the features.

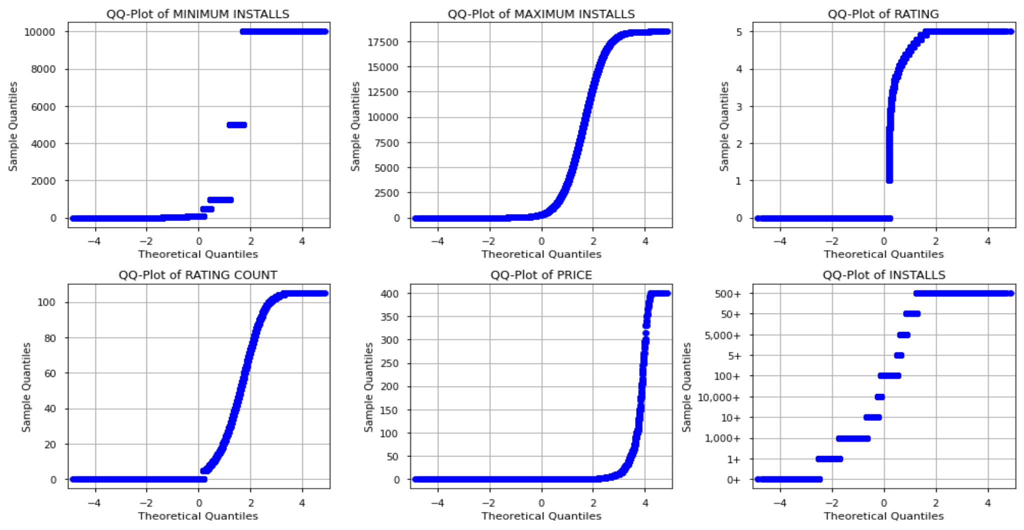


Figure 10: QQ-Plot of numeric columns

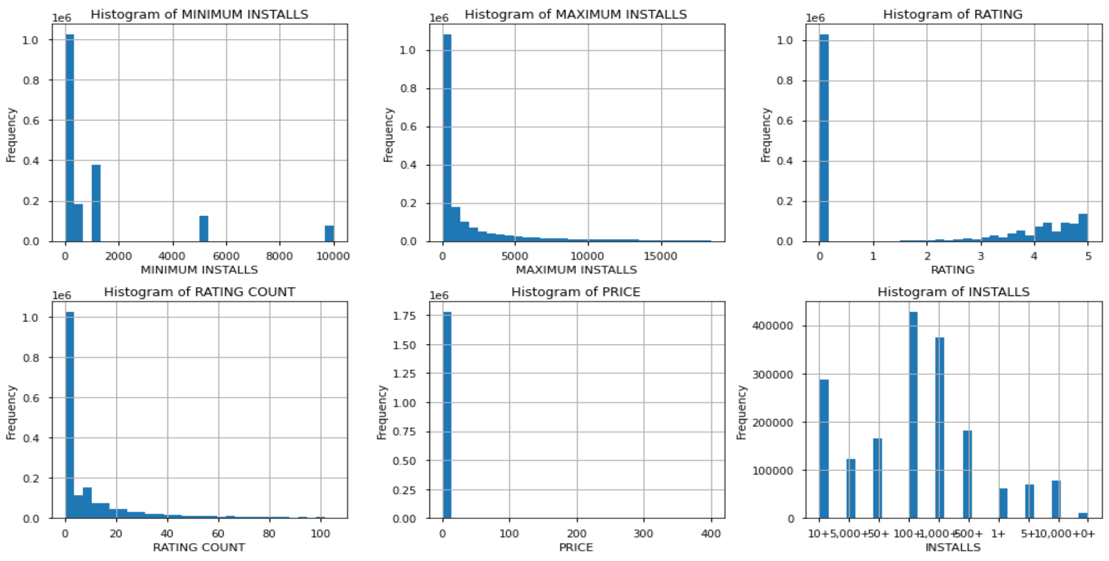
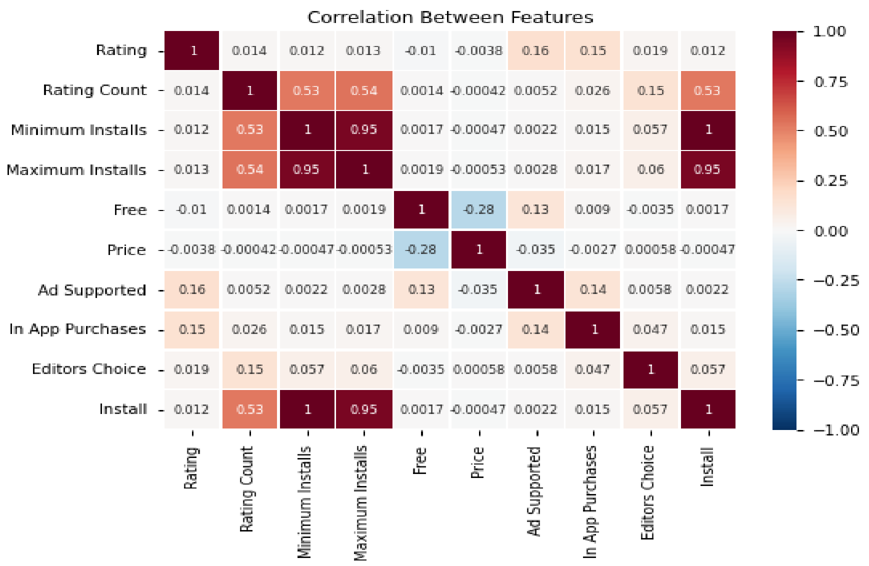


Figure 11: Histogram of numeric columns

As the columns are not normally distributed, we could perform a normal transformation on these datasets, but by doing so the columns original data would be manipulated and become meaningless, for example rating and price would include negative values. For this concern I am not performing normal distribution transformation on this dataset.

Heatmap & Pearson correlation coefficient matrix:

Correlation heatmaps are a type of plot that visualize the strength of relationships between numerical variables. Pearson correlation coefficient is a statistical measure of the linear relationship between two variables [2]. Which measures from -1 to 1. Correlation value near -1 indicates strong negative correlation and near 1 means strong positive correlation. Values near 0 indicates no correlation between the features.



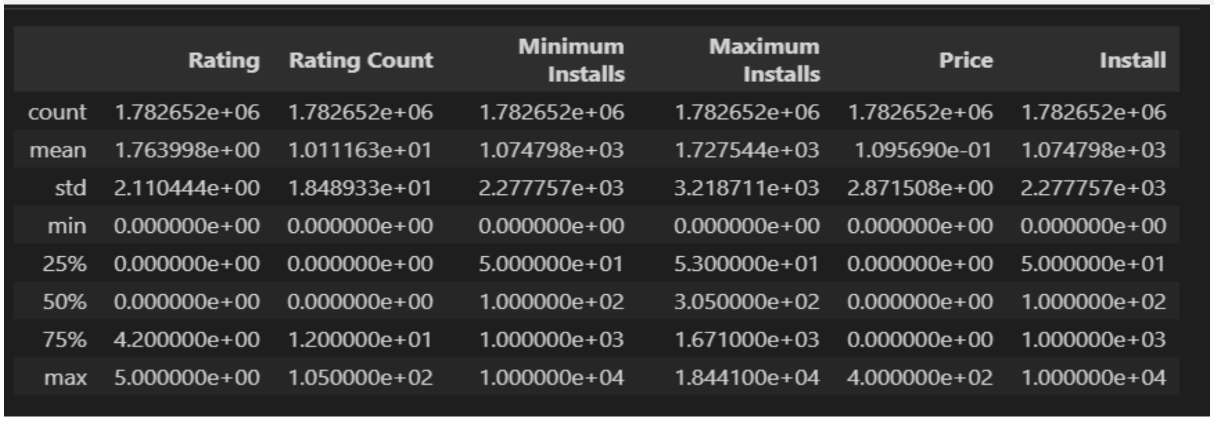
	Rating	Rating Count	Minimum Installs	Maximum Installs	Free	Price	Ad Supported	In App Purchases	Editors Choice	Install
Rating	1	0.014	0.012	0.013	-0.01	-0.0038	0.16	0.15	0.019	0.012
Rating Count	0.014	1	0.53	0.54	0.0014	-0.00042	0.0052	0.026	0.15	0.53
Minimum Installs	0.012	0.53	1	0.95	0.0017	-0.00047	0.0022	0.015	0.057	1
Maximum Installs	0.013	0.54	0.95	1	0.0019	-0.00053	0.0028	0.017	0.06	0.95
Free	-0.01	0.0014	0.0017	0.0019	1	-0.28	0.13	0.009	-0.0035	0.0017
Price	-0.0038	-0.00042	-0.00047	-0.00053	-0.28	1	-0.035	-0.0027	0.00058	-0.00047
Ad Supported	0.16	0.0052	0.0022	0.0028	0.13	-0.035	1	0.14	0.0058	0.0022
In App Purchases	0.15	0.026	0.015	0.017	0.009	-0.0027	0.14	1	0.047	0.015
Editors Choice	0.019	0.15	0.057	0.06	-0.0035	0.00058	0.0058	0.047	1	0.057
Install	0.012	0.53	1	0.95	0.0017	-0.00047	0.0022	0.015	0.057	1

Figure 12: Correlation matrix of features

From the correlation plot above (figure 12), we can notice that minimum installs and maximum installs both have strong positive correlation with installs. Whereas rating count has medium positive correlation with installs, minimum and maximum installs. Only price and free have low negative correlation. All other columns seem to have neutral to no correlation to each other.

Statistics:

Table 2: Statistics of the Numeric Columns



	Rating	Rating Count	Minimum Installs	Maximum Installs	Price	Install
count	1.782652e+06	1.782652e+06	1.782652e+06	1.782652e+06	1.782652e+06	1.782652e+06
mean	1.763998e+00	1.011163e+01	1.074798e+03	1.727544e+03	1.095690e-01	1.074798e+03
std	2.110444e+00	1.848933e+01	2.277757e+03	3.218711e+03	2.871508e+00	2.277757e+03
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	5.000000e+01	5.300000e+01	0.000000e+00	5.000000e+01
50%	0.000000e+00	0.000000e+00	1.000000e+02	3.050000e+02	0.000000e+00	1.000000e+02
75%	4.200000e+00	1.200000e+01	1.000000e+03	1.671000e+03	0.000000e+00	1.000000e+03
max	5.000000e+00	1.050000e+02	1.000000e+04	1.844100e+04	4.000000e+02	1.000000e+04

If we now look into the distribution (figure 13.1) of rating and price column we can see that the majority of ratings are around 0 or 4-5 and those are mostly free apps. Also, the majority of the app prices are around 0 and they are not ad supported.

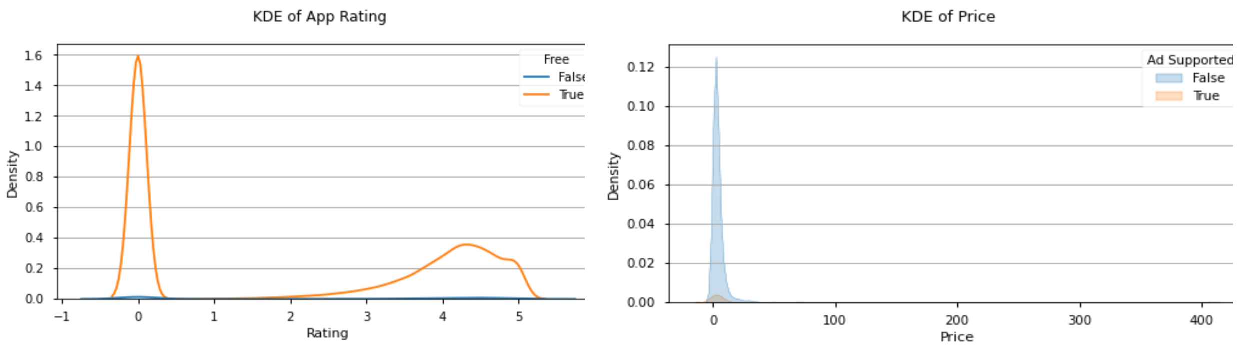


Figure 13.1: Kernel Density Estimation of app rating and price

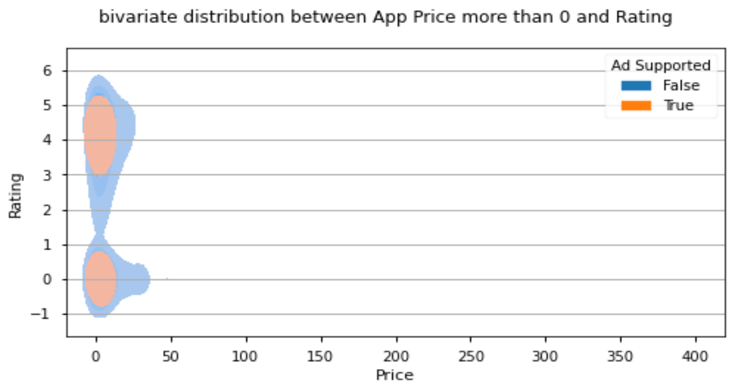


Figure 13.2: Bivariate Distribution of App price higher than 0

Now for better observation and faster code execution, let's look into the price distribution of app with price more than 0$ and their ratings. Here from the plot (figure 13.2) it is clearly noticeable that majority app prices aren't much higher than few cents and they have rating of 0 or in between 3 to 5.

Data Visualization:

a. Line-plot

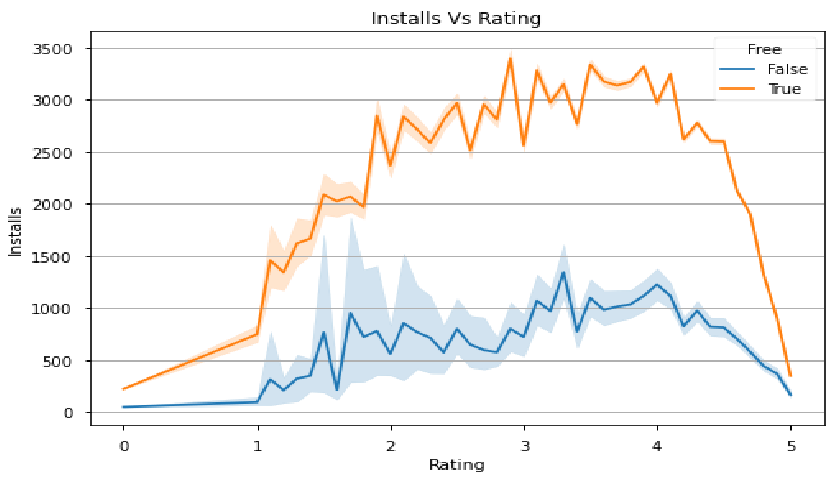


Figure 14: Line plot of App Installs vs Rating

Here in the figure 14 of app installs vs rating by free apps, we can observe that free apps are having higher installations and the rating is around 3 and 4.

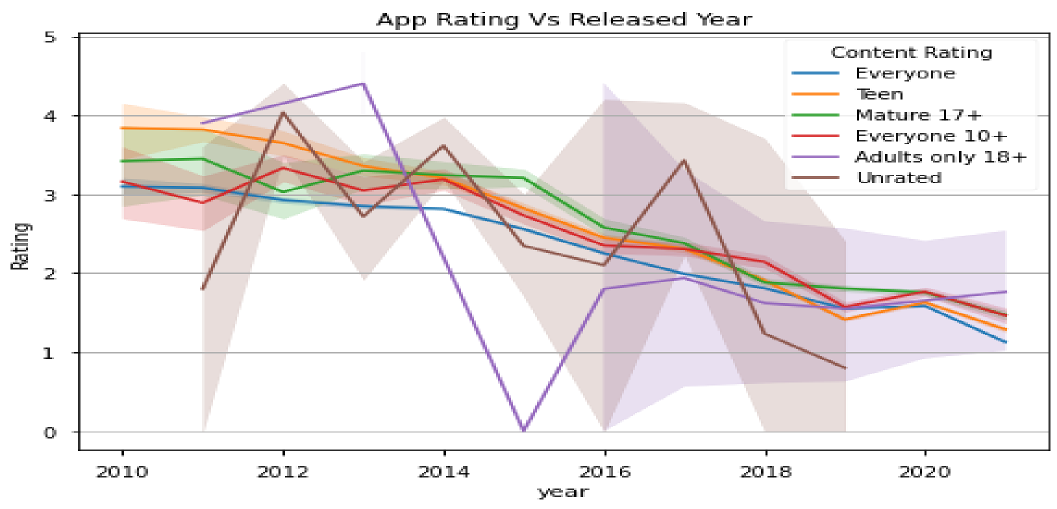


Figure 15: App rating vs released year by content rating

Here in figure 15, I have compared apps different content rating categories by rating and released year. We can observe that the rating kept decreasing over the years and adults only apps had a peak rating in 2013 but a huge drop in 2015. Also, many apps were kept unrated on 2012 and 2015.

b. Bar-plot:

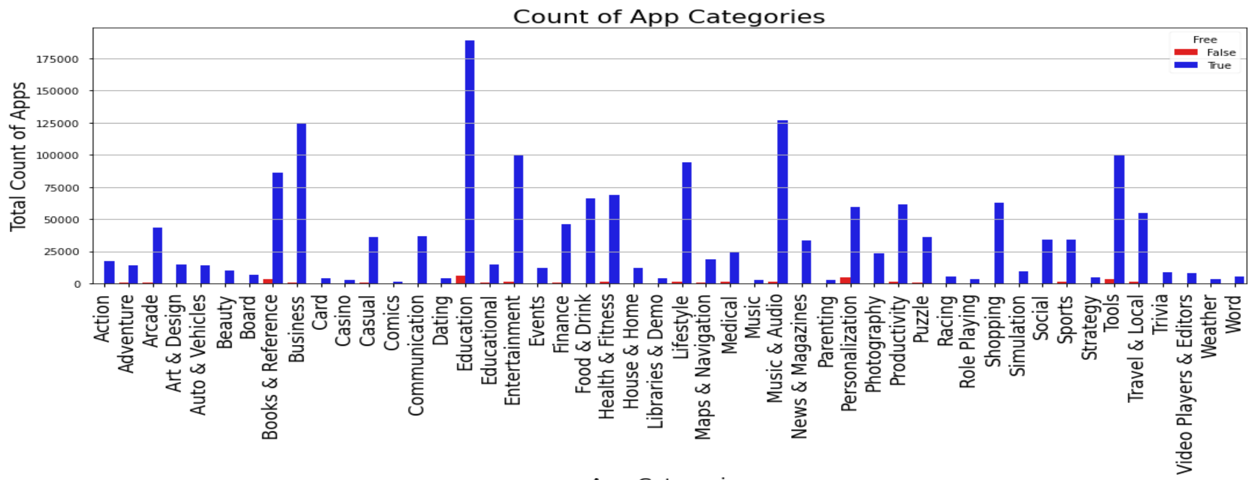


Figure 16: Count of app categories

As we can observe from the distribution of apps on distinct categories there are only a few apps which are not free and education app category has the highest number of apps in google play store. Business and music & audio are the second highest apps.

c. Count-plot

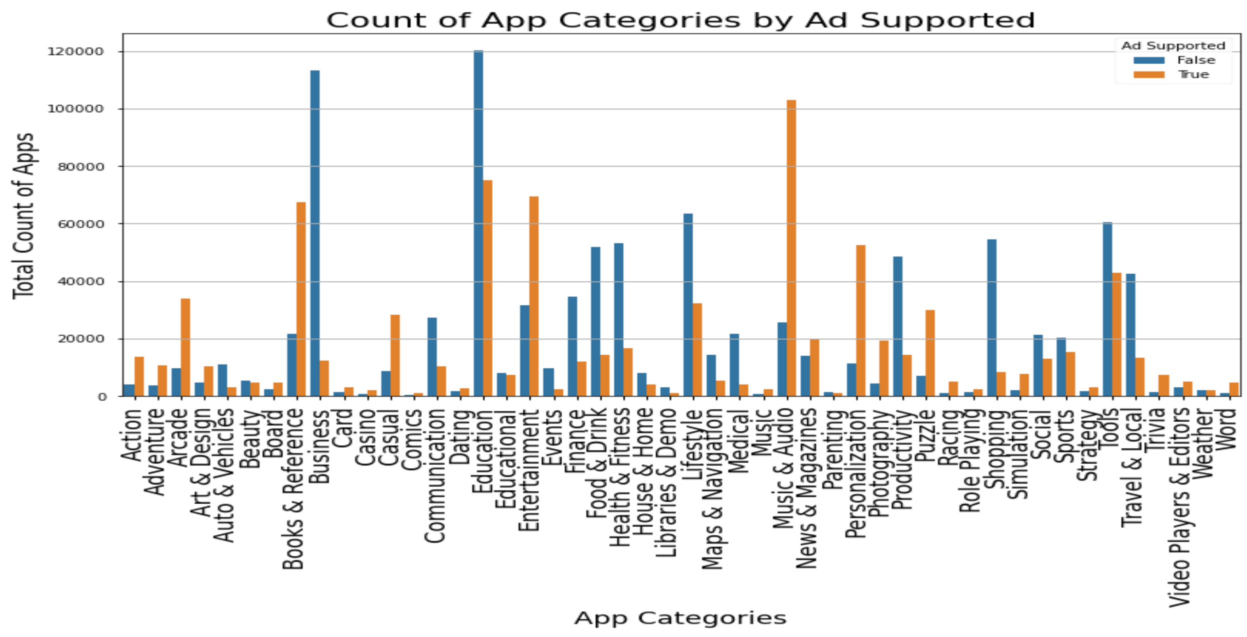


Figure 17: Count of app categories by ad supported

As we found out the top three app categories are from education, business, and music & audio, from figure 17 we can see that most of the business and education apps does not support adds but music & audio category have high ad supported app counts.

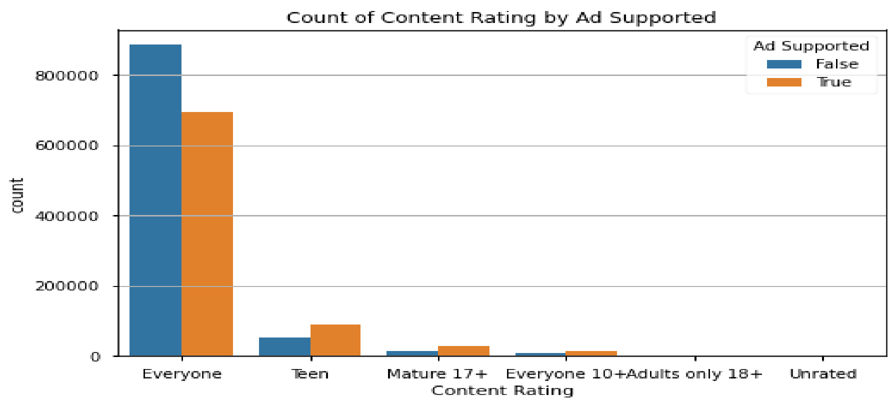


Figure 18: Count of app categories by add supported

In terms of content rating of apps, most of the apps are for everyone and among them majority supports ad. (figure 18)

d. Cat-plot

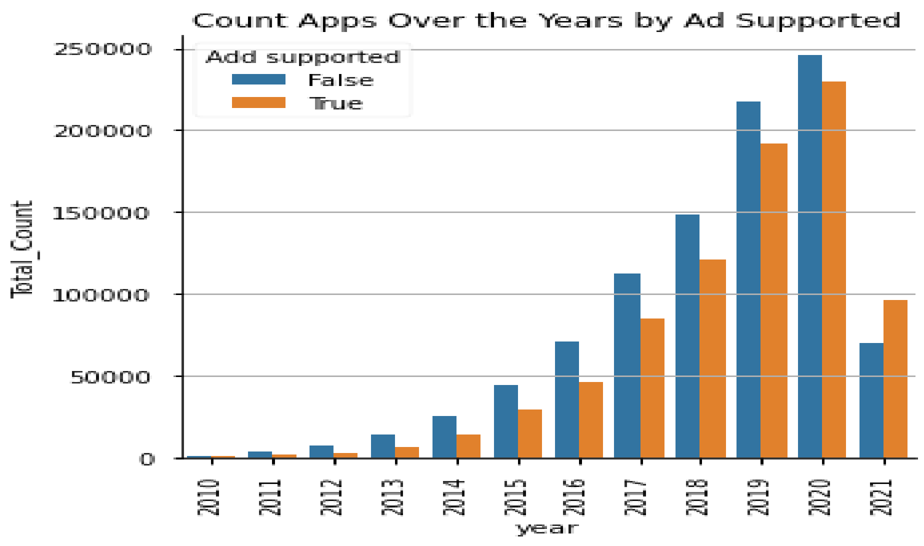


Figure 19: Count of apps over the years by ad supported

Here in figure 19, we can see that release of apps were increasing significantly over the years and in 2020 there were almost 250k apps and majority of them were not ad supported.

e. Pie-chart

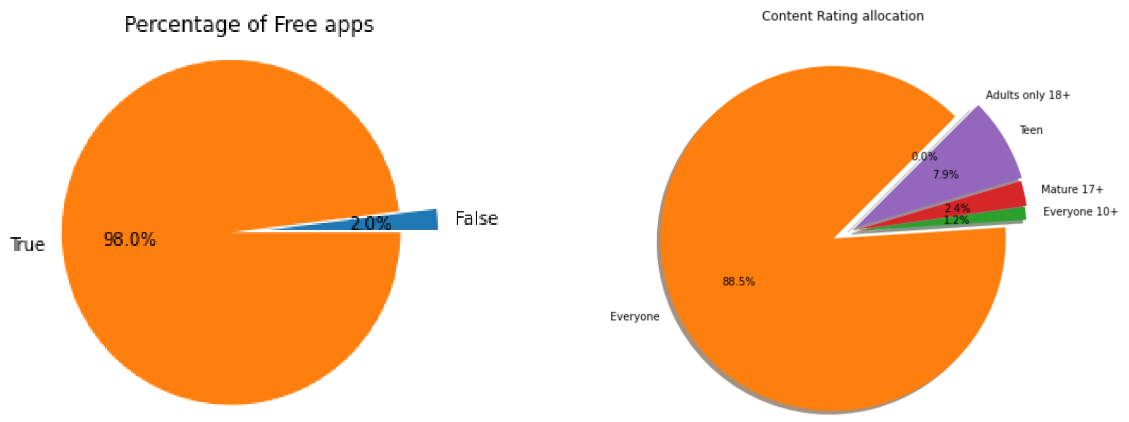


Figure 20: Pie chart of percentage of free apps and content rating

Here we can see only 2% of apps were not free and 88.5% of apps were for everyone.

f. Displot

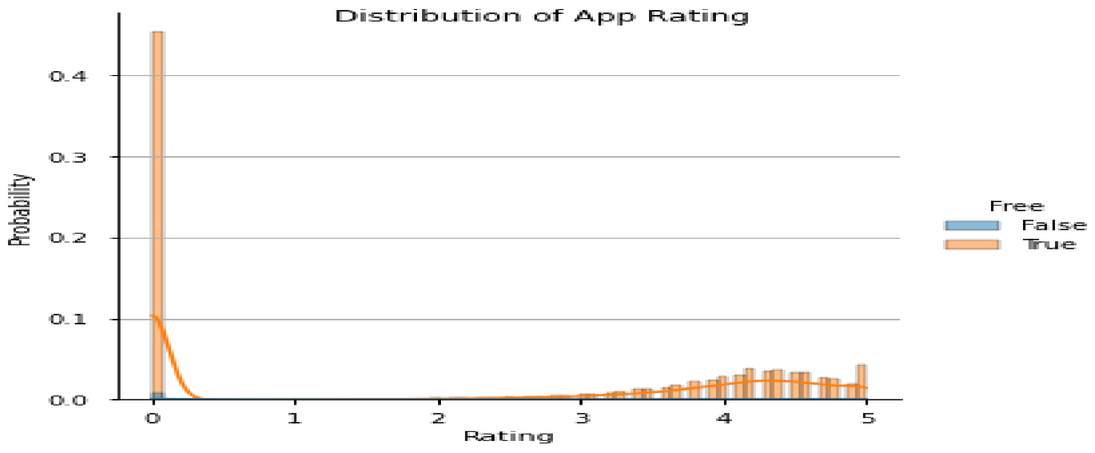


Figure 21: Distribution of app rating

From figure 21, we can say that majority of the apps had zero ratings and they were free. There is more than 40% probability that app rating is around zero. Other than that, there is less than 5% of probability that app rating is around 4-5.

g. Pair plot

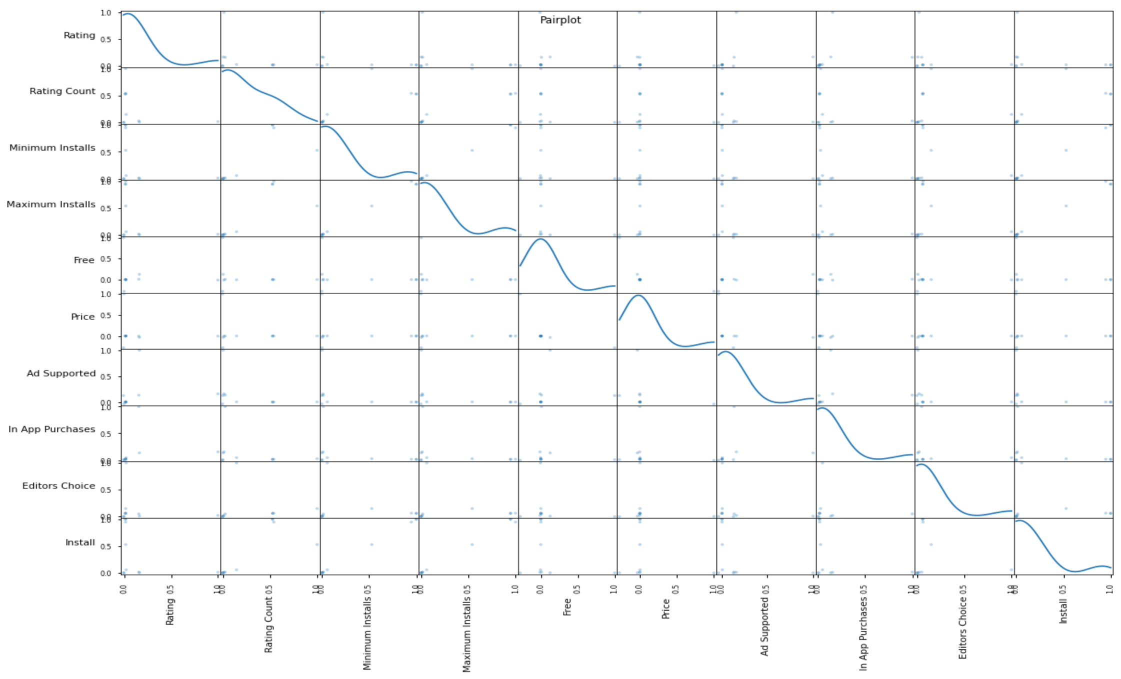


Figure 22.1: Pair plot of all the features

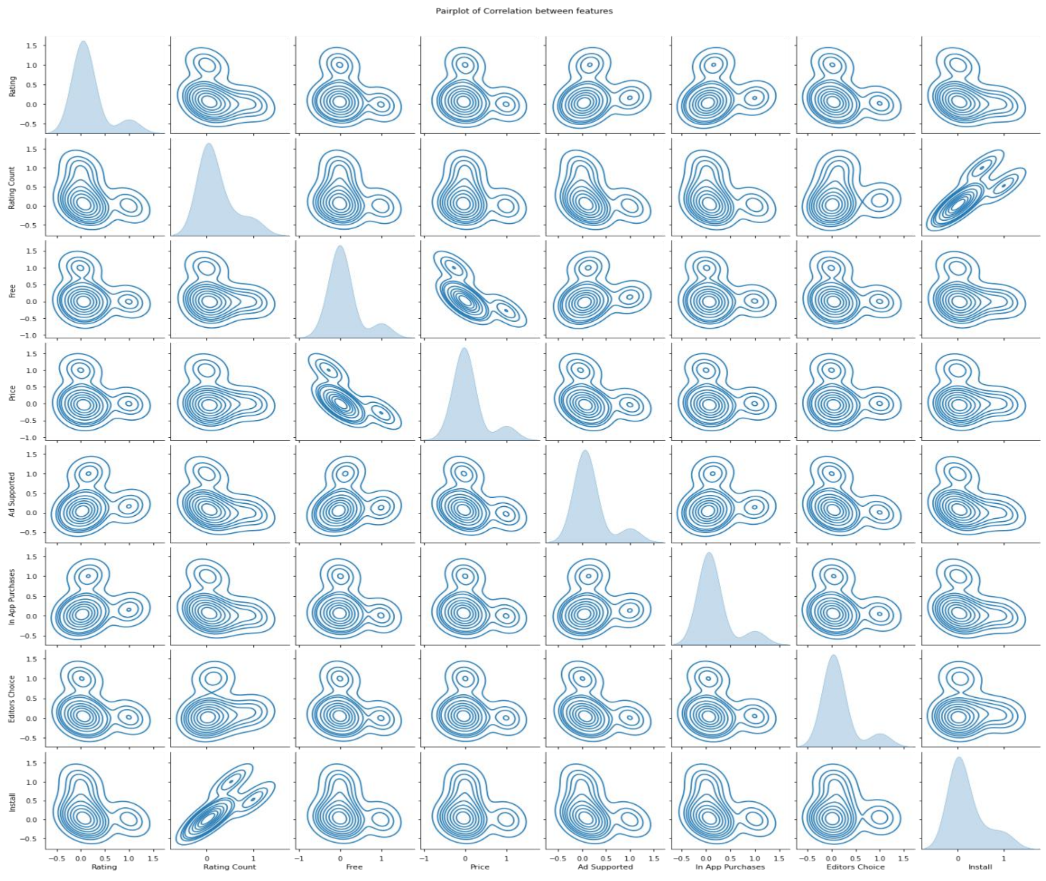


Figure 22.2: Pair plot of correlation between features excluding the installs

h. Hist-plot

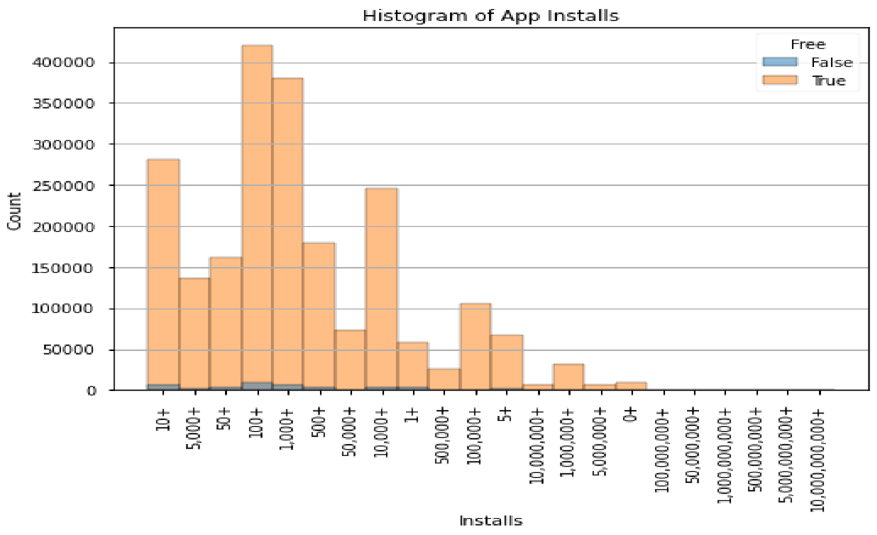


Figure 23: Histogram of app installs by free apps

Here in figure 23, we can see majority of apps have more than 100 to 1000 installs and they are mostly free apps.

i. Kernel density estimate

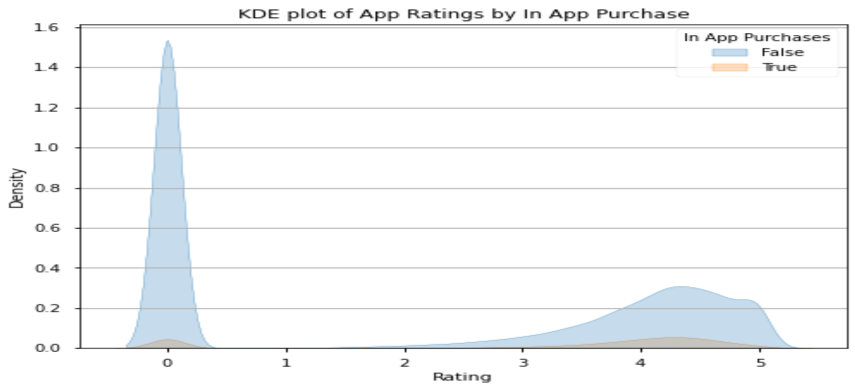


Figure 24: Kernel density estimation plot of ratings by in-app purchases

Majority of app ratings are near zero and they do not have in app purchase option. (Figure 24)

j. Scatter plot and regression line

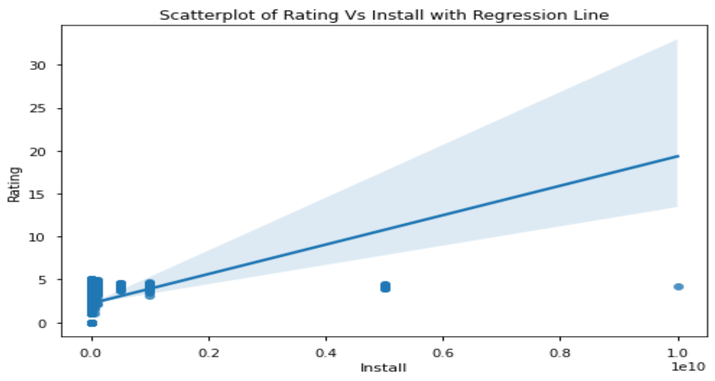


Figure 25: Scatter plot with linear regression

There is a linearly positive relationship between rating and installs. High ratings result into higher installations (figure 25).

k. Multivariate Box plot

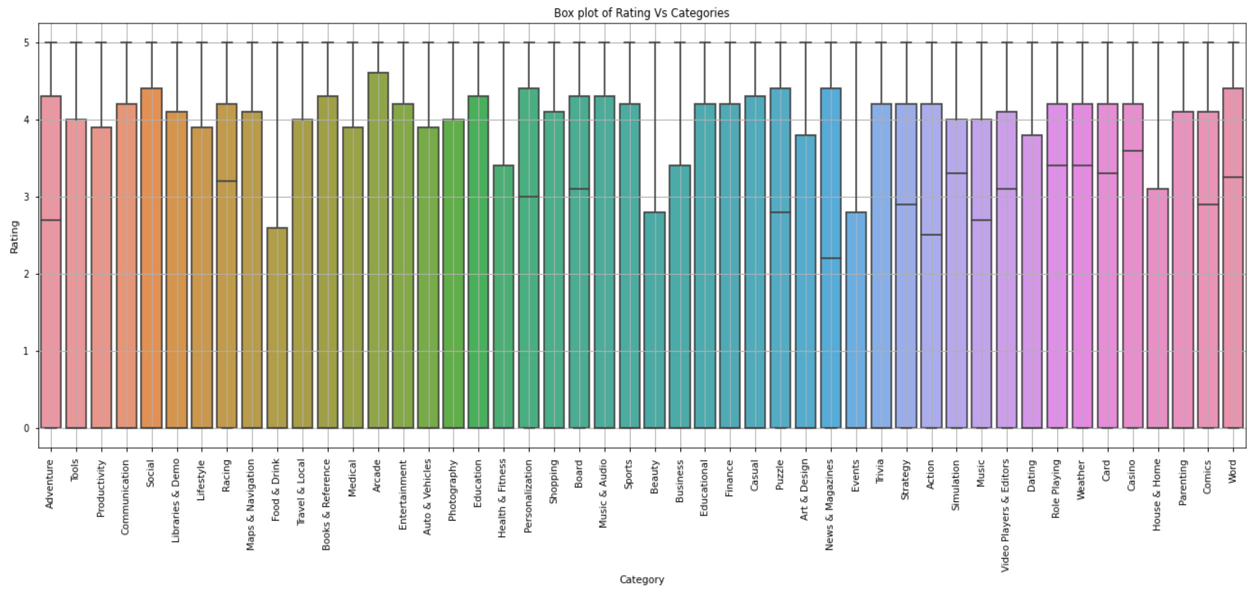


Figure 26: Multivariate box plot of rating by per app categories

From the boxplot of categories (figure 26) and ratings we can say that overall, the distribution of ratings in each category are not balanced and there are no outliers. Also, only news & magazines category has a balanced rating distribution.

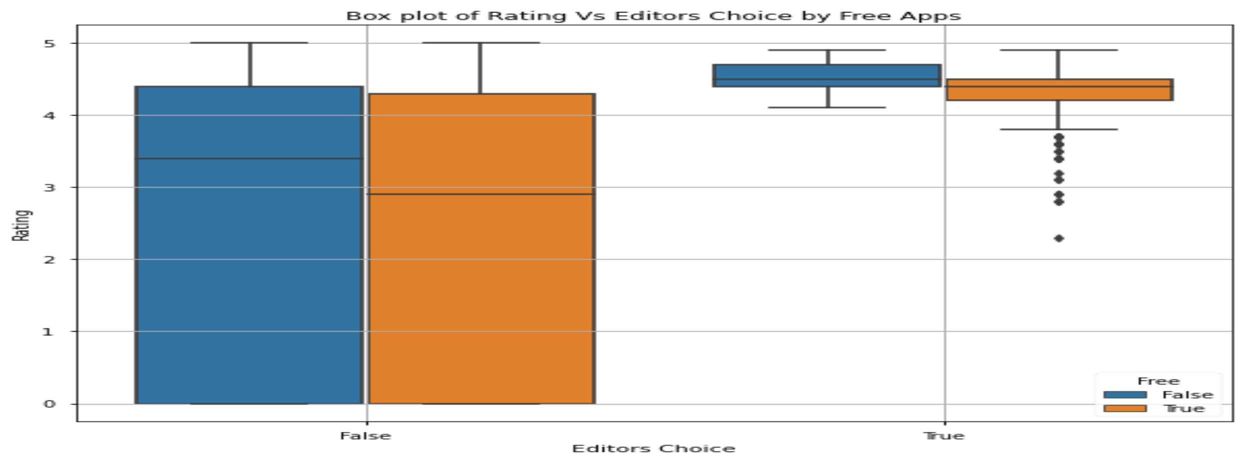


Figure 27: Multivariate box plot of rating by free apps

In this box plot of rating vs editors' choice, we can see some outliers in the free apps with editors' choice. Overall app with editor's choice has only high ratings. (Figure 27)

l. Violin plot

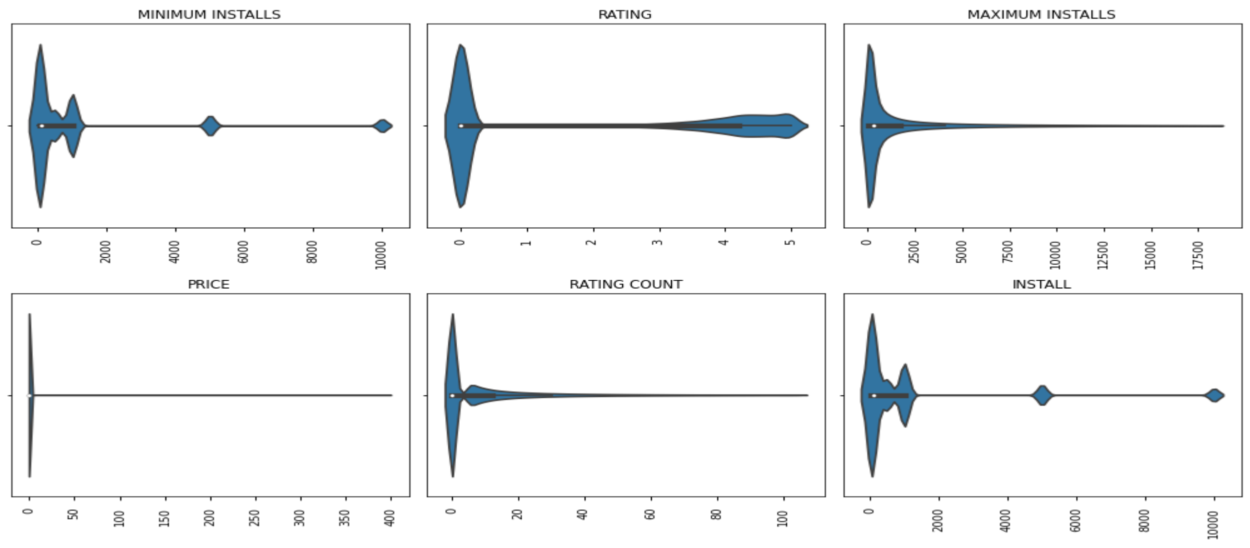


Figure 28: Violin plot of numeric columns

Here we can see the distribution of the six numeric columns in violin plot from. Maximum installs, price and rating counts seem to have high amounts of data in only around zero (figure 28).

m. Subplots:

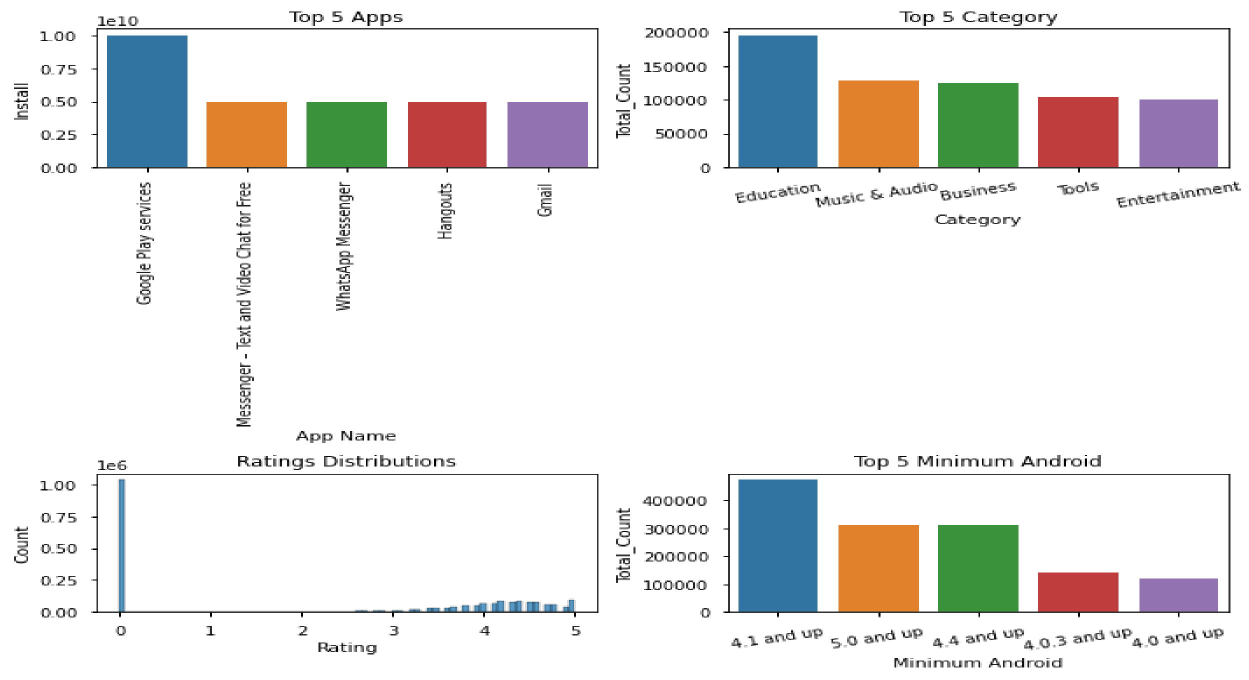


Figure 29: Top 5 apps, categories, minimum android, and rating distribution

Here I have focused on the top 5 of apps, categories, and minimum android. In terms of categories google play service, messenger and WhatsApp are the top 3 apps. Most apps are from education category and android 4.1 is the most apps minimum requirement.

Dashboard:

To visualize the google play store market in dashboard I am using python dash and plotly packages. As the original dataset has more than 2 million rows, implementing dashboard with this huge data lowers drown the performance of it. So, for dashboard implementation I have reduced the dataset by only taking installs count more than 100k and rating count more than 4500. Farther, I have dropped Scrapped time and Last updated column as it had high missing values. The final dataset for dash has 50775 rows and 22 columns.

I have used 5 tabs in the dashboard. In the first tab I am showing some statistics of the user preferred filtered apps.

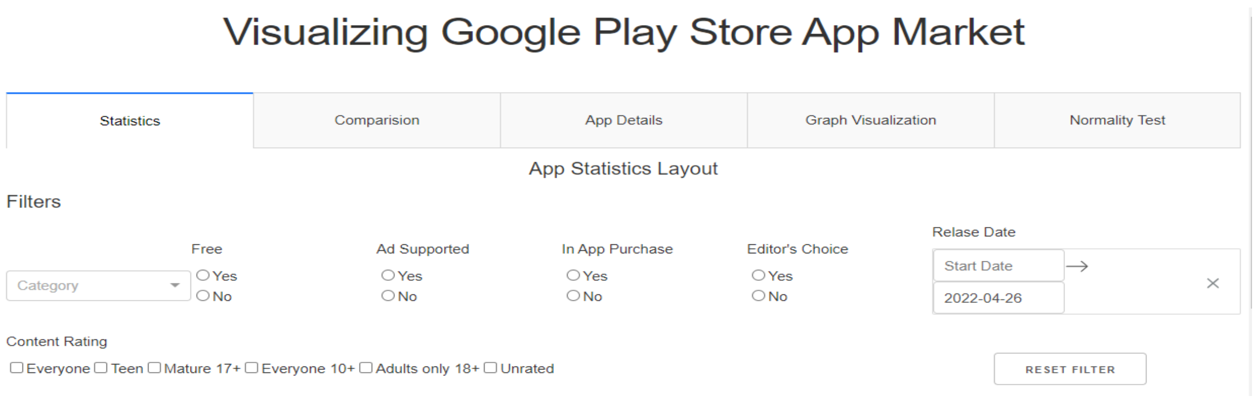


Figure 30.1: Dashboard tab 1 - Statistics

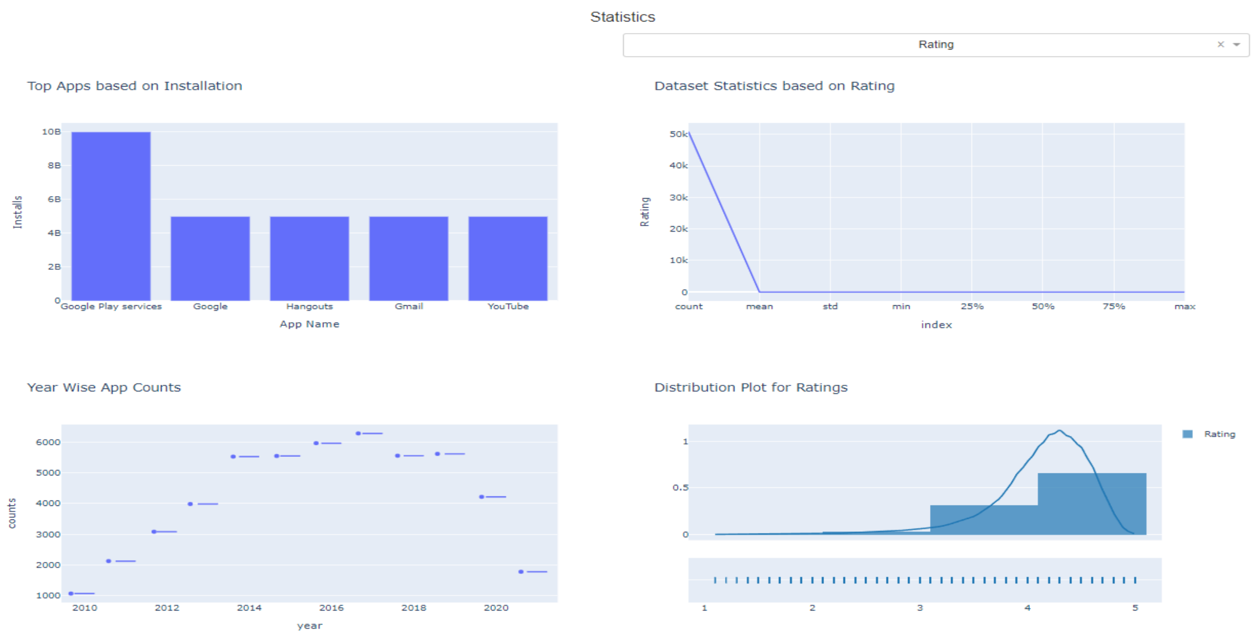


Figure 30.2: Outputs of tab 1

In this tab (figure 30.1) user can filter the apps based on the released date range (DataPickerRange), category (dropdown menu), free, ad supported, in-app purchase and editor's choice (radio button) and content rating (check box). There is a reset filter button to reset all the filters. In the output field (figure 30.2), there is a bar chart of top 5 apps and the count of their installs. Beside that there is a graph field to show the filtered apps statistics. In the app statistics output user can select the column from a dropdown menu of the numeric features (figure 30.3).

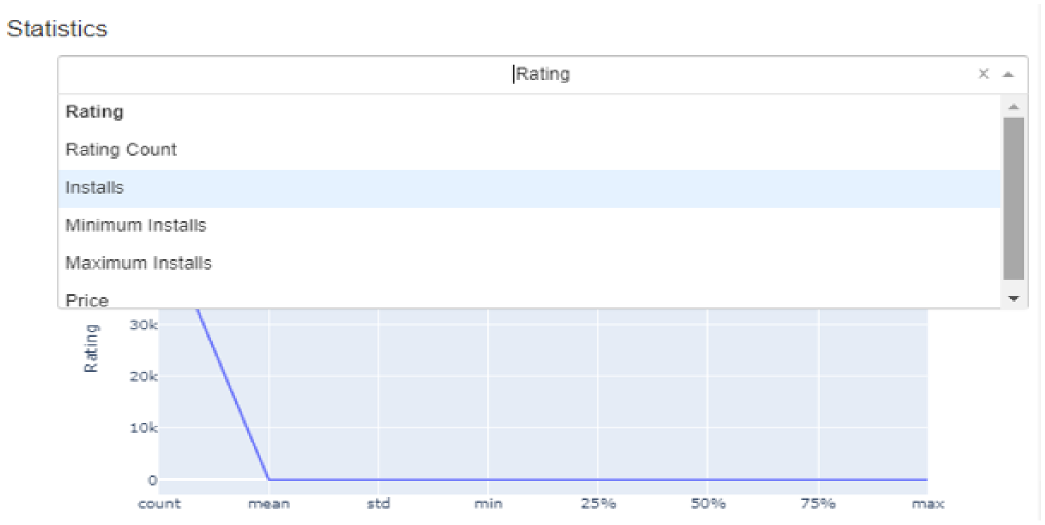


Figure 30.3: Statistics column selection

This tab also shows year wise count of apps and distribution of ratings of the user filtered apps.

In the second tab, I have compared two apps side by side. Here user can select two apps from dropdown menu and in the output section there are four graphs and a comparison table.

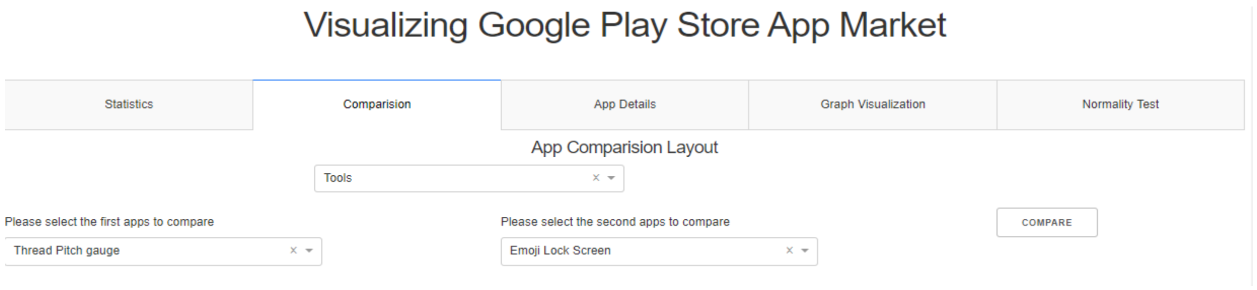
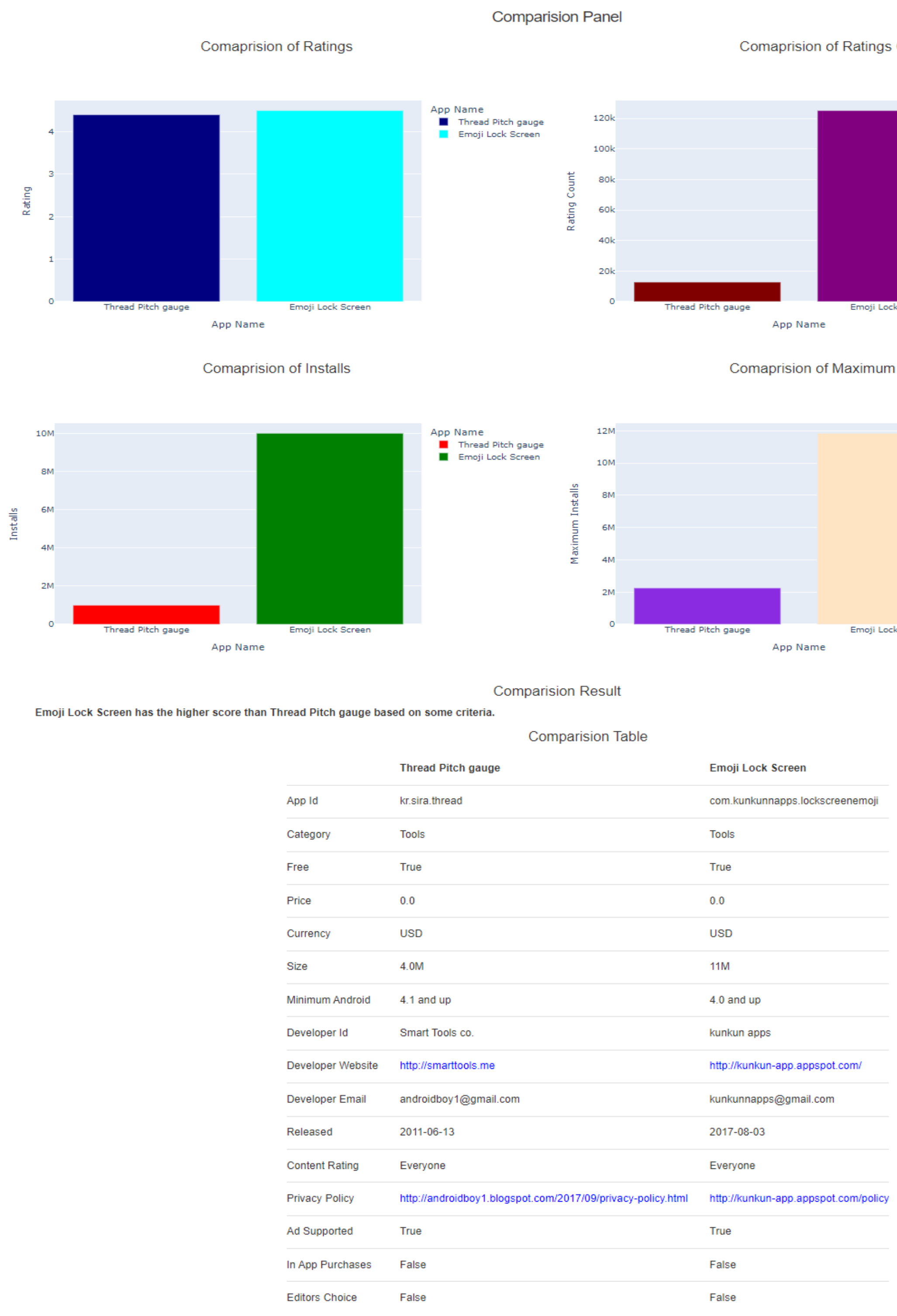


Figure 31.1: Dashboard tab 2 – app comparison

User can also category wise filter the apps in the two apps dropdown menu. (Figure 31.1)



Comparision Result

Emoji Lock Screen has the higher score than Thread Pitch gauge based on some criteria.

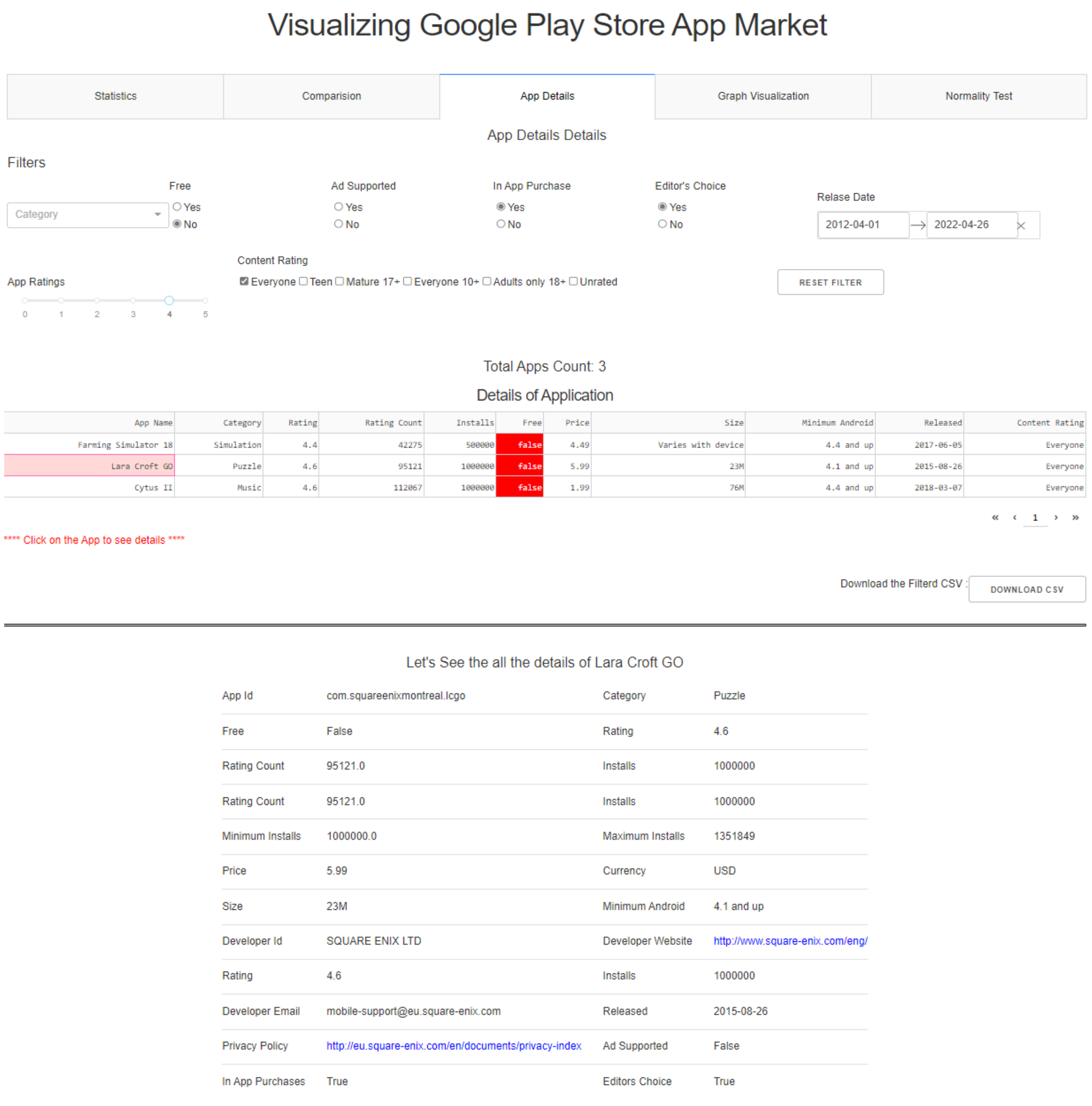
Comparision Table

	Thread Pitch gauge	**Emoji Lock Screen**
App Id	kr.sira.thread	com.kunkunnapps.lockscreenemoji
Category	Tools	Tools
Free	True	True
Price	0.0	0.0
Currency	USD	USD
Size	4.0M	11M
Minimum Android	4.1 and up	4.0 and up
Developer Id	Smart Tools co.	kunkun apps
Developer Website	http://smarttools.me	http://kunkun-app.appspot.com/
Developer Email	androidboy1@gmail.com	kunkunnapps@gmail.com
Released	2011-06-13	2017-08-03
Content Rating	Everyone	Everyone
Privacy Policy	http://androidboy1.blogspot.com/2017/09/privacy-policy.html	http://kunkun-app.appspot.com/policy
Ad Supported	True	True
In App Purchases	False	False
Editors Choice	False	False

Figure 31.2: Output of tab 2 app comparison

On the comparison table user can also directly visit the developer website and read the privacy policy of the app (figure 31.2).

In the third tab (figure 32), user can filter apps by category, free, ad supported, in-app purchase, editor's choice, release date, content rating and app rating. In the output there is a list of apps from that filtered apps and user can also download the filtered data in a CSV file.


Visualizing Google Play Store App Market

App Details Details

Filters

Category

Free: Yes [✓ No]

Ad Supported: Yes No

In App Purchase: [✓ Yes] No

Editor's Choice: [✓ Yes] No

Relase Date: 2012-04-01 → 2022-04-26

App Ratings: 0 1 2 3 [✓ 4] 5

Content Rating: [✓ Everyone] Teen Mature 17+ Everyone 10+ Adults only 18+ Unrated

Total Apps Count: 3

Details of Application

App Name	Category	Rating	Rating Count	Installs	Free	Price	Size	Minimum Android	Released	Content Rating
Farming Simulator 18	Simulation	4.4	42275	500000	false	4.49	Varies with device	4.4 and up	2017-06-05	Everyone
Lara Croft GO	Puzzle	4.6	95121	1000000	false	5.99	23M	4.1 and up	2015-08-26	Everyone
Cytus II	Music	4.6	112067	1000000	false	1.99	76M	4.4 and up	2018-03-07	Everyone

**** Click on the App to see details ****

Download the Filterd CSV :

Let's See the all the details of Lara Croft GO

App Id	com.squareenixmontreal.lcgo	Category	Puzzle
Free	False	Rating	4.6
Rating Count	95121.0	Installs	1000000
Rating Count	95121.0	Installs	1000000
Minimum Installs	1000000.0	Maximum Installs	1351849
Price	5.99	Currency	USD
Size	23M	Minimum Android	4.1 and up
Developer Id	SQUARE ENIX LTD	Developer Website	http://www.square-enix.com/eng/
Rating	4.6	Installs	1000000
Developer Email	mobile-support@eu.square-enix.com	Released	2015-08-26
Privacy Policy	http://eu.square-enix.com/en/documents/privacy-index	Ad Supported	False
In App Purchases	True	Editors Choice	True

Figure 32: Dashboard tab 3 – App details

Also, by clicking on the table row, user can see that specific app details in a table format bellow. Here also user can directly access the links of developer website and privacy policy.

In the next tab (figure 33.1), I have visualized the dataset using several types of graphs. User can select the type of graphs from the drop-down menu. There is line, bar, cat, count, pie chart, displot, histogram, heatmap, scatter, box, and violin plot options.

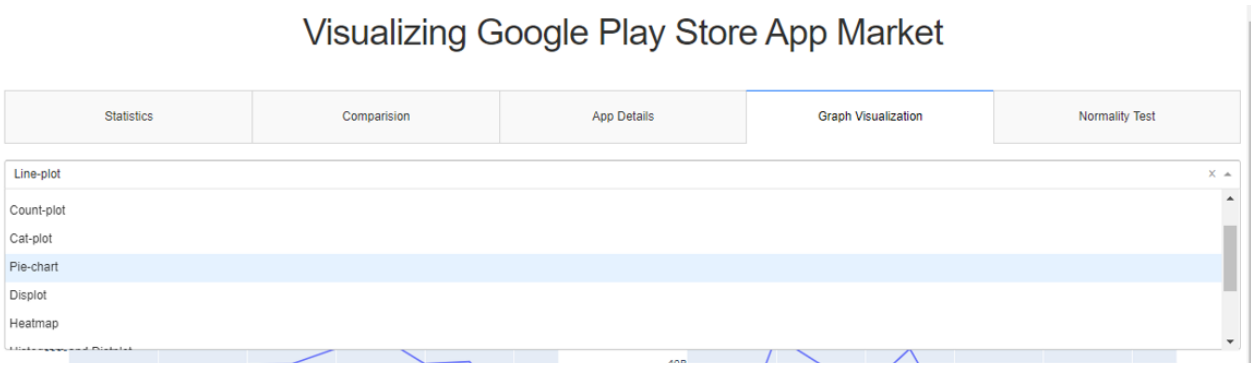


Figure 33.1: Tab 4 – Graph Visualization

Line Plot:

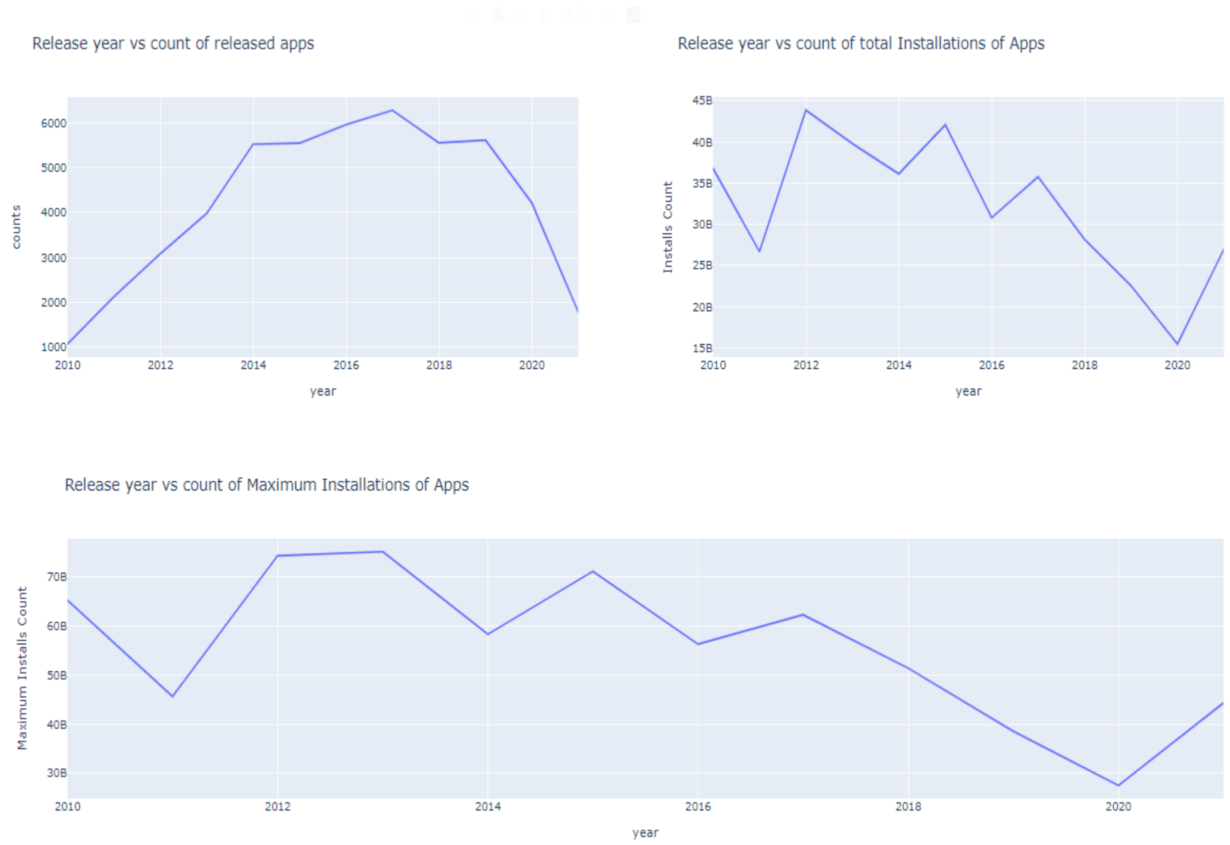


Figure 33.2: line plot outputs

These line plots indicates that year 2017 had the highest popular app releases but highest popular app installation was on year 2012. However, in 2020 there was a drop of maximum install of popular apps.

Bar Plot:

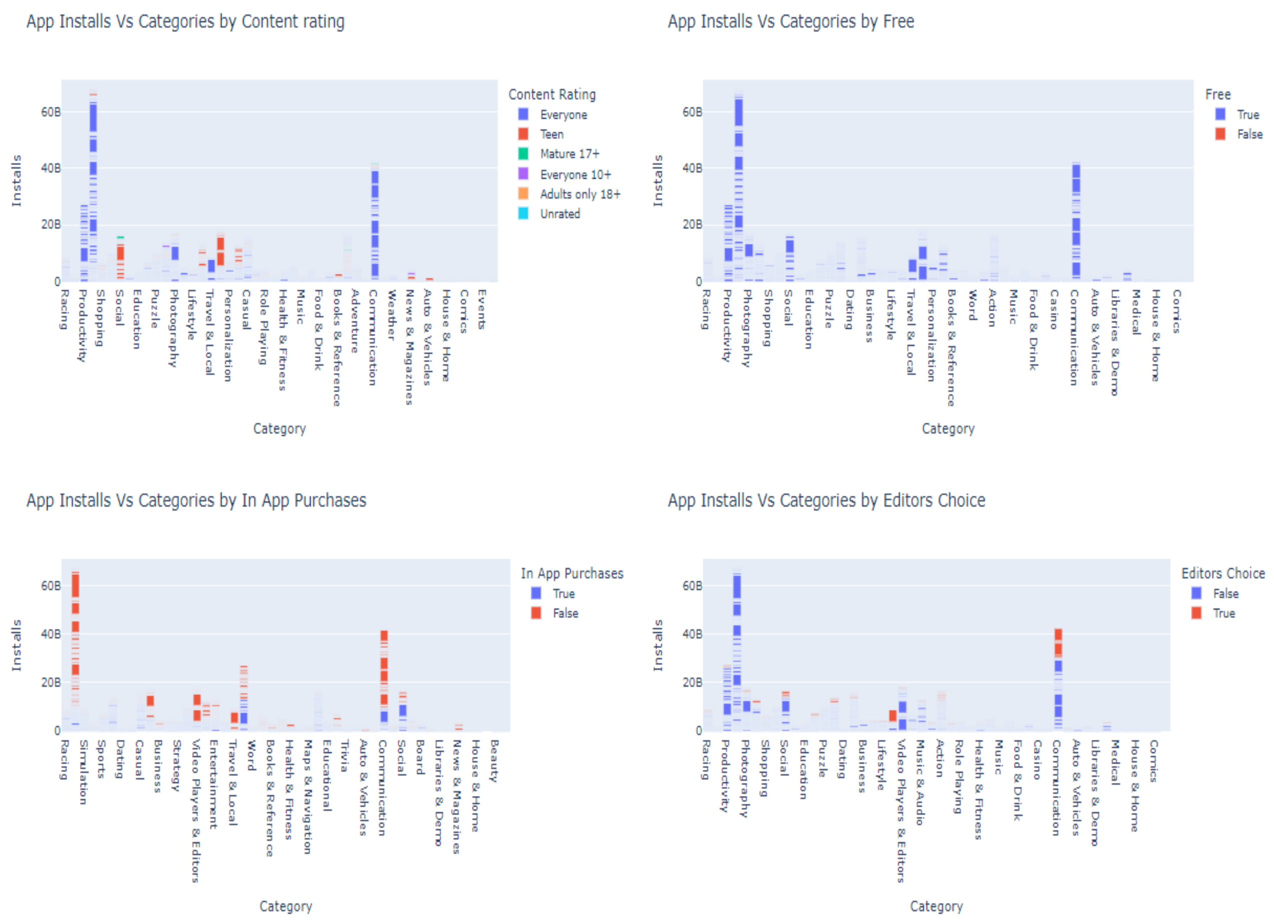


Figure 33.3: Bar plot outputs

Count Plot:

In the count plot section (figure 33.4), I have shown the count of minimum android version, categories, app releases over the years and content rating. Most of the popular apps requires minimum android version 4.1 and up. Tools category has the highest number of apps and most of the apps are from content rating of ‘everyone.’

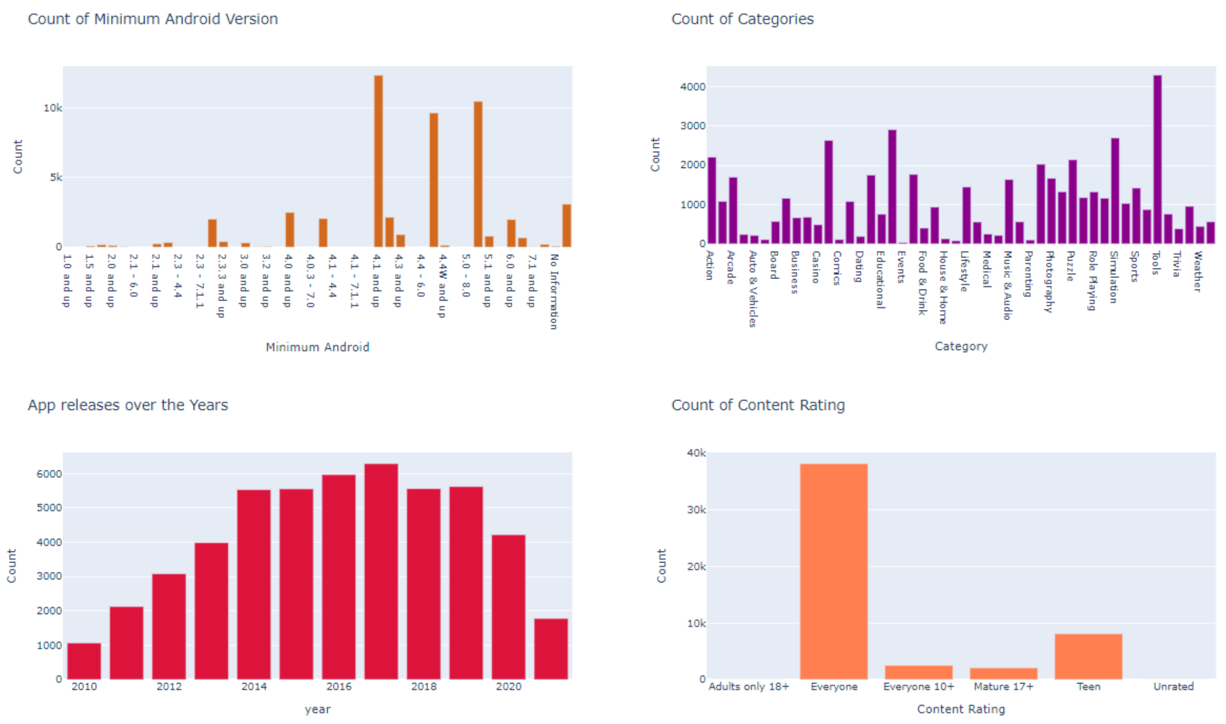


Figure 33.4: Count plot outputs

Catplot:

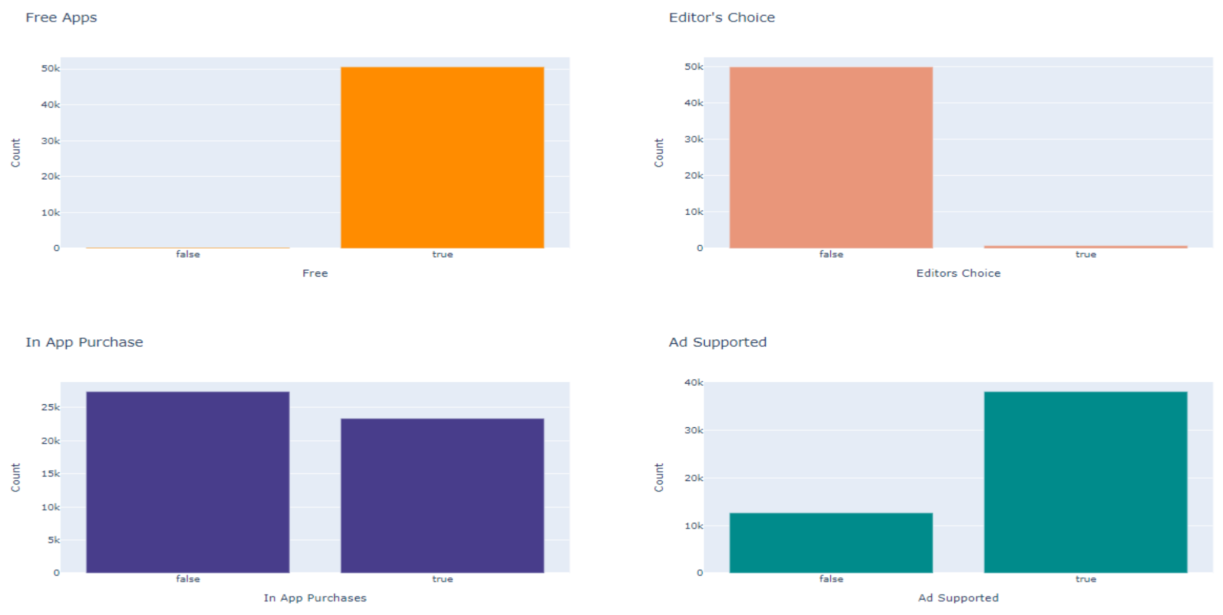


Figure 33.5: Cat plot outputs

In the catplot, I have basically showed the count of four Boolean type of columns as plotly package doesn't have any in built catplot like seaborn. From these graphs (figure 33.5), we can say most popular apps are free in play store, most of them are ad supported, most of them are not from editors' choice and most apps does not have in-app purchase option.

Pie Chart:

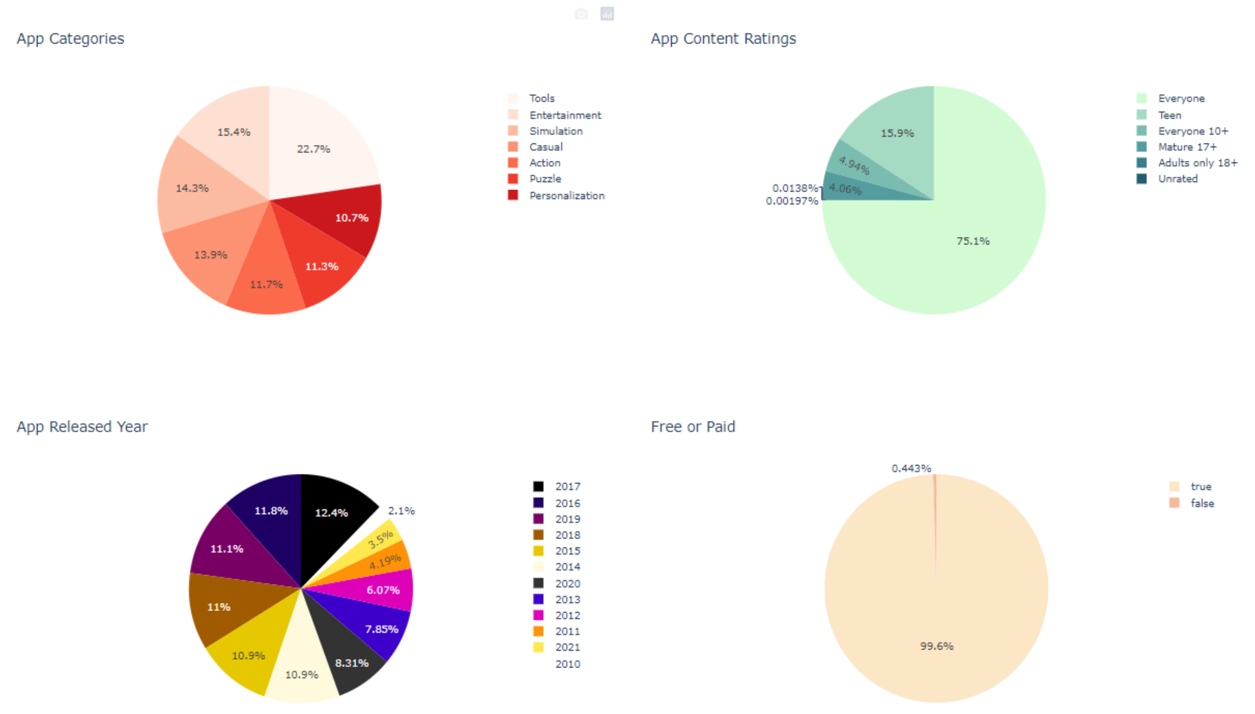


Figure 33.6: Pie chart outputs

From these pie charts, we can see there are 22.7% of popular apps are from tools category, 75% of popular apps are for everyone, 12.4% of popular apps were released in 2017. Only 0.4% of app are not free in the play store.

Displot:

In this plot section (figure 33.7), I have visualized the distribution of popular app prices, installs, rating and rating counts over the years.

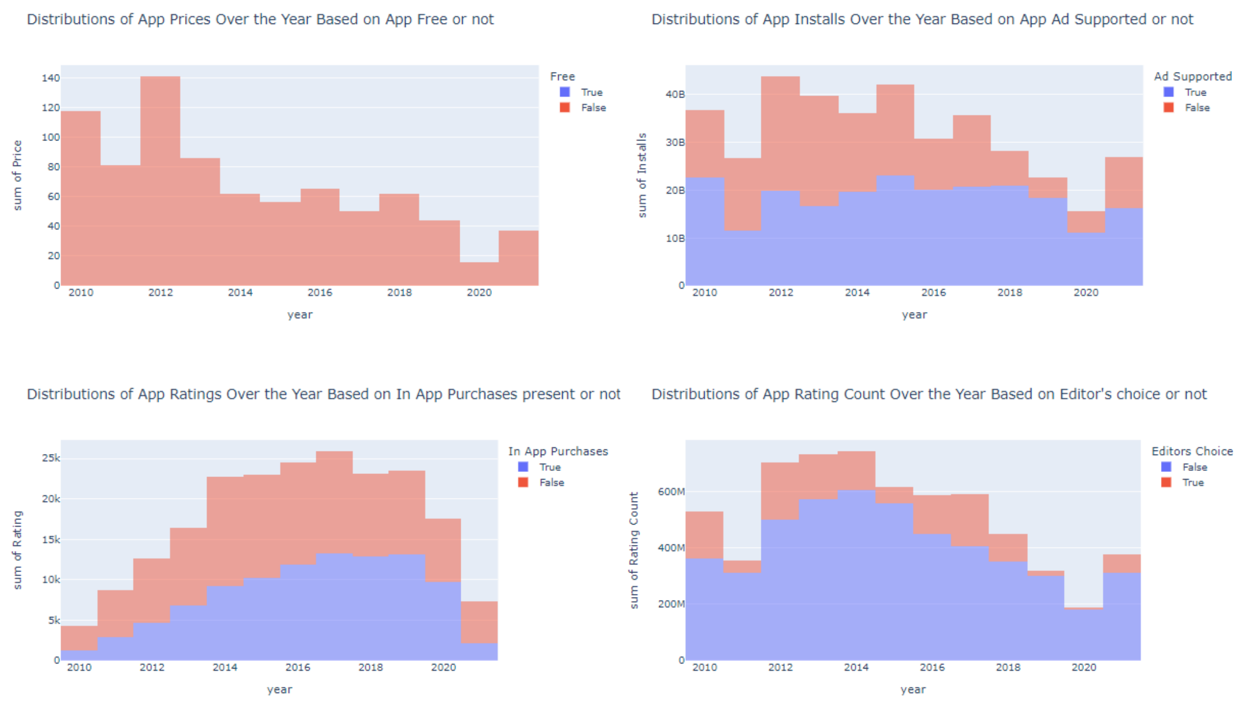


Figure 33.7: Distribution plot outputs

Heatmap:

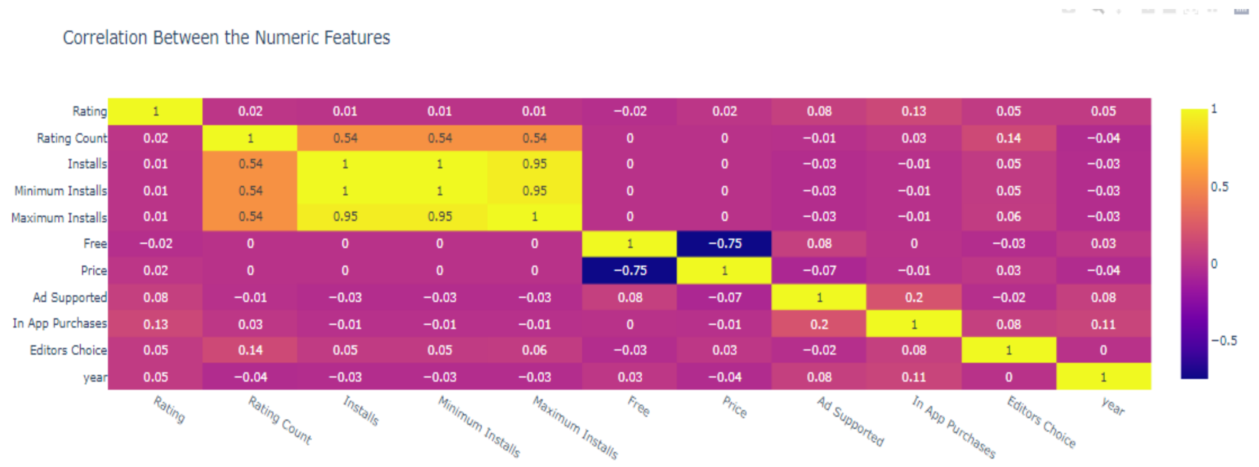


Figure 33.8: Heatmap output

Here is a person correlation matrix plotted as an heatmap.

Histogram and Distplot:

From these distributions we can see that the majority of the popular apps have 4 rating and app releases increased gradually till 2017 from 2010 but after that there is a decrease in the number of popular apps.

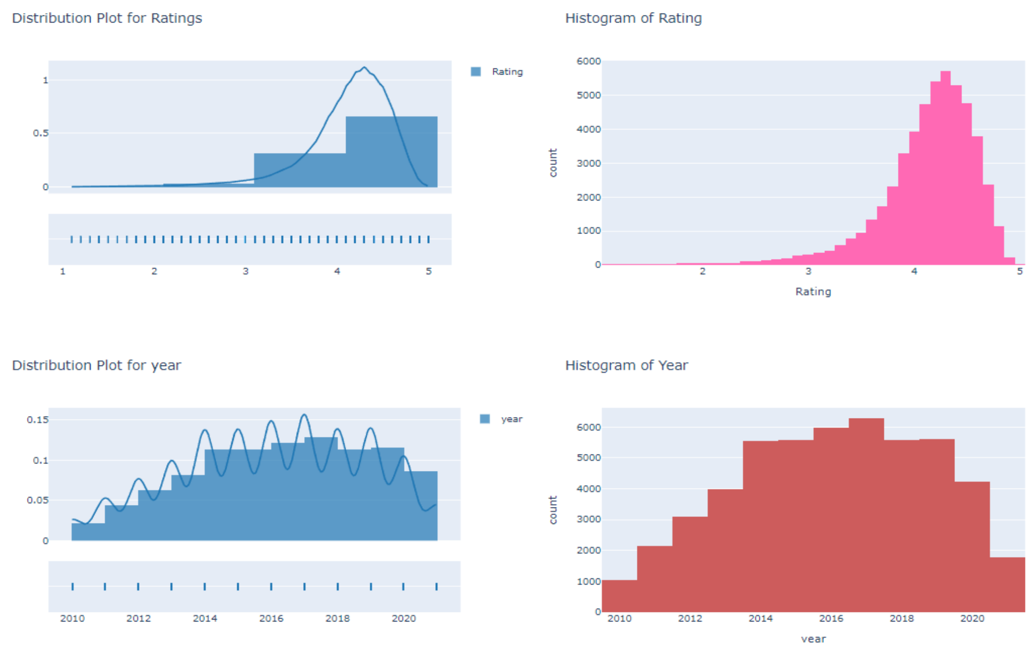


Figure 33.9: Histogram and Dist plot outputs

Scatter Plot:

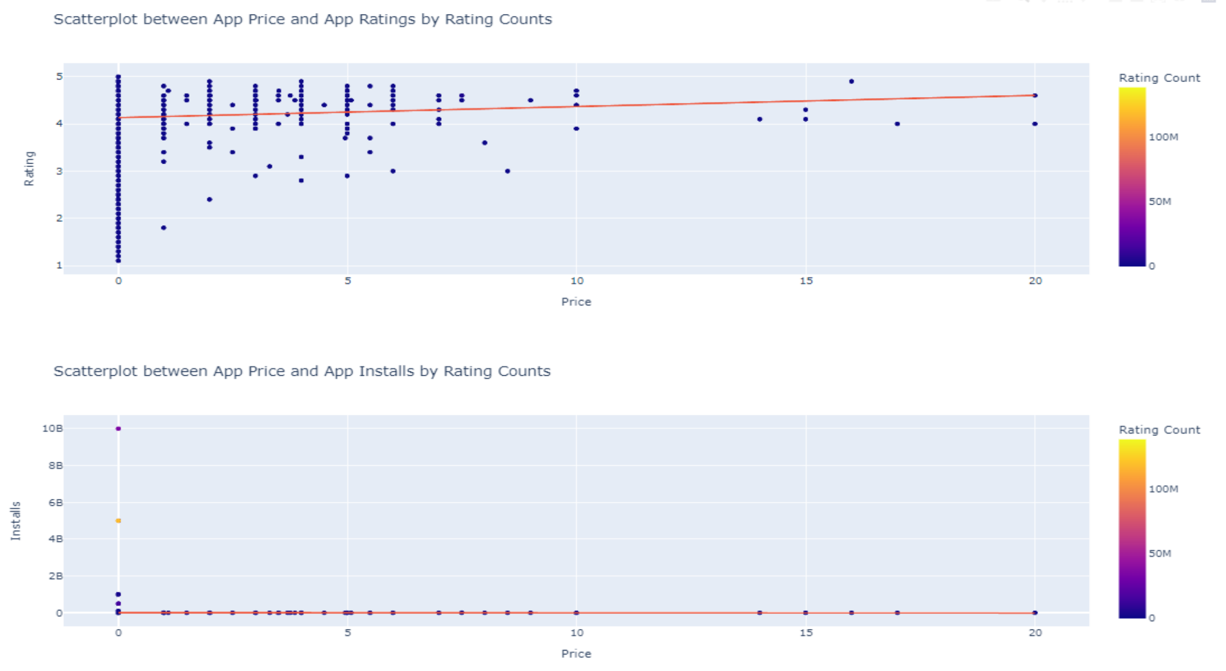


Figure 33.10: Scatter plot outputs

These scatterplots indicated that price and rating have a positive linear relationship but installs and price have a constant relationship.

Multivariate Box Plot:

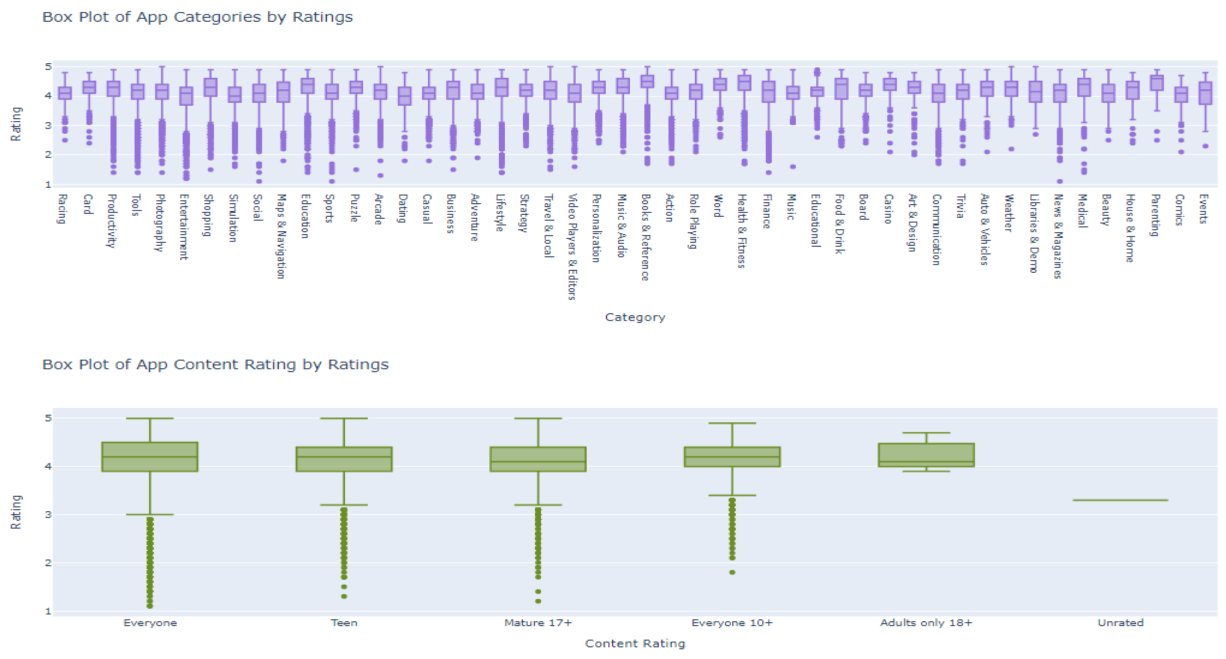


Figure 33.11: Box plot outputs

Violin Plot:

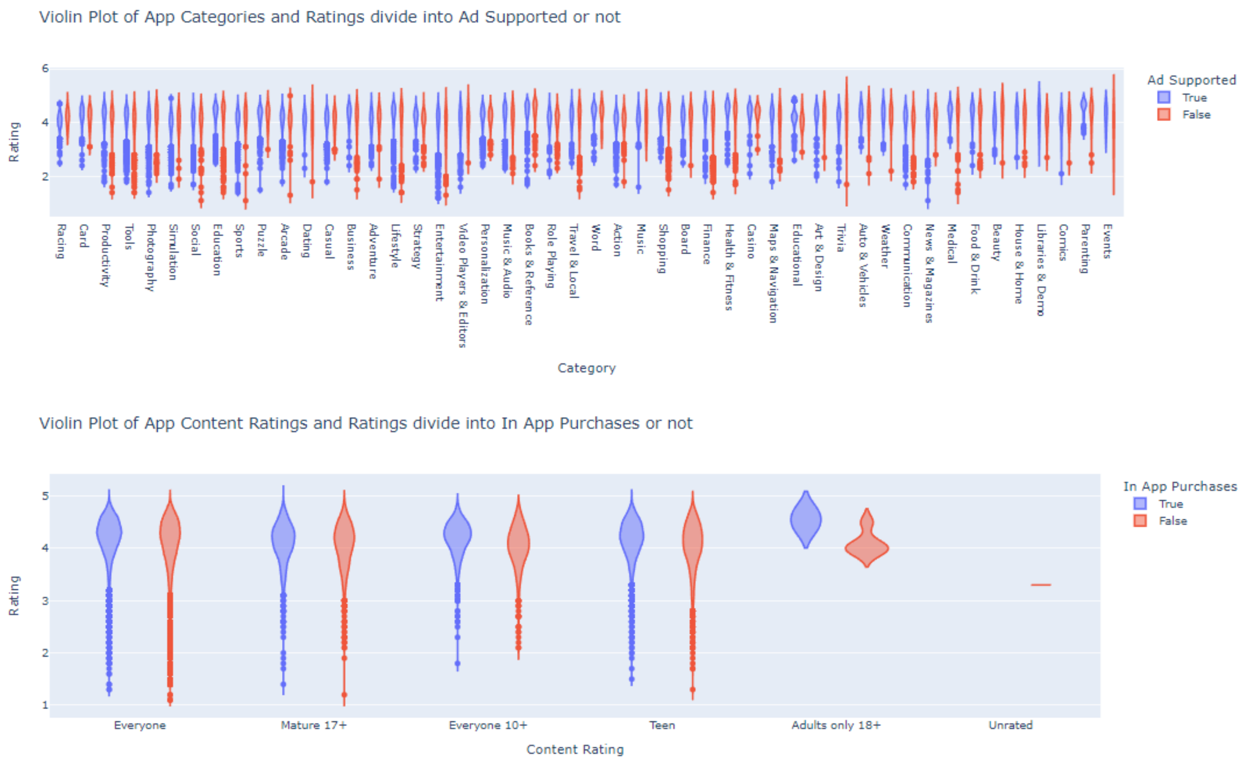
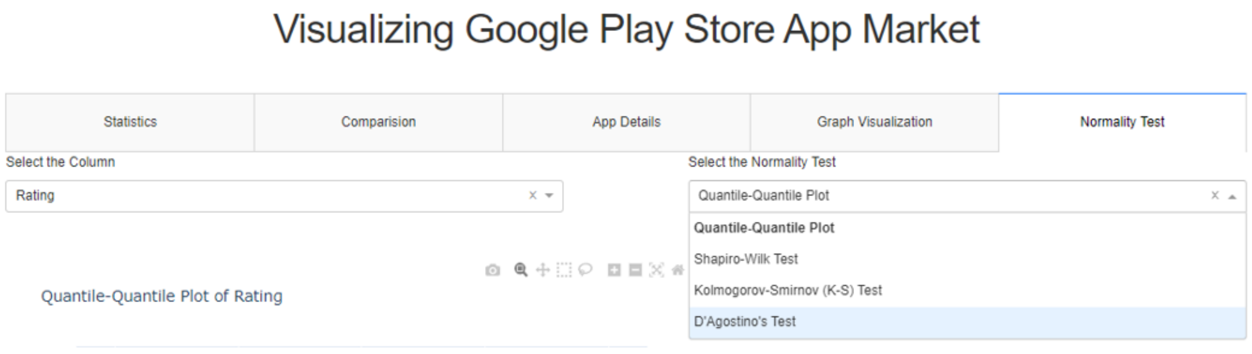


Figure 33.12: Violin plot outputs

Both these box plots (figure 33.11) and violin plots (figure 33.12) are indicating there are many outliers in the distinct categories and content rating. Overall, the distributions are not normal.

In the last tab on the dashboard, I have performed a normality test of the numeric columns. User can select the column as well as the type of test they want to perform. There is QQ plot to visually check the normality of the column (figure 34.1) and three statistical tests Shapiro walk, K-S test and D'Agostino test.



Visualizing Google Play Store App Market

Select the Column: Rating

Select the Normality Test: Quantile-Quantile Plot

Quantile-Quantile Plot
Shapiro-Wilk Test
Kolmogorov-Smirnov (K-S) Test
D'Agostino's Test
Quantile-Quantile Plot of Rating

Figure 42: Dashboard Tab 5 – Normality Test

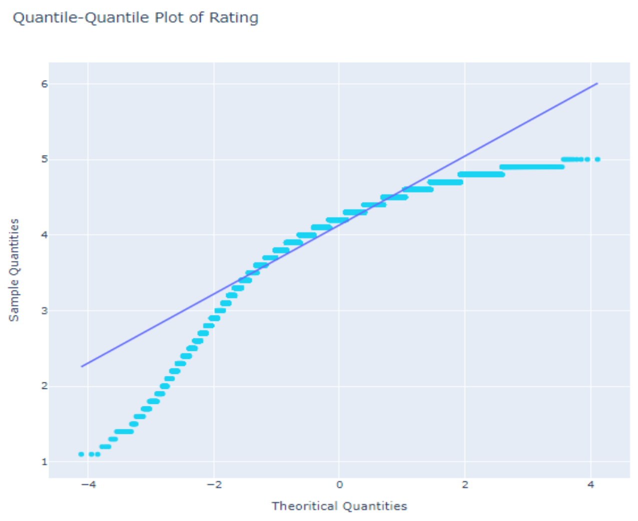
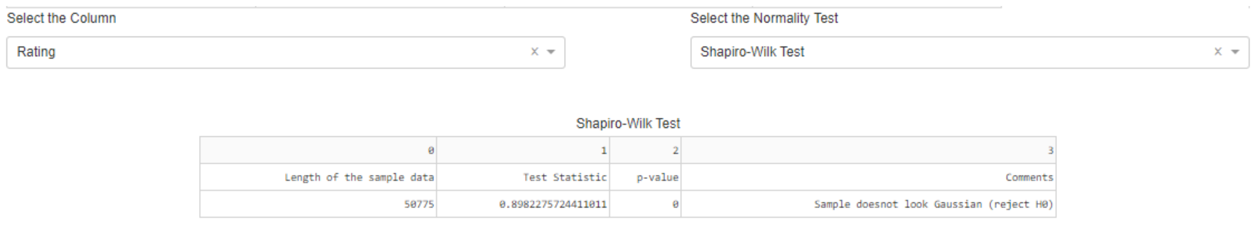


Figure 34.1: Output of normality test tab

The output of the statistical tests is shown in a tabular format (figure 34.2).



Select the Column: Rating

Select the Normality Test: Shapiro-Wilk Test

Shapiro-Wilk Test

0	1	2	3
Length of the sample data	Test Statistic	p-value	Comments
50775	0.8982275724411011	0	Sample doesnot look Gaussian (reject H0)

Figure 34.2: Output of normality test tab

Recommendations:

From this analysis on Google Play Store apps by visualizing we achieved a brief overview of the play store market. We can conclude that this markets main profit comes from user usages, ads and in app purchases rather than the price of the apps as majority of the apps are free or costs just few cents. We can also notice that the market received an increase of users gradually from 2010 to 2017 but a huge drop after 2019. Overall, in an analytical sense, the dataset is very imbalanced. To perform any prediction or forecasting this dataset needs extensive manipulation by statistically transforming and removing outliers.

In the dashboard, I have designed the web app in 5 tabs so that users can gain information for our datasets. From the first tab users can see few statistics of the overall dataset and, they can filter the dataset and see the statistics. Another tab gives the list of apps in the dataset and the details of a single app. You can also check that the dataset is gaussian or not. Our app can compare two apps and visualizing the comparison which can be an extremely useful tool for android users.

The website is very functional. Users can filter the dataset as they want. I have added a lot of filter options in the statistics tab as well as app details tab. Users can also reset the filter if they do not like the filtering. They can also download the filtered dataset as CSV file, with that user can further analyze on their desired dataset. Users can also see the details of the singles app by clicking the table row. In every tab there are multiple options to change the graphs and data interactively. Upon getting few reviews from users its recommended that the web app is a bit slower than desired. Other than that, they found it highly informative and user friendly.

References:

[1] https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps

[2] https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186#:~:text=Principal%20component%20analysis%2C%20or%20PCA,more%20easily%20visualized%20and%20analyzed.

[3] https://vitalflux.com/correlation-heatmap-with-seaborn-pandas/

Link to the web app: Dash (dashapp-s4afy3ytoq-rj.a.run.app)

Link to the GitHub repository: https://github.com/NusratNawshin/Analyzing-Play-Store-Apps

Appendix:

Source code of analysis using Seaborn:

```
#!/usr/bin/env python
# coding: utf-8

#%%

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
# import os
import seaborn as sns
# from scipy import stats
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from statsmodels.graphics.gofplots import qqplot
import plotly.express as px
# from scipy.stats.stats import pearsonr
from pandas.plotting import scatter_matrix
from matplotlib import axis, rcParams
# from sklearn.neighbors import KernelDensity
# from numpy import array, linspace
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
plt.style.use('seaborn-notebook')
# ### Dataset Description

#%%


play_store_app_df=pd.read_csv("data/Google-Playstore.csv")


#%%


print("Shape of the dataset: ",play_store_app_df.shape)


#%%


print("1st 5 rows of the dataset:\n",play_store_app_df.head(5))


# Dataset Description:
# The dataset is from kaggle(https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps).  This
dataset contain approximately 2.3 million google app store data.
# It contains 24 columns. In which 19 columns are categorical and 5 columns are numerical.
#
# Numerical columns list:
# - Rating
# - Rating Count
# - Minimum Installs
# - Maximum Installs
# - Price
#
# Categorical Columns List:
# - Object
#     - App Name
#     - App Id
#     - Category
#     - Installs
#     - Currency
#     - Size
#     - Minimum Android
#     - Developer Id
#     - Developer Website
#     - Developer Email
#     - Released
#     - Last Updated
#     - Content Rating
#     - Privacy Policy
#     - Scraped Time
# - Bool
#     - Free
#     - Ad Supported
#     - In App Purchases
#     - Editors Choice

#%%


print("Dataset Columns and Data types:")
print(play_store_app_df.info())


#%%

print("Dataset Column Statistics:")
print(play_store_app_df.describe())


# # Pre-Processing Dataset
# ## Statistics

#%%

print(f"Number of rows: {play_store_app_df.shape[0]}")
print(f"Number of columns: {play_store_app_df.shape[1]}")

print(f"Total Cells in the dataset is : {play_store_app_df.shape[0]*play_store_app_df.shape[1]}")
total_number_of_missing_data=play_store_app_df.isnull().sum().sum()
print(f"Total Number of missing data: {total_number_of_missing_data}")
percentage=(total_number_of_missing_data/(play_store_app_df.shape[0]*play_store_app_df.shape[1]))*100
print(f"Percentage of Missing values in play store dataset: {percentage:.2f}%")


#%%


column_wise_missing_values=play_store_app_df.isnull().sum()
print("Missing values in columns: \n",column_wise_missing_values)


print("From the upper section we can see that 'Developer Website' and 'Privacy Policy'\
 both has lot of missing value. If we delete this row then almost 1.1 million data row\
 will be missing. So our plan is to drop these columns from the dataset.")
#%%


Column_wise_missing_values_percentage=column_wise_missing_values*100/len(play_store_app_df)
print(f"Missing value percentages: \n{Column_wise_missing_values_percentage}")


#%%

play_store_app_df=play_store_app_df.drop(['Developer Website','Privacy Policy'],axis=1)
print("Dataset after dropping Developer Website and Privacy Policy:\n",play_store_app_df.head(5))
```

```
# Updated Dataset statistics

#%%

print(f"Number of rows: {play_store_app_df.shape[0]}")
print(f"Number of columns: {play_store_app_df.shape[1]}")
print(f"Total Cells in the dataset is : {play_store_app_df.shape[0]*play_store_app_df.shape[1]}")
total_number_of_missing_data=play_store_app_df.isnull().sum().sum()
print(f"Total Number of missing data: {total_number_of_missing_data}")
percentage=(total_number_of_missing_data/(play_store_app_df.shape[0]*play_store_app_df.shape[1]))*100
print(f"Percentage of Missing values in play store dataset: {percentage:.2f}%")

#%%


play_store_app_df.isnull().sum()


print("For removing missing values from the dataset, for the categorical data I\
 will remove the row from the dataset and for the numerical data I will replace\
 it with average value.")

#%%

# Imputing missing values
play_store_app_df['Rating'].fillna((play_store_app_df['Rating'].mean()), inplace=True)
play_store_app_df['Rating Count'].fillna((play_store_app_df['Rating Count'].mean()), inplace=True)
play_store_app_df['Minimum Installs'].fillna((play_store_app_df['Minimum Installs'].mean()), inplace=True)


#%%

# Drop all rows with missing values
play_store_app_df=play_store_app_df.dropna()


#%%

print("Final Dataset: \n",play_store_app_df.isnull().sum())

#%%

# Final dataset
print(f"Number of rows: {play_store_app_df.shape[0]}")
print(f"Number of columns: {play_store_app_df.shape[1]}")
print(f"Total Cells in the dataset is : {play_store_app_df.shape[0]*play_store_app_df.shape[1]}")
total_number_of_missing_data=play_store_app_df.isnull().sum().sum()
print(f"Total Number of missing data: {total_number_of_missing_data}")
percentage=(total_number_of_missing_data/(play_store_app_df.shape[0]*play_store_app_df.shape[1]))*100
print(f"Percentage of Missing values in play store dataset: {percentage:.2f}%")


# # Outlier detection and removal
#
# Normally outliers can be found in the numerical columns.
# In out dataset there is 5 numerical dataset which are
# - Rating,
# - Rating Count ,
# - Minimum Installs,
# - Maximum Installs,
# - Price.
#
# At First We will visualize the this 5 columns so that we can check that if there is any outliers.
#

#%%
```

```
# Rating
plt.figure()
sns.boxplot(play_store_app_df['Rating'])
plt.title("Boxplot of App Rating")
plt.show()

#%%

# Rating count
plt.figure()
sns.boxplot(play_store_app_df['Rating Count'])
plt.title("Boxplot of App Rating Count")
plt.show()

#%%

plt.figure()
sns.boxplot(play_store_app_df['Minimum Installs'])
plt.title("Boxplot of App Minimum Installs")
plt.show()

#%%
plt.figure()
sns.boxplot(play_store_app_df['Maximum Installs'])
plt.title("Boxplot of App Maximum Installs")
plt.show()
#%%

plt.figure()
sns.boxplot(play_store_app_df['Price'])
plt.title("Boxplot of App Price")
plt.show()

#%%

plt.figure()
sns.scatterplot(data=play_store_app_df, x='Rating', y='Rating Count')
plt.title("Rating Count Vs Rating")
plt.grid()
plt.show()

#%%

# Scatterplot
plt.figure()
sns.scatterplot(data=play_store_app_df, x='Maximum Installs', y='Minimum Installs')
plt.title("Minimum Installs Vs Maximum Installs")
plt.grid()
plt.show()


#%%


# print(play_store_app_df.shape)

# From Upper Graphs you can see that some of the numerical columns has outlier.
# Rating Count, Maximum Installs and Minimum Installs have some data which are
# very far from the most of the data. So For getting the better visualization we
# need to remove those data from the dataset and check the outliers again.

#%%


def outliers_detection(play_df,columns):
    Q1=play_df[columns].quantile(0.25)
    Q3=play_df[columns].quantile(0.75)
    IQR=Q3-Q1
    Lower_limit=Q1-1.5*IQR
    Upper_limit=Q3+1.5*IQR
    ls=play_df.index[(play_df[columns]<Lower_limit) | (play_df[columns]>Upper_limit)]
    return ls


#%%


def remove_outliers(play_df,detected_index):
    detected_index=sorted(set(detected_index))
    play_df=play_df.drop(detected_index)
    return play_df


#%%

outlier_index_list=[]
# outlier_index_list.extend(outliers_detection(play_store_app_df,'Minimum Installs'))
for column in ['Minimum Installs','Rating Count','Maximum Installs']:
    outlier_index_list.extend(outliers_detection(play_store_app_df,column))


#%%

print("Total Outliers: ",len(outlier_index_list))


#%%

# Remove outliers
play_store_app_df_removed_outlier=remove_outliers(play_store_app_df,outlier_index_list)
print("After removing outliers shape of the dataset:",play_store_app_df_removed_outlier.shape)


#%%



f,axes=plt.subplots(1,2)
sns.boxplot(play_store_app_df_removed_outlier['Minimum Installs'],ax=axes[1])
axes[0].set_title("Before removing outlier")
sns.boxplot(play_store_app_df['Minimum Installs'],ax=axes[0])
axes[1].set_title("After removing outlier")
plt.suptitle("Minimum Installs")
plt.tight_layout()
plt.show()

#%%



f,axes=plt.subplots(1,2)
sns.boxplot(play_store_app_df['Maximum Installs'],ax=axes[0])
axes[0].set_title("Before removing outlier")
sns.boxplot(play_store_app_df_removed_outlier['Maximum Installs'],ax=axes[1])
axes[1].set_title("After removing outlier")
plt.suptitle("Maximum Installs")
plt.tight_layout()
plt.show()

#%%

f,axes=plt.subplots(1,2)
sns.boxplot(play_store_app_df['Rating Count'],ax=axes[0])
axes[0].set_title("Before removing outlier")
sns.boxplot(play_store_app_df_removed_outlier['Rating Count'],ax=axes[1])
axes[1].set_title("After removing outlier")
plt.suptitle("Rating Count")
plt.tight_layout()
plt.show()



#%%

print("Shape of dataset after removing outliers: ",play_store_app_df_removed_outlier.shape)


# # Principal Component Analysis (PCA)


# We are using only those columns which are numerical for PCA .
# So total 5 features have used in PCA.
# Features are :Rating,  Rating Count,Minimum Installs,Maximum Installs,Price
#

#%%


numerical_columns=play_store_app_df[['Rating','Rating Count','Minimum Installs','Maximum
Installs','Price']]
Converted_dataframe=StandardScaler().fit_transform(numerical_columns)
# Converted_dataframe


#%%


Singular_values=np.matmul(numerical_columns[numerical_columns.columns[1:]].values.T,numerical_columns[nume
rical_columns.columns[1:]].values)
Condition_number=np.linalg.svd(Singular_values)
print(f'Singular Values:\n{Singular_values}')
print(f'Condition Number:\n{np.linalg.cond(numerical_columns[numerical_columns.columns[1:]].values)}')


#%%


originals=numerical_columns[numerical_columns.columns[:]]
plt.figure(figsize=(6,4))
sns.heatmap(originals.corr(),annot=True,linewidths=1)
plt.suptitle("Correlation Coefficient between features-original feature space")
plt.show()


#%%


pca=PCA(n_components='mle',svd_solver='full')
pca.fit(Converted_dataframe)
playstore_pca=pca.transform(Converted_dataframe)

print("Original Dimension:",Converted_dataframe.shape)
print("Transformed dimension:", playstore_pca.shape)
print("Explained variance ration:\n",pca.explained_variance_ratio_)
x=np.arange(1,len(np.cumsum(pca.explained_variance_ratio_))+1,1)
plt.plot(x,np.cumsum(pca.explained_variance_ratio_))
plt.xticks(x)
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Cumulative Explained Variance vs Number of Components")
plt.grid()
plt.show()
```

```python
#%%


print('Eventhough PCA already reduced the features from from 5 to 4,one more feature can be removed as
with 3 features we are getting almost 90%explained variance. ')
# Making new reduced feature space with 3 components
pcaf=PCA(n_components=3,svd_solver='full')
pcaf.fit(Converted_dataframe)
PlayStore_pcaf=pcaf.transform(Converted_dataframe)

print("Original Dimension:",Converted_dataframe.shape)
print("Transformed Dimension:",PlayStore_pcaf.shape)
print("Explained variance ratio:\n",pcaf.explained_variance_ratio_)


#%%


x=np.arange(1,len(np.cumsum(pcaf.explained_variance_ratio_))+1,1)
plt.plot(x,np.cumsum(pcaf.explained_variance_ratio_))
plt.xticks(x)
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Cumulative Explained Variance Vs Number of Components")
plt.suptitle("Reduced Feature Space")
plt.grid()
plt.show()


#%%


PlayStore_pcaf_df=pd.DataFrame(PlayStore_pcaf).corr()
column=[]
for i in range(PlayStore_pcaf.shape[1]):
    column.append(f'Priciple Column{i+1}')
plt.figure(figsize=(8,6))
sns.heatmap(PlayStore_pcaf_df,annot=True, xticklabels=column,yticklabels=column)
plt.title("Correlation Coefficient of Reduced Feature Space")
plt.show()


#%%


PlayStore_pcaf_df_with_target=pd.DataFrame(data=PlayStore_pcaf,columns=column)
PlayStore_pcaf_df_with_target

PlayStore_pcaf_df_with_target.insert(0,'Category',play_store_app_df['Category'])
print("Reduced Feature Dataframe:\n",PlayStore_pcaf_df_with_target.head())


#%%

# # Normality Test
# We will use at first Graphical method. We will use histogram and QQ plot.

# Histogram
plt.figure(figsize=(14, 8))
columns=['Minimum Installs','Maximum Installs','Rating','Rating Count','Price',
         'Installs']
for n,column in enumerate(columns):
    ax = plt.subplot(2, 3, n + 1)
    play_store_app_df_removed_outlier[column].hist(bins=30).plot(ax=ax)
    ax.set_title(f"Histogram of {column.upper()}")
    ax.set_ylabel("Frequency")
    ax.set_xlabel(f"{column.upper()}")

plt.tight_layout()
plt.show()


#%%

# qqplot(play_store_app_df_removed_outlier['Maximum Installs'],line='s')

# Histogram
plt.figure(figsize=(14, 8))
columns=['Minimum Installs','Maximum Installs','Rating','Rating Count','Price',
         'Installs']
for n,column in enumerate(columns):
    ax = plt.subplot(2, 3, n + 1)
    qqplot(play_store_app_df_removed_outlier[column],ax=ax)
    ax.set_title(column.upper())
    ax.set_title(f"QQ-Plot of {column.upper()}")
    ax.grid()

plt.tight_layout()
plt.show()


# From the both graph, we can see that this dataset does not come from
# gaussian distribution.


#%%

# # Data Transformation

# I am doing only 1 data Transformation in this project.
# One is total install number which is string and will convert to int.
play_store_app_df_removed_outlier['Install']=play_store_app_df_removed_outlier.Installs.str.replace('[+,",
"]', '').astype(float)
play_store_app_df['Install']=play_store_app_df.Installs.str.replace('[+,","]', '').astype(float)
#%%
import scipy.stats as st
play_store_app_df2 = play_store_app_df.copy()
play_store_app_df2['Install'] =
st.norm.ppf(st.rankdata(play_store_app_df2['Install'])/(len(play_store_app_df2['Install']) + 1))
play_store_app_df2['Minimum Installs'] = st.norm.ppf(st.rankdata(play_store_app_df2['Minimum
Installs'])/(len(play_store_app_df2['Minimum Installs']) + 1))
play_store_app_df2['Maximum Installs'] = st.norm.ppf(st.rankdata(play_store_app_df2['Maximum
Installs'])/(len(play_store_app_df2['Maximum Installs']) + 1))
play_store_app_df2['Rating'] =
st.norm.ppf(st.rankdata(play_store_app_df2['Rating'])/(len(play_store_app_df2['Rating']) + 1))
play_store_app_df2['Rating Count'] = st.norm.ppf(st.rankdata(play_store_app_df2['Rating
Count'])/(len(play_store_app_df2['Rating Count']) + 1))
play_store_app_df2['Price'] =
st.norm.ppf(st.rankdata(play_store_app_df2['Price'])/(len(play_store_app_df2['Price']) + 1))

#%%
plt.figure(figsize=(14, 8))
columns=['Minimum Installs','Maximum Installs','Rating','Rating Count','Price',
         'Installs']
for n,column in enumerate(columns):
    ax = plt.subplot(2, 3, n + 1)
    play_store_app_df2[column].hist(bins=30).plot(ax=ax)
    ax.set_title(f"Histogram of {column.upper()}")
    ax.set_ylabel("Frequency")
    ax.set_xlabel(f"{column.upper()}")
    ax.tick_params(axis='x', labelrotation=90)

plt.tight_layout()
plt.show()
```

```python
#%%


# # HeatMap & Pearson Co-relation Matrix

corr_matrix=play_store_app_df.corr(method='pearson')
sns.heatmap(corr_matrix,xticklabels=corr_matrix.columns,yticklabels=corr_matrix.columns,cmap='RdBu_r',
           annot=True,vmin=-1,vmax=1,linewidths=0.5, annot_kws={"fontsize":8})

plt.title("Correlation Between Features")
plt.show()

#%%


axs=scatter_matrix(corr_matrix, alpha = 0.3, figsize = (17, 12), diagonal = 'kde')
for ax in axs.flatten():
    ax.xaxis.label.set_rotation(90)
    ax.yaxis.label.set_rotation(0)
    ax.yaxis.label.set_ha('right')
plt.tight_layout()
plt.gcf().subplots_adjust(wspace=0, hspace=0)
plt.suptitle("Pairplot")
plt.show()

#%%


sns.pairplot(corr_matrix)
plt.show()


#%%
# Question 12: Statistics:

#%%

plt.figure(figsize = (8,4))
sns.kdeplot(data=play_store_app_df, x="Price",
    hue='Ad Supported', legend=True, fill=True)
# plt.xticks(rotation=90)
plt.suptitle("KDE of Price")
plt.grid(axis='y')
# plt.legend(loc='upper right')
plt.show()


#%%

price_morethan_0 = play_store_app_df[play_store_app_df['Price'] > 0]
plt.figure(figsize = (8,4))
sns.kdeplot(data=price_morethan_0, x="Price", y='Rating',
    hue='Ad Supported', legend=True, fill=True)
# plt.xticks(rotation=90)
plt.suptitle("bivariate distribution between App Price more than 0 and Rating")
plt.grid(axis='y')
# plt.legend(loc='upper right')
plt.show()




#%%

# Mean and median mode of each column
play_store_app_df_removed_outlier.describe()
```

```
#%%


corr_matrix=play_store_app_df.corr()
sns.heatmap(corr_matrix,xticklabels=corr_matrix.columns,yticklabels=corr_matrix.columns,cmap='RdBu_r',
           annot=True,linewidths=0.5, annot_kws={"fontsize":8})
plt.show()




# # Data Visualization:

#%%

# play_store_app_df


# ### Line Plot
# Plot a relationship Rating vs Installs for apps based on Apps are free or not.

#%%

plt.figure()
sns.lineplot(data=play_store_app_df_removed_outlier,x="Rating",
             y=play_store_app_df_removed_outlier.Installs.str.replace('[+,",")]',
'').astype(float),hue='Free')
plt.title("Installs Vs Rating")
plt.grid(axis='y')
# plt.legend()
plt.show()


# We can see that pattern is same for both free and paid apps.
# Trends are low in 0 and 5 Ratings app.

#%%

play_store_app_df_removed_outlier['Released'] =
pd.to_datetime(play_store_app_df_removed_outlier['Released'])
play_store_app_df_removed_outlier['year']=play_store_app_df_removed_outlier['Released'].dt.year
plt.figure()
sns.lineplot(data=play_store_app_df_removed_outlier,x="year",
             y='Rating', hue='Content Rating')
plt.title("App Rating Vs Released Year")
plt.grid(axis='y')
# plt.legend()
plt.show()


#%%
# # Bar-Plot : Stack, group
# Give the statistics of app category count


unique_count = play_store_app_df_removed_outlier.groupby(['Category','Free'],
as_index=False).agg(Total_Count=('Category', pd.Series.count))
print(unique_count)
rcParams['figure.figsize'] = 15,8

sns.barplot(x='Category',y='Total_Count',data=unique_count, hue='Free',palette=['r','b'])

# plt.figure(figsize=(15,8))
plt.xticks(rotation=90,fontsize=18)
plt.title("Count of App Categories",fontsize=22)
plt.tight_layout()
plt.xlabel("App Categories", fontsize=18)
plt.ylabel("Total Count of Apps", fontsize=18)
plt.grid(axis='y')
plt.show()

#%%


unique_count_Supported = play_store_app_df_removed_outlier.groupby(['Ad Supported','Category'],
as_index=False).agg(Total_Count=('Category', pd.Series.count))
rcParams['figure.figsize'] = 13,7
sns.barplot(x='Category',y='Total_Count',hue='Ad Supported',data=unique_count_Supported)
plt.xticks(rotation=90,fontsize=18)
plt.title("Count of App Categories by Ad Supported",fontsize=22)
plt.xlabel("App Categories", fontsize=18)
plt.ylabel("Total Count of Apps", fontsize=18)


plt.grid(axis='y')
plt.show()
#%%




#%%
# # Count-plot
plt.figure(figsize=(8,5))
sns.countplot(x='Content Rating',hue='Ad Supported',data=play_store_app_df_removed_outlier)
plt.title("Count of Content Rating by Ad Supported")
# plt.legend()
plt.grid(axis='y')
plt.show()

# # Cat-Plot

#%%


unique_count_Free = play_store_app_df_removed_outlier.groupby(['Ad Supported','year'],
as_index=False).agg(Total_Count=('year', pd.Series.count))
plt.figure(figsize=(7,13))
sns.catplot(x='year',y='Total_Count',
        data=unique_count_Free, hue='Ad Supported',
        kind="bar", legend=False,
        )
plt.xticks(rotation=90)
plt.title("Count Apps Over the Years by Ad Supported")
plt.legend(loc='upper left', title='Add supported')
plt.tight_layout()
plt.grid(axis='y')
plt.show()
# unique_count_Free


# # Pie-chart
#%%


# How many apps are free?
unique_count_Free = play_store_app_df_removed_outlier.groupby(['Free'],
                                                              as_index=False).agg(Total_Count=('Free',
pd.Series.count))

fig = plt.figure(figsize =(6, 4))
explode = [0.02, 0.2]
plt.pie(unique_count_Free['Total_Count'],  labels=unique_count_Free['Free'],
        explode = explode, autopct = '%.1f%%')
plt.axis('square')
plt.axis('equal')
plt.title("Percentage of Free apps")

plt.show()


#%%


# Content Rating allocation?
unique_count_Free = play_store_app_df_removed_outlier.groupby(['Content Rating'],
                                                              as_index=False).agg(Total_Count=('Content
Rating', pd.Series.count))
unique_count_Free = unique_count_Free.drop(unique_count_Free[unique_count_Free['Content Rating']
=='Unrated'].index)
print(unique_count_Free)
explode = [0.02, 0.1, 0.03,0.04,0.05]
fig = plt.figure(figsize =(10, 7))
plt.pie(unique_count_Free['Total_Count'],  labels=unique_count_Free['Content Rating'], autopct='%1.1f%%',
        # wedgeprops={'linewidth': 3.0, 'edgecolor': 'white'},textprops={'size': 'x-large'},
        explode=explode,
        shadow=True, startangle=45)
plt.axis('equal')
plt.axis('square')
plt.title("Content Rating allocation")

plt.show()


# # Displot

#%%

plt.figure(figsize = (8,4))
sns.displot(data=play_store_app_df, x="Rating",
    hue='Free', legend=True,
    stat='probability', kde=True)
# plt.xticks(rotation=90)
plt.suptitle("Distribution of App Rating")
plt.grid(axis='y')
# plt.legend(loc='upper right')
plt.show()




# # Pair Plot
#%%
# execution hidden as criterion already satisfied above
# sns.pairplot(play_store_app_df, kind="kde")
# corr_matrix=play_store_app_df.corr()
# sns.pairplot(corr_matrix,kind="kde")
# plt.show()

pairplot_df = play_store_app_df.copy()
pairplot_df.drop("Maximum Installs", axis=1, inplace=True)
pairplot_df.drop("Minimum Installs", axis=1, inplace=True)
pairplot_df.drop("Installs", axis=1, inplace=True)
# pairplot_df['Installs'] = pairplot_df['Installs'].astype(float)
# pairplot_df['Maximum Installs'] = pairplot_df['Maximum Installs'].astype(float)
sns.pairplot(pairplot_df.corr(),kind="kde")
plt.suptitle("Pairplot of Correlation between features", y=1.02)
plt.show()

# # HeatMap
#%%
# execution hidden as criterion already satisfied above
# sns.heatmap(corr_matrix,xticklabels=corr_matrix.columns,yticklabels=corr_matrix.columns,cmap='RdBu_r',
#             annot=True,linewidths=0.5)
# plt.show()



# # Histplot
#%%
plt.figure()
sns.histplot(data=play_store_app_df,x='Installs', hue='Free')
plt.xticks(rotation = 90)
plt.title("Histogram of App Installs")
plt.grid(axis='y')
plt.show()

#%%
plt.figure()
sns.histplot(data=play_store_app_df,x='Rating', hue='Ad Supported')
plt.xticks(rotation = 90)
plt.title("Histogram of App Ratings by Ad Supported")
plt.grid(axis='y')
plt.show()


# # QQ Plot

#%%



plt.figure(figsize=(14, 8))
columns=['Minimum Installs','Rating','Installs']
for n,column in enumerate(columns):
    ax = plt.subplot(1, 3, n + 1)
    qqplot(play_store_app_df_removed_outlier[column],ax=ax)
    ax.set_title(column.upper())
    ax.set_xlabel("")
    # ax.tick_params(labelrotation=90)
plt.tight_layout()
plt.show()


# # KDE

#%%



plt.figure(figsize=(8, 6))
sns.kdeplot(data=play_store_app_df,x='Rating',hue='In App Purchases', fill=True)
plt.title("KDE plot of App Ratings by In App Purchase")
plt.grid(axis='y')
plt.show()




#%%
# # Scatter plot and regression line using sklearn
### # execution hidden as it takes lot of time to execute, output shown in the report

# ax = sns.regplot(x="Install", y="Rating", data=play_store_app_df,
#          fit_reg=True)
# plt.title("Scatterplot of Rating Vs Install with Regression Line")
# plt.show()





# # Multivariate Box Plot
#%%
```

```
plt.figure(figsize=(24, 8))
sns.boxplot(x="Category", y="Rating", data=play_store_app_df_removed_outlier)
plt.ylabel('Rating')
plt.xlabel('Category')
plt.title("Box plot of Rating Vs Categories")
plt.xticks(rotation=90)
plt.grid()
plt.show()

#%%

plt.figure(figsize=(10, 8))
sns.boxplot(x="Editors Choice", y="Rating",hue="Free", data=play_store_app_df)
plt.ylabel('Rating')
plt.xlabel('Editors Choice')
plt.title("Box plot of Rating Vs Editors Choice by Free Apps")
plt.grid()
plt.show()

# # Violin Plot

#%%
plt.figure(figsize=(14, 8))
columns=['Minimum Installs','Rating','Maximum Installs','Price','Rating Count','Install']
for n,column in enumerate(columns):
    ax = plt.subplot(2, 3, n + 1)
    sns.violinplot(x=play_store_app_df_removed_outlier[column],ax=ax)
    ax.set_title(column.upper())
    ax.set_xlabel("")
    ax.tick_params(labelrotation=90)
plt.tight_layout()
plt.show()

# # SubPlots

#%%

top5Apps=play_store_app_df.sort_values(by=['Install'],ascending=False).head(5)
Category_count = play_store_app_df_removed_outlier.groupby(['Category'],
                                                              as_index=False).agg(Total_Count=('Category',
pd.Series.count))
top5Category=Category_count.sort_values(by=['Total_Count'],ascending=False).head(5)
Minimum_android = play_store_app_df_removed_outlier.groupby(['Minimum Android'],
                                                              as_index=False).agg(Total_Count=('Minimum
Android', pd.Series.count))
top5Androidcoverage=Minimum_android.sort_values(by=['Total_Count'],ascending=False).head(5)
top5Androidcoverage


#%%

# In this subplots history we will discuss about the app history statistics.

plt.figure(figsize=(10, 8))
ax = plt.subplot(2, 2, 1)
sns.barplot(x="App Name", y="Install", data=top5Apps)
plt.xticks(rotation=90)
ax.set_title("Top 5 Apps")
# ax.xticks(rotation=90)
ax = plt.subplot(2, 2, 2)
sns.barplot(x="Category", y="Total_Count", data=top5Category)
plt.xticks(rotation=12)
ax.set_title("Top 5 Category")
ax = plt.subplot(2, 2, 3)
sns.histplot(data=play_store_app_df,x='Rating')
ax.set_title("Ratings Distributions")
```

```
ax = plt.subplot(2, 2, 4)
sns.barplot(x="Minimum Android", y="Total_Count", data=top5Androidcoverage)
plt.xticks(rotation=12)
ax.set_title("Top 5 Minimum Android")


plt.tight_layout()
plt.show()

#%%
```

Source code of Dash App:

```
#%%
from dash import dash_table
import plotly.express as px
import plotly.graph_objs as go
import dash as dash
from dash import dcc
from dash import html
import pandas as pd
import dash_bootstrap_components as dbc
from datetime import date, datetime
# import time
import plotly.figure_factory as ff
from statsmodels.graphics.gofplots import qqplot
import scipy.stats as st


from dash.dependencies import Input, Output, State

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
play_store_apps_df=pd.read_csv('data/Play_Store_Dash_new.csv')

Categorylist=play_store_apps_df.Category.unique()
content_rating=play_store_apps_df['Content Rating'].unique()
# print(play_store_apps_df)

temp_df=pd.DataFrame()
csv_df=pd.DataFrame()
filter_df=pd.DataFrame()
filter_df=play_store_apps_df

my_app = dash.Dash('Visualizing Google Play Store App Market', external_stylesheets=external_stylesheets)

my_app.layout = html.Div([ html.H1('Visualizing Google Play Store App Market', style =
{'textAlign':'center'}),
                html.Br(),
                dcc.Tabs(id = 'project',
                    children = [
                        dcc.Tab(label='Statistics', value ='app'),
                        dcc.Tab(label='Comparision', value ='compare'),
                        dcc.Tab(label='App Details', value ='details'),
                        dcc.Tab(label='Graph Visualization', value ='graph'),
                        dcc.Tab(label='Normality Test', value ='normality'),

                        ]),
                html.Div(id = 'layout')
```

```
])
########## Apps Statistics Layout ###########
app_statistics_layout=html.Div([
            html.H6("App Statistics Layout",style={'textAlign': 'center'}),
            html.Div([ html.H6('Filters'),
            html.Div([

                html.Div([
                    # html.P("Category"),
                    dcc.Dropdown(placeholder="Category",
                    options=[{'label':category, 'value':category} for category in
Categorylist],id='Category-Selector'
                    )
                    ],
                    style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Free"),
                    dcc.RadioItems(['Yes','No'],id="free-radio")
                    ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Ad Supported"),
                    dcc.RadioItems(['Yes','No'],id='ad-radio'),
                ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("In App Purchase"),
                    dcc.RadioItems(['Yes','No'],id='purchase-radio'),
                ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Editor's Choice"),
                    dcc.RadioItems(['Yes','No'],id='editor-radio'),
                ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Relase Date"),
                    # dcc.DatePickerRange(
                    # id='my-date-picker-range',
                    # max_date_allowed=dt.today,
                    # start_date_placeholder_text='YYYY-MM-DD',
                    # end_date=dt.today
                    # ),
                    dcc.DatePickerRange(
                        display_format='YYYY-MM-DD',
                        clearable=True,
                        max_date_allowed=date.today(),
                        # end_date=date.today(),
                        # start_date=date(2010,1,28),
                        end_date=date.today(),
                        id='realse-picker1'
                    ),
                ],style={'width': '25%','display': 'inline-block'}),

            ],style={'margin-bottom':'20px'}),
            html.Div([
                # html.Div([
                #     html.P("App Ratings"),
                #     dcc.RangeSlider(min=-1, max=5, step=1, value=[-1], id='app-ratings-slider'),
                # ],style={'width': '20%','display': 'inline-block','margin-right':'20px'}),
                html.Div([
                    html.P("Content Rating"),
                    # dcc.RangeSlider(min=-1, max=5, step=1, value=[-1], id='my-range-slider'),
                    # dcc.Checklist(['New York City', 'Montréal', 'San Francisco','New York City',
'Montréal', 'San Francisco'],inline=True)
                    dcc.Checklist(content_rating,inline=True,id='content-ratings-checklist2')

                ],style={'width': '80%','display': 'inline-block'}),
                html.Div([
                    # html.P("App Ratings"),
                    html.Button('Reset Filter',id='reset-button',n_clicks=0)
                ],style={'width': '20%','display': 'inline-block'}),
                html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
            ]),

            ]),
            html.Div([
                # dcc.Graph(id = 'firstsubplot')
            ],id="Statistics",style={'textAlign': 'center'})
])
########## Apps Statistics Layout ###########

@my_app.callback(Output(component_id='Category-Selector', component_property='value'),
                Output(component_id='free-radio', component_property='value'),
                Output(component_id='ad-radio', component_property='value'),
                Output(component_id='purchase-radio', component_property='value'),
                Output(component_id='editor-radio', component_property='value'),
                Output(component_id='realse-picker1', component_property='start_date'),
                Output(component_id='realse-picker1', component_property='end_date'),
                Output(component_id='content-ratings-checklist2', component_property='value'),
                Input('reset-button','n_clicks'),)
def reset_button(n_clicks):
    return None,None,None,None,None,None,date.today(),[]

###################### Filter Selection Functions ################
# Output(component_id='First_graph', component_property='figure'),
#                  Output(component_id='Second_graph', component_property='figure'),
#                  Output(component_id='Third_graph', component_property='figure'),
#                  Output(component_id='Fourth_graph', component_property='figure')

@my_app.callback(Output(component_id='Statistics', component_property='children'),
                [Input(component_id='Category-Selector', component_property='value'),
                Input(component_id='free-radio', component_property='value'),
                Input(component_id='ad-radio', component_property='value'),
                Input(component_id='purchase-radio', component_property='value'),
                Input(component_id='editor-radio', component_property='value'),
                Input(component_id='realse-picker1', component_property='start_date'),
                Input(component_id='realse-picker1', component_property='end_date'),
                Input(component_id='content-ratings-checklist2', component_property='value'),
                # Input(component_id='app-ratings-slider', component_property='value')
                ]
                )
def Filter(Category,free,ad,purchase,editor,start_date,end_date,contentRatings):
    play_store_apps_df_filtered=play_store_apps_df
    # print(play_store_apps_df_filtered)
    if Category is not None:

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered.Category==Category
]
        print(play_store_apps_df_filtered.shape)
    else:
        print("Category Empty")
    if free is not None:
        if(free == 'Yes'):

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered.Free==True]
            print(play_store_apps_df_filtered.shape)
        else:

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered.Free==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("free empty")
    if ad is not None:
        if(ad == 'Yes'):
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Ad
Supported']==True]
            print(play_store_apps_df_filtered.shape)
        else:
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Ad
Supported']==False]
```

```
            print(play_store_apps_df_filtered.shape)
    else:
        print("ad empty")
    if purchase is not None:
        if(purchase == 'Yes'):
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['In
App Purchases']==True]
            print(play_store_apps_df_filtered.shape)
        else:
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['In
App Purchases']==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("purchase empty")
    if editor is not None:
        if(editor == 'Yes'):

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Editors
Choice']==True]
            print(play_store_apps_df_filtered.shape)
        else:

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Editors
Choice']==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("editor empty")
    if start_date is not None:
        if end_date is None:
            print(date.today())
            end_date=date.today()

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[(play_store_apps_df_filtered['Released']>start
_date) &(play_store_apps_df_filtered['Released']<end_date) ]
        print(play_store_apps_df_filtered.shape)
    else:
        print("start_date empty")
    if contentRatings is not None and (len(contentRatings)>0):
        play_store_apps_df_filtered=play_store_apps_df_filtered.loc[(play_store_apps_df_filtered['Content
Rating'].isin(contentRatings))]
        print(play_store_apps_df_filtered.shape)
    else:
        print("contentRatings empty")
    # if(appratings[0]!=-1):
    #     start=appratings[0]
    #     end=start+1
    #
play_store_apps_df_filtered=play_store_apps_df_filtered.loc[((play_store_apps_df_filtered['Rating']>=start
) & (play_store_apps_df_filtered['Rating']<end) )]
    #     print(play_store_apps_df_filtered.shape)
    # else:
    #     print("appratings empty")
    # print(play_store_apps_df)
    print(play_store_apps_df_filtered.shape)
    # if(play_store_apps_df_filtered.shape[0]>5):
    #     print("Enongh Data")
    # else:
    #     print("Not Enough Data")

    if(play_store_apps_df_filtered.shape[0]<=1):
        filter_df=play_store_apps_df_filtered
        filtered_div=html.Div([
            html.H6("There is no data with selected filter")
        ])
    else:
        # temp_year=pd.DataFrame()
        # play_store_apps_df[]
        play_store_apps_df_filtered['Released'] = pd.to_datetime(play_store_apps_df['Released'])
        play_store_apps_df_filtered['year']=play_store_apps_df['Released'].dt.year
        sorted_df=play_store_apps_df_filtered
        sorted_df=sorted_df.sort_values(by='Installs',ascending=False)
        sorted_df=sorted_df.head(5)

        temp_years=pd.DataFrame()
        temp_years =
play_store_apps_df_filtered.year.value_counts(sort=True,ascending=False).rename_axis('year').reset_index(n
ame='counts')
        temp_years=temp_years.sort_values(by=['year'])
        # print(sorted_df)
        # print(play_store_apps_df_filtered.describe())
        describe_df=play_store_apps_df_filtered.describe()
        describe_df=describe_df.drop(['year','Unnamed: 0'],axis=1)
        columns_list=describe_df.columns
        # columns_list=columns_list.drop(['Unnamed 0'])
        print(columns_list)
        figure1=px.bar(sorted_df,x='App Name',y='Installs',title="Top Apps based on Installation")
        # figure2=px.line(describe_df,x=describe_df.index,y='Installs',title="Dataset Statistics based on
")
        figure3=px.box(temp_years,x='year',y='counts',points='all',title="Year Wise App Counts")
        # figure3=px.box(temp_years,x='year')
        # figure4=px.histogram(play_store_apps_df_filtered,x='Rating')
        hist_data1=play_store_apps_df_filtered['Rating']
        hist_data2=play_store_apps_df_filtered['Installs']
        hist_data=[hist_data1]
        group_labels=['Rating']
        fig=ff.create_distplot(hist_data=hist_data,group_labels=group_labels)
        fig.layout.update({'title': 'Distribution Plot for Ratings'})
        filtered_div=html.Div([
        html.H6('Statistics'),
        html.Div([
                    html.Div([
                        # html.P("First Subplot"),
                        # dcc.Graph(id="First_graph"),
                        dcc.Graph(figure=figure1),
                    ],style={'width': '50%','display': 'inline-block'},id='first_plot'),
                    html.Div([
                        # html.P("Second Subplot"),
                        dcc.Dropdown(placeholder="Column",
                    options=[{'label':columnss, 'value':columnss} for columnss in
columns_list],id='column-Selector',value='Rating'
                    ),
                        # dcc.Graph(id="Second_graph"),
                        dcc.Graph(id='filter_graph2'),
                    ],style={'width': '50%','display': 'inline-block'},id="second_plot"),
                ],id="first_subplot_div"),

                html.Div([
                    html.Div([
                        # html.P("Third Subplot"),

                        # dcc.Graph(id="Third_graph"),
                        dcc.Graph(figure=figure3),
                    ],style={'width': '50%','display': 'inline-block'},id="third_plot"),
                    html.Div([
                        # html.P("Fourth Subplot"),
                        # dcc.Graph(id="Fourth_graph"),
                        # html.P(""),
                        dcc.Graph(figure=fig),
                    ],style={'width': '50%','display': 'inline-block'},id="fourth_plot"),
                ],id="second_subplot_div"),

    ])



    return filtered_div
```

```
###################### Filter Selection Functions ################

########## Apps Comparision Layout ###########
app_comparision_layout=html.Div([

            html.Div([html.H5("App Comparision Layout",style={'textAlign': 'center'}),

            html.Div([
                    # html.P("Category",style={'width':'50%','left':'50%','right':'auto'}),
                    dcc.Dropdown(placeholder="Category",
                    options=[{'label':category, 'value':category} for category in
Categorylist],id='Category-Selector-comparision',style={'width':'50%','left':'50%','right':'auto'}
                    ),],style={"padding-bottom":'20px'}),
            html.Div([
                html.P("Please select the first apps to compare"),
            ],style={'width': '40%','display': 'inline-block'},id="first_app_dropdown"),
            html.Div([
                html.P("Please select the second apps to compare"),
            ],style={'width': '40%','display': 'inline-block'},id="second_app_dropdown"),
            html.Div([
                # html.P("Please select the second apps to compare"),
                html.Button('Compare',id='compare_button',n_clicks=0)
            ],style={'width': '20%','display': 'inline-block'}),
            ]),
            html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
            html.Div([html.H6("Comparision Panel",style={'textAlign': 'center'}),


            ],id="comaprision_result"),

])
########## Apps Comparision Layout ###########

#####column selector callback ##########
@my_app.callback(Output(component_id='filter_graph2', component_property='figure'),
                    [Input(component_id='column-Selector', component_property='value'),])

def column_updater(column_value):
    # print(play_store_apps_df)
    # print(filter_df)
    describes_df=filter_df.describe()
    describes_df=describes_df.drop(['year'],axis=1)
    figure2=px.line(describes_df,x=describes_df.index,y=column_value,title=f"Dataset Statistics based on
{column_value} ")

    return figure2


#####column selector callback ##########
############# Comparision CallBack for Category ##############
@my_app.callback([Output(component_id='first_app_dropdown',component_property='children'),
                  Output(component_id='second_app_dropdown',component_property='children')],
                  [Input(component_id='Category-Selector-comparision',component_property='value')])
def categoryComparisionUpdate(category):
    print("Comparision pdf")
    print(category)
    play_store_compare=play_store_apps_df
    if category is None:
        appslist=play_store_compare['App Name'].unique()
        print(appslist.shape)
    else:
        play_store_compare=play_store_compare.loc[play_store_compare.Category==category]
        appslist=play_store_compare['App Name'].unique()
        print(appslist.shape)
    first_app_dropdown=html.Div([
    html.P("Please select the first apps to compare"),
    dcc.Dropdown(placeholder="First App",
                options=[{'label':app, 'value':app} for app in
appslist],id='First_App_Comparision',style={'width':'80%'})
    ])
    second_app_dropdown=html.Div([
    html.P("Please select the second apps to compare"),
    dcc.Dropdown(placeholder="Second App",
                options=[{'label':app, 'value':app} for app in appslist],id='Second_App_Comparision',
                style={'width':'80%'})])
    # time.sleep(2)
    return first_app_dropdown,second_app_dropdown
########## call back for comparision layer ############
@my_app.callback(Output(component_id='comaprision_result',component_property='children'),
                [Input(component_id="compare_button",component_property="n_clicks")],
                [State("First_App_Comparision","value"),
                State("Second_App_Comparision","value"),])
def comparision_result(n_clicks,apps1,apps2):
    print(n_clicks)
    print("Comapre Button Clicks")
    if(apps1 is None or apps2 is None):
        if(apps1 is None and apps2 is None):
            divs=html.Div([
                html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
                html.H5("Comparision Panel",style={'textAlign': 'center'}),
                html.P("You have not selected any apps for comparing")
            ])
            return divs
        elif(apps2 is None):
            divs=html.Div([
                html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
                html.H5("Comparision Panel",style={'textAlign': 'center'}),
                html.P("Please select the Second Apps")
            ])
            return divs
        else:
            divs=html.Div([
                html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
                html.H5("Comparision Panel",style={'textAlign': 'center'}),
                html.P("Please select the First Apps")

            ])
            return divs
    elif(apps1==apps2):
        divs=html.Div([
            html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
            html.H5("Comparision Panel",style={'textAlign': 'center'}),
            html.P("You have selected the same apps")
            ])
        return divs
    else:
        print(apps1)
        print(apps2)
        df1=play_store_apps_df.loc[play_store_apps_df['App Name']==apps1]
        df2=play_store_apps_df.loc[play_store_apps_df['App Name']==apps2]
        df1=df1.sort_values(by='Installs',ascending=False)
        df2=df2.sort_values(by='Installs',ascending=False)
        df1=df1[:1]
        df2=df2[:1]
        final_df=pd.concat([df1,df2])
        apps1_score=0
        apps2_score=0
        if(df1.Rating.values[0]>df2.Rating.values[0]):
            apps1_score=apps1_score+1
        elif(df2.Rating.values[0]>df1.Rating.values[0]):
            apps2_score=apps2_score+1
        if(df1.Free.values[0]==True):
            apps1_score=apps1_score+1
        else:
            apps1_score=apps1_score-1
        if(df2.Free.values[0]==True):
            apps2_score=apps2_score+1
        else:
            apps2_score=apps2_score-1
        if(df1.Installs.values[0]>df2.Installs.values[0]):
            apps1_score=apps1_score+1
        elif(df2.Installs.values[0]>df1.Installs.values[0]):
            apps2_score=apps2_score+1
        if(df1.Price.values[0]>df2.Price.values[0]):
            apps1_score=apps1_score+1
        elif(df2.Price.values[0]>df1.Price.values[0]):
            apps2_score=apps2_score+1
        if(df1['Rating Count'].values[0]>df2['Rating Count'].values[0]):
            apps1_score=apps1_score+1
        elif(df2['Rating Count'].values[0]>df1['Rating Count'].values[0]):
            apps2_score=apps2_score+1

        print(apps1_score)
        print(apps2_score)
        if(apps1_score>apps2_score):
            score=f"{apps1} has the higher score than {apps2} based on some criteria."
        elif(apps2_score>apps1_score):
            score=f"{apps2} has the higher score than {apps1} based on some criteria."
        else:
            score=f"{apps1} has the same score as {apps2}"
        fig_rating_color={}
        fig_rating_color[0]='red'
        fig_rating_color[1]='green'
        fig_ratings=px.bar(final_df,x='App Name',y='Rating',color='App
Name',color_discrete_map={f'{apps1}':'navy',f'{apps2}':'aqua'})
        fig_ratings_count=px.bar(final_df,x='App Name',y='Rating Count',color='App
Name',color_discrete_map={f'{apps1}':'maroon',f'{apps2}':'purple'})
        fig_Installs=px.bar(final_df,x='App Name',y='Installs',color='App
Name',color_discrete_map={f'{apps1}':'red',f'{apps2}':'green'})
        fig_maximum_installs=px.bar(final_df,x='App Name',y='Maximum Installs',color='App
Name',color_discrete_map={f'{apps1}':'blueviolet',f'{apps2}':'bisque'})
        html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
        divs=html.Div([html.H5("Comparision Panel",style={'textAlign': 'center'}),

        html.Div([
            html.Div([
                # html.H6(f"Comaprision of Ratings between {apps1} and {apps2}",style={'textAlign':
'center'}),
                html.H6(f"Comaprision of Ratings ",style={'textAlign': 'center'}),
                dcc.Graph(figure=fig_ratings)
            ],style={'width': '50%','display': 'inline-block'}),
            html.Div([
                # html.H6(f"Comaprision of Ratings Count between {apps1} and {apps2}",style={'textAlign':
'center'}),
                html.H6(f"Comaprision of Ratings Count ",style={'textAlign': 'center'}),
                dcc.Graph(figure=fig_ratings_count)
            ],style={'width': '50%','display': 'inline-block'}),
        ]),
        html.Div([
            html.Div([
                # html.H6(f"Comaprision of Installs between {apps1} and {apps2}",style={'textAlign':
'center'}),
                html.H6(f"Comaprision of Installs",style={'textAlign': 'center'}),
                dcc.Graph(figure=fig_Installs)
            ],style={'width': '50%','display': 'inline-block'}),
            html.Div([
                # html.H6(f"Comaprision of Maximum Installs between {apps1} and
{apps2}",style={'textAlign': 'center'}),
                html.H6(f"Comaprision of Maximum Installs",style={'textAlign': 'center'}),
                dcc.Graph(figure=fig_maximum_installs)
            ],style={'width': '50%','display': 'inline-block'}),
        ]),
        #['Unnamed: 0', 'App Name', 'App Id', 'Category', 'Rating',
    #    'Rating Count', 'Installs', 'Minimum Installs', 'Maximum Installs',
    #    'Free', 'Price', 'Currency', 'Size', 'Minimum Android', 'Developer Id',
    #    'Developer Website', 'Developer Email', 'Released', 'Last Updated',
    #    'Content Rating', 'Privacy Policy', 'Ad Supported', 'In App Purchases',
    #    'Editors Choice', 'Scraped Time']
    html.Div([
                html.H6("Comparision Result",style={'textAlign': 'center'}),
                html.B(f"{score}",style={'textAlign': 'center'})

    ]),
        html.Div([
                html.H6("Comparision Table",style={'textAlign': 'center'}),
                html.Table([
                    html.Tr([
                        html.Th(''),
                        html.Th(f'{apps1}'),
                        html.Th(f'{apps2}'),

                    ]),
                    html.Tr([
                        html.Td('App Id'),
                        html.Td(f'{df1["App Id"].values[0]}'),
                        html.Td(f'{df2["App Id"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Category'),
                        html.Td(f'{df1["Category"].values[0]}'),
                        html.Td(f'{df2["Category"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Free'),
                        html.Td(f'{df1["Free"].values[0]}'),
                        html.Td(f'{df2["Free"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Price'),
                        html.Td(f'{df1["Price"].values[0]}'),
                        html.Td(f'{df2["Price"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Currency'),
                        html.Td(f'{df1["Currency"].values[0]}'),
                        html.Td(f'{df2["Currency"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Size'),
                        html.Td(f'{df1["Size"].values[0]}'),
                        html.Td(f'{df2["Size"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Minimum Android'),
                        html.Td(f'{df1["Minimum Android"].values[0]}'),
                        html.Td(f'{df2["Minimum Android"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Developer Id'),
                        html.Td(f'{df1["Developer Id"].values[0]}'),
                        html.Td(f'{df2["Developer Id"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Developer Website'),
                        # html.Td(f'{df1["Developer Website"].values[0]}'),
                        # html.Td(f'{df2["Developer Website"].values[0]}'),
                        html.Td(html.A(children=f'{df1["Developer
Website"].values[0]}',href=f'{df1["Developer Website"].values[0]}',style={'color': 'blue', 'text-
decoration': 'none'}  ),),
                        html.Td(html.A(children=f'{df2["Developer
Website"].values[0]}',href=f'{df2["Developer Website"].values[0]}',style={'color': 'blue', 'text-
decoration': 'none'}  ))

                    ]),
                    html.Tr([
                        html.Td('Developer Email'),
                        html.Td(f'{df1["Developer Email"].values[0]}'),
                        html.Td(f'{df2["Developer Email"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Released'),
                        html.Td(f'{df1["Released"].values[0]}'),
                        html.Td(f'{df2["Released"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Content Rating'),
                        html.Td(f'{df1["Content Rating"].values[0]}'),
                        html.Td(f'{df2["Content Rating"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Privacy Policy'),
                        # html.Td(f'{df1["Privacy Policy"].values[0]}'),
                        # html.Td(f'{df2["Privacy Policy"].values[0]}'),
                        html.Td(html.A(children=f'{df1["Privacy Policy"].values[0]}',href=f'{df1["Privacy
Policy"].values[0]}',style={'color': 'blue', 'text-decoration': 'none'}  ),),
                        html.Td(html.A(children=f'{df2["Privacy Policy"].values[0]}',href=f'{df2["Privacy
Policy"].values[0]}',style={'color': 'blue', 'text-decoration': 'none'}  ))

                    ]),
                    html.Tr([
                        html.Td('Ad Supported'),
                        html.Td(f'{df1["Ad Supported"].values[0]}'),
                        html.Td(f'{df2["Ad Supported"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('In App Purchases'),
                        html.Td(f'{df1["In App Purchases"].values[0]}'),
                        html.Td(f'{df2["In App Purchases"].values[0]}'),

                    ]),
                    html.Tr([
                        html.Td('Editors Choice'),
                        html.Td(f'{df1["Editors Choice"].values[0]}'),
                        html.Td(f'{df2["Editors Choice"].values[0]}'),

                    ]),
                    # html.Tr([
                    #     html.Td('Last Updated'),
                    #     html.Td(f'{df1["Last Updated"].values[0]}'),
                    #     html.Td(f'{df2["Last Updated"].values[0]}'),

                    # ])


                ],style={'margin':'0px auto','text-align':'center','padding-bottom':'30px'})

        ])

        ])
        return divs
```

```
########## call back for comparision layer ############
############# Comparision CallBack for Category ##############

############### code block for app details layout ##########################

app_details_layout=html.Div([
            html.H6("App Details Details",style={'textAlign': 'center'}),
            html.Div([ html.H6('Filters'),
            html.Div([

                html.Div([
                    # html.P("Category"),
                    dcc.Dropdown(placeholder="Category",
                    options=[{'label':category, 'value':category} for category in
Categorylist],id='Category-Selector1'
                    )
                    ],
                    style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Free"),
                    dcc.RadioItems(['Yes','No'],id="free-radio1")
                    ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Ad Supported"),
                    dcc.RadioItems(['Yes','No'],id='ad-radio1'),
                ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("In App Purchase"),
                    dcc.RadioItems(['Yes','No'],id='purchase-radio1'),
                ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Editor's Choice"),
                    dcc.RadioItems(['Yes','No'],id='editor-radio1'),
                ],style={'width': '15%','display': 'inline-block'}),
                html.Div([
                    html.P("Relase Date"),
                    # dcc.DatePickerRange(
                    # id='my-date-picker-range',
                    # max_date_allowed=dt.today,
                    # start_date_placeholder_text='YYYY-MM-DD',
                    # end_date=dt.today
                    # ),
                    dcc.DatePickerRange(
                        display_format='YYYY-MM-DD',
                        clearable=True,
                        max_date_allowed=date.today(),
                        end_date=date.today(),
                        id='realse-picker'
                    ),
                ],style={'width': '25%','display': 'inline-block'}),

            ],style={'margin-bottom':'20px'}),
            html.Div([
                html.Div([
                    html.P("App Ratings"),
                    dcc.RangeSlider(min=0, max=5, step=1, value=[-1], id='app-ratings-slider'),
                ],style={'width': '20%','display': 'inline-block','margin-right':'20px'}),
                html.Div([
                    html.P("Content Rating"),
                    # dcc.RangeSlider(min=-1, max=5, step=1, value=[-1], id='my-range-slider'),
                    # dcc.Checklist(['New York City', 'Montréal', 'San Francisco','New York City',
'Montréal', 'San Francisco'],inline=True)
                    dcc.Checklist(content_rating,inline=True,id='content-ratings-checklist')

                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    # html.P("App Ratings"),
                    html.Button('Reset Filter',id='reset-button1',n_clicks=0)
                ],style={'width': '20%','display': 'inline-block'})
            ]),]),

            html.Div([

            ],id='app_list'),

            html.Div([],id='singleappdetails')
])
############### code block for app details layout ##########################
@my_app.callback(Output(component_id='Category-Selector1', component_property='value'),
                Output(component_id='free-radio1', component_property='value'),
                Output(component_id='ad-radio1', component_property='value'),
                Output(component_id='purchase-radio1', component_property='value'),
                Output(component_id='editor-radio1', component_property='value'),
                Output(component_id='app-ratings-slider', component_property='value'),
                Output(component_id='content-ratings-checklist', component_property='value'),
                Output(component_id='realse-picker', component_property='start_date'),
                Output(component_id='realse-picker', component_property='end_date'),
                Input('reset-button1','n_clicks'),)
def reset_button(n_clicks):
    return None,None,None,None,None,[0],[],None,date.today()
##################### callback for app list ##################
@my_app.callback(Output(component_id='app_list',component_property='children'),
                [Input(component_id='Category-Selector1', component_property='value'),
                Input(component_id='free-radio1', component_property='value'),
                Input(component_id='ad-radio1', component_property='value'),
                Input(component_id='purchase-radio1', component_property='value'),
                Input(component_id='editor-radio1', component_property='value'),
                Input(component_id='realse-picker', component_property='start_date'),
                Input(component_id='realse-picker', component_property='end_date'),
                Input(component_id='content-ratings-checklist', component_property='value'),
                Input(component_id='app-ratings-slider', component_property='value')]
                )
def Filter(Category,free,ad,purchase,editor,start_date,end_date,contentRatings,appratings):
    play_store_apps_df_filtered=play_store_apps_df
    # print(play_store_apps_df_filtered)
    if Category is not None:

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered.Category==Category
]
        print(play_store_apps_df_filtered.shape)
    else:
        print("Category Empty")
    if free is not None:
        if(free == 'Yes'):

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered.Free==True]
            print(play_store_apps_df_filtered.shape)
        else:

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered.Free==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("free empty")
    if ad is not None:
        if(ad == 'Yes'):
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Ad
Supported']==True]
            print(play_store_apps_df_filtered.shape)
        else:
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Ad
Supported']==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("ad empty")
    if purchase is not None:
        if(purchase == 'Yes'):
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['In
App Purchases']==True]
            print(play_store_apps_df_filtered.shape)
        else:
            play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['In
App Purchases']==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("purchase empty")
    if editor is not None:
        if(editor == 'Yes'):

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Editors
Choice']==True]
            print(play_store_apps_df_filtered.shape)
        else:

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[play_store_apps_df_filtered['Editors
Choice']==False]
            print(play_store_apps_df_filtered.shape)
    else:
        print("editor empty")
    if start_date is not None:
        if end_date is None:
            print(date.today())
            end_date=date.today()

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[(play_store_apps_df_filtered['Released']>start
_date) &(play_store_apps_df_filtered['Released']<end_date) ]
        print(play_store_apps_df_filtered.shape)
    else:
        print("start_date empty")
    if contentRatings is not None and (len(contentRatings)>0):
        play_store_apps_df_filtered=play_store_apps_df_filtered.loc[(play_store_apps_df_filtered['Content
Rating'].isin(contentRatings))]
        print(play_store_apps_df_filtered.shape)
    else:
        print("contentRatings empty")
    if(appratings[0]!=0):
        start=appratings[0]
        end=start+1

play_store_apps_df_filtered=play_store_apps_df_filtered.loc[((play_store_apps_df_filtered['Rating']>=start
) & (play_store_apps_df_filtered['Rating']<end) )]
        print(play_store_apps_df_filtered.shape)
    else:
        print("appratings empty")

    score=play_store_apps_df_filtered.shape[0]
    print(type(play_store_apps_df))

    global temp_df
    global csv_df
    csv_df=play_store_apps_df_filtered
    # temp_df=play_store_apps_df_filtered[['Index','App Name','Category','Rating','Rating
Count','Installs','Maximum Installs'
    # ,'Free','Price','Size','Minimum Android','Released','Content Rating','In App Purchases','Editors
Choice','Ad Supported']]

    temp_df=play_store_apps_df_filtered[['App Name','Category','Rating','Rating Count','Installs'
    ,'Free','Price','Size','Minimum Android','Released','Content Rating']]
    # temp_df=play_store_apps_df_filtered

    div=html.Div([
        html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
        html.H6(f"Total Apps Count: {score}",style={'text-align':'center',}),
        html.H5("Details of Application",style={'text-align':'center'}),
        dash_table.DataTable(
            id="app_details_table",
            columns=[{"name":i,"id":i} for i in (temp_df.columns) ],
            page_current=0,
            page_size=10,
            page_action='custom',
            sort_mode='multi',
            # filter_action='native',
            sort_by=[],
            selected_rows=['App Name'],
            style_data_conditional=[
               {'if':{
                    'filter_query': '{Free} contains "true"',
                    'column_id': 'Free'
                },
                'backgroundColor': 'green',
                'color':'white'
               },
               {'if':{
                    'filter_query': '{Free} contains "false"',
                    'column_id': 'Free'
                },
                'backgroundColor': 'red',
                'color':'white'
               }
            ]

        ),
        html.P("**** Click on the App to see details ****",style={'color':'red'}),
        html.Div([

            html.P("Download the Filterd CSV   :  ",style={'display': 'inline-block'}),
            html.Button('Download CSV',id='csv_download_Button',style={'display': 'inline-block'}),
            dcc.Download(id='save_csv_promot'),
        ],style={'display':'flex','justify-content':'flex-end','padding-top':'30px'}),

        # dbc.Pagination(max_value=page,fully_expanded=False)

    ])
    return div
################# dash table #################
@my_app.callback(Output('app_details_table', "data"),
                Output(component_id='singleappdetails',component_property='children'),
                Input('app_details_table', "page_current"),
                Input('app_details_table', "page_size"),
                Input('app_details_table', "sort_by"),
                Input('app_details_table', 'active_cell'))
def update_dash_table(page_current,page_size,sort_by,active_cell):
    # print(temp_df)
    # active_row_id=active_cell['row_id'] if active_cell else None
    print("Row table")
    div2=html.Div([
        html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
        ])
    print(active_cell)
    if active_cell is not None:
        print(active_cell['row'])
        single_df=csv_df.iloc[[active_cell['row']]]
        print(single_df)
        div2=html.Div([
            html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
            html.H6(f"Let's See the all the details of {single_df['App Name'].values[0]}",style={'text-
align':'center',}),
            html.Table([
            html.Tr([html.Td('App Id'),
            html.Td(f'{single_df["App Id"].values[0]}'),
            html.Td('Category'),
            html.Td(f'{single_df["Category"].values[0]}'),]),

            html.Tr([
                html.Td('Free'),
                html.Td(f'{single_df["Free"].values[0]}'),
                html.Td('Rating'),
                html.Td(f'{single_df["Rating"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Rating Count'),
                html.Td(f'{single_df["Rating Count"].values[0]}'),
                html.Td('Installs'),
                html.Td(f'{single_df["Installs"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Rating Count'),
                html.Td(f'{single_df["Rating Count"].values[0]}'),
                html.Td('Installs'),
                html.Td(f'{single_df["Installs"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Minimum Installs'),
                html.Td(f'{single_df["Minimum Installs"].values[0]}'),
                html.Td('Maximum Installs'),
                html.Td(f'{single_df["Maximum Installs"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Price'),
                html.Td(f'{single_df["Price"].values[0]}'),
                html.Td('Currency'),
                html.Td(f'{single_df["Currency"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Size'),
                html.Td(f'{single_df["Size"].values[0]}'),
                html.Td('Minimum Android'),
                html.Td(f'{single_df["Minimum Android"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Developer Id'),
                html.Td(f'{single_df["Developer Id"].values[0]}'),
                html.Td('Developer Website'),
                html.Td(html.A(children=f'{single_df["Developer
Website"].values[0]}',href=f'{single_df["Developer Website"].values[0]}',style={'color': 'blue', 'text-
decoration': 'none'}  )  ),
            ]),
            html.Tr([
                html.Td('Rating'),
                html.Td(f'{single_df["Rating"].values[0]}'),
                html.Td('Installs'),
                html.Td(f'{single_df["Installs"].values[0]}'),
            ]),
            html.Tr([
                html.Td('Developer Email'),
                html.Td(f'{single_df["Developer Email"].values[0]}'),
                html.Td('Released'),
                html.Td(f'{single_df["Released"].values[0]}'),
            ]),
            # html.Tr([
            #     html.Td('Last Updated'),
            #     html.Td(f'{single_df["Last Updated"].values[0]}'),
            #     html.Td('Content Rating'),
            #     html.Td(f'{single_df["Content Rating"].values[0]}'),
            # ]),
            html.Tr([
                html.Td('Privacy Policy'),
                html.Td(html.A(children=f'{single_df["Privacy
Policy"].values[0]}',href=f'{single_df["Privacy Policy"].values[0]}',style={'color': 'blue', 'text-
decoration': 'none'}   ) ),
                html.Td('Ad Supported'),
                html.Td(f'{single_df["Ad Supported"].values[0]}'),
            ]),
            html.Tr([
                html.Td('In App Purchases'),
                html.Td(f'{single_df["In App Purchases"].values[0]}'),
                html.Td('Editors Choice'),
                html.Td(f'{single_df["Editors Choice"].values[0]}'),
            ]),

            ],style={'margin':'0px auto','text-align':'center','padding-bottom':'30px'})

        ],)
    else:
        div2=html.Div([
        html.Hr(style={'border':'2px solid black','border-radius':'5px'}),
        ])

    if len(sort_by):
        dff=temp_df.sort_values(
            [col['column_id'] for col in sort_by],
            ascending=[
                col['direction'] == 'asc'
                for col in sort_by
            ],
            inplace=False
        )
    else:
        # No sort is applied
        dff = temp_df
    return dff.iloc[
        page_current*page_size:(page_current+ 1)*page_size
    ].to_dict('records'),div2
################# dash table #################

############ CSV download Code Blocks##########
@my_app.callback(Output('save_csv_promot','data'),
                Input('csv_download_Button','n_clicks'),
                prevent_initial_call=True,)
def save_csv(n_clicks):
    return dcc.send_data_frame(csv_df.to_csv,'filtered_csv_from_dash.csv')
############ CSV download Code Blocks##########
##################### callback for app list ##################



############################ Graph Visualization Layer #########################
app_graph_visualization_layer=html.Div([
    dcc.Dropdown(
        options=[
       {'label': 'Line-plot', 'value': 'lineplot'},
       {'label': 'Bar-plot ', 'value': 'barplot'},
       {'label': 'Count-plot', 'value': 'countplot'},
        {'label': 'Cat-plot', 'value': 'catplot'},
       {'label': 'Pie-chart', 'value': 'piechart'},
       {'label': 'Displot', 'value': 'displot'},
       {'label': 'Heatmap', 'value': 'heatmap'},
        {'label': 'Histogram and Distplot', 'value': 'histplot'},
        {'label': 'Scatter plot', 'value': 'scatter'},
       {'label': 'Multivariate Box plot', 'value': 'multiboxplot'},
       {'label': 'Violin plot', 'value': 'violinplot'},
   ],placeholder="Select Your Favorite Graph", searchable=False, id='graph_selector',style={'margin-
top':'20px'}
    ),
    html.Hr(style={'border':'2px solid black','border-radius':'5px'}),

    html.Div(id="graph_div")

])

############################ Graph Visualization Callback #########################
```

```
@my_app.callback(Output(component_id='graph_div',component_property='children'),
                [Input(component_id='graph_selector',component_property='value')])
def graph_selector(graph_selector):
    # print(play_store_apps_df)
    temp_year=pd.DataFrame()
    # play_store_apps_df[]
    play_store_apps_df['Released'] = pd.to_datetime(play_store_apps_df['Released'])
    play_store_apps_df['year']=play_store_apps_df['Released'].dt.year
    temp_year =
play_store_apps_df.year.value_counts(sort=True,ascending=False).rename_axis('year').reset_index(name='coun
ts')
    temp_year=temp_year.sort_values(by=['year'])

released_year=play_store_apps_df.groupby('year')['Installs'].sum().rename_axis('year').reset_index(name='I
nstalls Count')
    released_maximum=play_store_apps_df.groupby('year')['Maximum
Installs'].sum().rename_axis('year').reset_index(name='Maximum Installs Count')
    if graph_selector=="lineplot":
        line_fig1= px.line(temp_year, x="year", y="counts",title='Release year vs count of released apps')
        line_fig2= px.line(released_year, x="year", y="Installs Count",title='Release year vs count of
total Installations of Apps')
        line_fig3= px.line(released_maximum, x="year", y="Maximum Installs Count",title='Release year vs
count of Maximum Installations of Apps')
        line_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=line_fig1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=line_fig2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    dcc.Graph(figure=line_fig3)
            ]),
        ])
        return line_div
    elif graph_selector=="barplot":
        bar_fig1=px.bar(play_store_apps_df,x='Category', y='Installs',color='Content Rating',title='App
Installs Vs Categories by Content rating')
        bar_fig2=px.bar(play_store_apps_df,x='Category', y='Installs',color='Free',title='App Installs Vs
Categories by Free')
        bar_fig3=px.bar(play_store_apps_df,x='Category', y='Installs',color='In App Purchases',title='App
Installs Vs Categories by In App Purchases')
        bar_fig4=px.bar(play_store_apps_df,x='Category', y='Installs',color='Editors Choice',title='App
Installs Vs Categories by Editors Choice')
        # bar_fig1=px.bar(play_store_apps_df,x='Category', y='Installs')
        # bar_fig2=px.bar(play_store_apps_df,x='Category', y='Installs')
        # bar_fig3=px.bar(play_store_apps_df,x='Category', y='Installs')
        # bar_fig4=px.bar(play_store_apps_df,x='Category', y='Installs')
        bar_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=bar_fig1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=bar_fig2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=bar_fig3)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=bar_fig4)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
        ])
        return bar_div
    elif graph_selector=="countplot":
        minimum_androind=play_store_apps_df.groupby('Minimum Android').count().reset_index()
        minimum_androind=minimum_androind.rename(columns={"Developer Id":"Count"})
        category=play_store_apps_df.groupby('Category').count().reset_index()
        category=category.rename(columns={"Developer Id":"Count"})
        released_year=play_store_apps_df.groupby('year').count().reset_index()
        released_year=released_year.rename(columns={"Developer Id":"Count"})
        content_rating=play_store_apps_df.groupby('Content Rating').count().reset_index()
        content_rating=content_rating.rename(columns={"Developer Id":"Count"})
        count_plot1=px.bar(minimum_androind,x='Minimum Android',y='Count',title="Count of Minimum Android
Version",color_discrete_sequence=['chocolate']*len(minimum_androind))
        count_plot2=px.bar(category,x='Category',y='Count',title="Count of
Categories",color_discrete_sequence=['darkmagenta']*len(category))
        count_plot3=px.bar(released_year,x='year',y='Count',title="App releases over the
Years",color_discrete_sequence=['crimson']*len(released_year))
        count_plot4=px.bar(content_rating,x='Content Rating',y='Count',title="Count of Content
Rating",color_discrete_sequence=['coral']*len(content_rating))
        count_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=count_plot1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=count_plot2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=count_plot3)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=count_plot4)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
        ])
        return count_div
    elif graph_selector=="catplot":
        free=play_store_apps_df.groupby('Free').count().reset_index()
        free=free.rename(columns={"Developer Id":"Count"})
        editor_choice=play_store_apps_df.groupby('Editors Choice').count().reset_index()
        editor_choice=editor_choice.rename(columns={"Developer Id":"Count"})
        in_app_purchase=play_store_apps_df.groupby('In App Purchases').count().reset_index()
        in_app_purchase=in_app_purchase.rename(columns={"Developer Id":"Count"})
        ad_supported=play_store_apps_df.groupby('Ad Supported').count().reset_index()
        ad_supported=ad_supported.rename(columns={"Developer Id":"Count"})
        cat_plot1=px.bar(free,x='Free',y='Count',title="Free
Apps",color_discrete_sequence=['darkorange']*len(free))
        cat_plot2=px.bar(editor_choice,x='Editors Choice',y='Count',title="Editor's
Choice",color_discrete_sequence=['darksalmon']*len(editor_choice))
        cat_plot3=px.bar(in_app_purchase,x='In App Purchases',y='Count',title="In App
Purchase",color_discrete_sequence=['darkslateblue']*len(in_app_purchase))
        cat_plot4=px.bar(ad_supported,x='Ad Supported',y='Count',title="Ad
Supported",color_discrete_sequence=['darkcyan']*len(ad_supported))
        cat_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=cat_plot1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=cat_plot2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=cat_plot3)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=cat_plot4)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
        ])
        return cat_div
    elif graph_selector=="piechart":
        category=play_store_apps_df.groupby('Category').count().reset_index()
        category=category.rename(columns={"Developer Id":"Categories"})
        category['Price']=category['Price'].astype(float)
        category=category.loc[category.Price > 2000]
        # print(category.dtypes)
        content_rating=play_store_apps_df.groupby('Content Rating').count().reset_index()
        content_rating=content_rating.rename(columns={"Developer Id":"Content Ratings"})
        released_year=play_store_apps_df.groupby('year').count().reset_index()
        released_year=released_year.rename(columns={"Developer Id":"Years"})
        currency=play_store_apps_df.groupby('Free').count().reset_index()
        currency=currency.rename(columns={"Developer Id":"Frees"})
        # print(category)
        pie_chart1=px.pie(category,values='Categories',names='Category',title="App
Categories",color_discrete_sequence=px.colors.sequential.Reds)
        pie_chart2=px.pie(content_rating,values='Content Ratings',names='Content Rating',title="App
Content Ratings",color_discrete_sequence=px.colors.sequential.Darkmint)
        pie_chart3=px.pie(released_year,values='Years',names="year",title="App Released
Year",color_discrete_sequence=px.colors.sequential.Electric)
        pie_chart4=px.pie(currency,values='Frees',names='Free',title="Free or
Paid",color_discrete_sequence=px.colors.sequential.Burgyl)
        pie_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=pie_chart1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=pie_chart2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=pie_chart3)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=pie_chart4)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
        ])
        return pie_div
    elif graph_selector=="displot":
        # Price (Distribution of App Prices)
        # Installs (Distribution of App Installs)
        # Rating (Distribution of App Ratings)
        dis_fig1=px.histogram(play_store_apps_df,x='year',y='Price',color='Free',
hover_data=play_store_apps_df.columns,
        title="Distributions of App Prices Over the Year Based on App Free or not",opacity=0.5)
        dis_fig2=px.histogram(play_store_apps_df,x='year',y='Installs',color='Ad Supported',
        hover_data=play_store_apps_df.columns,title="Distributions of App Installs Over the Year Based on
App Ad Supported or not",opacity=0.5)
        dis_fig3=px.histogram(play_store_apps_df,x='year',y='Rating',color='In App Purchases',
        hover_data=play_store_apps_df.columns,title="Distributions of App Ratings Over the Year Based on
In App Purchases present or not",opacity=0.5)
        dis_fig4=px.histogram(play_store_apps_df,x='year',y='Rating Count',color='Editors Choice',
        hover_data=play_store_apps_df.columns,title="Distributions of App Rating Count Over the Year Based
on Editor's choice or not",opacity=0.5)
```

```
        #  dis_fig1=px.histogram(play_store_apps_df,x='year',y='Price',color='Free',marginal="box",
hover_data=play_store_apps_df.columns,
        # title="Distributions of App Prices Over the Year Based on App Free or not")
        # dis_fig2=px.histogram(play_store_apps_df,x='year',y='Installs',color='Ad
Supported',marginal="violin",
        # hover_data=play_store_apps_df.columns,title="Distributions of App Installs Over the Year Based
on App Ad Supported or not")
        # dis_fig3=px.histogram(play_store_apps_df,x='year',y='Rating',color='In App
Purchases',marginal="rug",
        # hover_data=play_store_apps_df.columns,title="Distributions of App Ratings Over the Year Based on
In App Purchases present or not")
        # dis_fig4=px.histogram(play_store_apps_df,x='year',y='Rating Count',color='Editors
Choice',marginal="violin",
        #  hover_data=play_store_apps_df.columns,title="Distributions of App Rating Count Over the Year
Based on Editor's choice or not")
        # dis_fig2=ff.create_distplot(hisdata,datalabels)
        dis_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=dis_fig1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=dis_fig2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=dis_fig3)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=dis_fig4)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
        ])

        return dis_div
    elif graph_selector=="heatmap":
        temp_df_heatmap=pd.DataFrame()
        temp_df_heatmap=play_store_apps_df.drop(['Unnamed: 0'],axis=1)
        # temp_df_heatmap=play_store_apps_df
        # print(temp_df_heatmap)
        mat_corr=temp_df_heatmap.corr()

        heat_fig1=px.imshow(mat_corr.round(2),text_auto=True,aspect="auto",title='Correlation Between the
Numeric Features')
        heat_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=heat_fig1)
                ]),

            ]),
            # html.Div([
            #         html.Div([
            #         # html.P("Release year vs count of released apps"),
            #         dcc.Graph(figure=heat_fig1)
            #     ],style={'width': '50%','display': 'inline-block'}),
            #     html.Div([
            #         dcc.Graph(figure=heat_fig1)
            #     ],style={'width': '50%','display': 'inline-block'}),
            # ]),
        ])
        return heat_div
    elif graph_selector=="histplot":
        # histgram_fig1=px.histogram(play_store_apps_df,x='Rating',title='Histogram of Ratings')
        # temp_price=play_store_apps_df[play_store_apps_df['Price']>0.0]
        # print(temp_price)
        histgram_fig2=px.histogram(play_store_apps_df,x='Rating',title='Histogram of 
Rating',color_discrete_sequence=['hotpink']*len(play_store_apps_df))
        histgram_fig4=px.histogram(play_store_apps_df,x='year',title='Histogram of 
Year',color_discrete_sequence=['indianred']*len(play_store_apps_df))
        # 
histgram_fig4=px.histogram(play_store_apps_df[play_store_apps_df['Price']>0.0],x='Price',title='Histogram 
of Price',color_discrete_sequence=['mediumorchid']*len(play_store_apps_df))
        hist_data11=play_store_apps_df['Rating']
        hist_data1=[hist_data11]
        group_labels1=['Rating']
        histgram_fig1=ff.create_distplot(hist_data=hist_data1,group_labels=group_labels1)
        histgram_fig1.layout.update({'title': 'Distribution Plot for Ratings'})

        hist_data12=play_store_apps_df['year']
        hist_data2=[hist_data12]
        group_labels2=['year']
        histgram_fig3=ff.create_distplot(hist_data=hist_data2,group_labels=group_labels2)
        histgram_fig3.layout.update({'title': 'Distribution Plot for year'})

        # hist_data13=play_store_apps_df['Installs']
        # hist_data3=[hist_data13]
        # group_labels3=['Installs']
        # histgram_fig3=ff.create_distplot(hist_data=hist_data3,group_labels=group_labels3)
        # histgram_fig3.layout.update({'title': 'Distribution Plot for Installs'})

        # hist_data14=play_store_apps_df['Price']
        # hist_data4=[hist_data14]
        # group_labels4=['Price']
        # histgram_fig4=ff.create_distplot(hist_data=hist_data4,group_labels=group_labels4)
        # histgram_fig4.layout.update({'title': 'Distribution Plot for Ratings Count'})


        histgram_div=html.Div([
        html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=histgram_fig1)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=histgram_fig2)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
            html.Div([
                    html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=histgram_fig3)
                ],style={'width': '50%','display': 'inline-block'}),
                html.Div([
                    dcc.Graph(figure=histgram_fig4)
                ],style={'width': '50%','display': 'inline-block'}),
            ]),
        ])
        return histgram_div
    elif graph_selector=="qqplot":
        # qqplot_fig1=qqplot(play_store_apps_df['Installs'])

        # qqplot_div=html.Div([
        #     html.Div([
        #         html.Div([
        #             # html.P("Release year vs count of released apps"),
        #             dcc.Graph(figure=qqplot_fig1)
        #         ],style={'width': '50%','display': 'inline-block'}),
        #         html.Div([
        #             dcc.Graph(figure=qqplot_fig1)
        #         ],style={'width': '50%','display': 'inline-block'}),
        #     ]),
        #     html.Div([
        #             html.Div([
        #             # html.P("Release year vs count of released apps"),
        #             dcc.Graph(figure=qqplot_fig1)
        #         ],style={'width': '50%','display': 'inline-block'}),
        #         html.Div([
        #             dcc.Graph(figure=qqplot_fig1)
        #         ],style={'width': '50%','display': 'inline-block'}),
        #     ]),
        # ])

        return html.P([f'{graph_selector} qqplot'])
    elif graph_selector=="kde":
        return html.P([f'{graph_selector} kde'])
    elif graph_selector=="scatter":
        # X = price Y=rating color = rating_count (Scatterplot between App Price and App Ratings by Rating
Counts)
        # X = price Y=installs color = rating_count  (Scatterplot between App Price and App Installs by
Rating Counts)
        sca_fig1=px.scatter(play_store_apps_df,x='Price',y='Rating',color='Rating Count',
trendline='ols',title='Scatterplot between App Price and App Ratings by Rating Counts')
        sca_fig2=px.scatter(play_store_apps_df,x='Price',y='Installs',color='Rating
Count',trendline='ols',title='Scatterplot between App Price and App Installs by Rating Counts')
        scatter_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=sca_fig1)
                ]),
                html.Div([
                    dcc.Graph(figure=sca_fig2)
                ]),
            ]),
        ])
        return scatter_div
    elif graph_selector=="multiboxplot":
        # Category, rating (Box Plot of App Categories by Ratings)
        # category , price (Box Plot of App Categories by Price)
        box_fig1=px.box(play_store_apps_df,x='Category',y='Rating',title='Box Plot of App Categories by
Ratings',color_discrete_sequence=['mediumpurple']*len(play_store_apps_df))
        box_fig2=px.box(play_store_apps_df,x='Content Rating',y='Rating',title='Box Plot of App Content
Rating by Ratings',color_discrete_sequence=['olivedrab']*len(play_store_apps_df))

        multibox_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=box_fig1)
                ]),
                html.Div([
                    dcc.Graph(figure=box_fig2)
                ]),
            ]),
        ])
        return multibox_div
    elif graph_selector=="areaplot":
        return html.P([f'{graph_selector} areaplot'])
    elif graph_selector=="violinplot":
        # Category, rating (Violin Plot of App Categories by Ratings)
        # category , price (Violin Plot of App Categories by Price)
        vio_fig1=px.violin(play_store_apps_df,x='Category',y='Rating',color='Ad Supported',title='Violin
Plot of App Categories and Ratings divide into Ad Supported or not')
        vio_fig2=px.violin(play_store_apps_df,x='Content Rating',y='Rating',color='In App
Purchases',title='Violin Plot of App Content Ratings and Ratings divide into In App Purchases or not')
        viloin_div=html.Div([
            html.Div([
                html.Div([
                    # html.P("Release year vs count of released apps"),
                    dcc.Graph(figure=vio_fig1)
                ]),
                html.Div([
                    dcc.Graph(figure=vio_fig2)
                ]),
            ]),
        ])

        return viloin_div


    # return html.P([f'{graph_selector}'])
############################ Graph Visualization Callback #########################


############################ Graph Visualization Layer #########################

################### Normality Test #######################
# trace=go.Histogram(
#     x=play_store_apps_df['Rating']
# )
# fig=iplot([trace],filename='JJJ')

# print(play_store_apps_df.dtypes)
app_graph_normality_layer=html.Div([
    html.Div([
        html.Div([
            html.P("Select the Column"),
            dcc.Dropdown(
                options=[
                    {'label': 'Installs', 'value': 'Installs'},
                    {'label': 'Rating Count', 'value': 'Rating Count'},
                    {'label': 'Rating', 'value': 'Rating'},
                    {'label': 'Minimum Installs', 'value': 'Minimum Installs'},
                    {'label': 'Maximum Installs ', 'value': 'Maximum Installs'},
                    {'label': 'Price', 'value': 'Price'},
                ],
                value='Rating',
                id='normal_dropdown_1'
            )
        ],style={'width': '45%','display': 'inline-block'}),
        html.Div([],style={'width': '10%','display': 'inline-block'}),
        html.Div([
            html.P("Select the Normality Test"),
            dcc.Dropdown(
                options=[
                    {'label': 'Quantile-Quantile Plot', 'value': 'qqplot'},
                    {'label': 'Shapiro-Wilk Test', 'value': 'shapiro'},
                    {'label': 'Kolmogorov-Smirnov (K-S) Test', 'value': 'anderson'},
                    {'label': "D'Agostino's Test", 'value': 'k2'},
                ],
                value='qqplot',
                id='normal_dropdown_2'
            )
        ],style={'width': '45%','display': 'inline-block'})

    ]),

    html.Div([

    ],id='Normality_div',style={'padding-top':'50px'})



])

########### normality callback #############
@my_app.callback(
        Output(component_id='Normality_div',component_property='children'),
        [Input(component_id='normal_dropdown_1',component_property='value'),
        Input(component_id='normal_dropdown_2',component_property='value')]
)
```

```
def normality_test(drop1,drop2):
    print(drop1)
    print(drop2)
    if (drop1 is None ) and (drop2 is None):
        return html.H3("Please Select Both of the options. If Any one of the dropdown is not selected then
it will not work",style={'text-align':'center'})
    else:
        if(drop2=='qqplot'):
            qqplot_data = qqplot(play_store_apps_df[drop1], line='s').gca().lines
            fig = go.Figure()

            fig.add_trace({
                'type': 'scatter',
                'x': qqplot_data[0].get_xdata(),
                'y': qqplot_data[0].get_ydata(),
                'mode': 'markers',
                'marker': {
                    'color': '#19d3f3'
                }
            })

            fig.add_trace({
                'type': 'scatter',
                'x': qqplot_data[1].get_xdata(),
                'y': qqplot_data[1].get_ydata(),
                'mode': 'lines',
                'line': {
                    'color': '#636efa'
                }

            })


            fig['layout'].update({
                'title': 'Quantile-Quantile Plot',
                'xaxis': {
                    'title': 'Theoritical Quantities',
                    'zeroline': False
                },
                'yaxis': {
                    'title': 'Sample Quantities'
                },
                'showlegend': False,
                'width': 800,
                'height': 700,
            })
            fig.update_layout(
                title=f"Quantile-Quantile Plot of {drop1}"
            )
            return dcc.Graph(figure=fig)
        elif drop2=='shapiro':
            stat,p=st.shapiro(play_store_apps_df[drop1])
            alpha=0.01
            if p > alpha:
                result='Sample looks Gaussian (fail to reject H0)'
            else:
                result='Sample doesnot look Gaussian (reject H0)'
            result_mat = [
            ['Length of the sample data', 'Test Statistic', 'p-value', 'Comments'],
            [len(play_store_apps_df), stat, p, result]
            ]

            dash_table_shapiro=dbc.Container([
                dbc.Label('Shapiro-Wilk Test',style={'text-align':'center'}),
                dash_table.DataTable(pd.DataFrame(result_mat).to_dict('result'),[{"name": i, "id": i} for
i in pd.DataFrame(result_mat).columns]),

            ])
            return dash_table_shapiro
        elif drop2=='anderson':
            stat,p=st.kstest(play_store_apps_df[drop1],'norm')
            alpha=0.01
            if p > alpha:
                result='Sample looks Gaussian (fail to reject H0)'
            else:
                result='Sample doesnot look Gaussian (reject H0)'
            result_mat = [
            ['Length of the sample data', 'Test Statistic', 'p-value', 'Comments'],
            [len(play_store_apps_df), stat, p, result]
            ]

            dash_table_ks=dbc.Container([
                dbc.Label('Kolmogorov-Smirnov (K-S) Test',style={'text-align':'center'}),
                dash_table.DataTable(pd.DataFrame(result_mat).to_dict('result'),[{"name": i, "id": i} for
i in pd.DataFrame(result_mat).columns]),

            ])
            return dash_table_ks
        elif drop2=='k2':
            stat,p=st.normaltest(play_store_apps_df[drop1])
            alpha=0.01
            if p > alpha:
                result='Sample looks Gaussian (fail to reject H0)'
            else:
                result='Sample doesnot look Gaussian (reject H0)'
            result_mat = [
            ['Length of the sample data', 'Test Statistic', 'p-value', 'Comments'],
            [len(play_store_apps_df), stat, p, result]
            ]

            dash_table_k2=dbc.Container([
                dbc.Label("D'Agostino's Test",style={'text-align':'center'}),
                dash_table.DataTable(pd.DataFrame(result_mat).to_dict('result'),[{"name": i, "id": i} for
i in pd.DataFrame(result_mat).columns]),

            ])
            return dash_table_k2

################### Normality Test #######################
############################# Code Blocks for Tabs ################################
@my_app.callback(Output(component_id='layout',component_property='children'),
                [Input(component_id='project',component_property='value')])

def update_layout(tabs):
    if tabs == 'app':
        return app_statistics_layout
    elif tabs == 'compare':
        return app_comparision_layout
    elif tabs=='details':
        return app_details_layout
    elif tabs=='graph':
        return app_graph_visualization_layer
    elif tabs=='normality':
        return app_graph_normality_layer

############################ Code Blocks for Tabs ################################
my_app.server.run(debug = 'True')
#%%
```