***Application Design for***

***an app launch on Google PlayStore***



**Submitted by : -** *Amurto Basu*

**Date of Submission: 10th July 2019.**

**Submitted to : \_CYBERACE INFOVISION PVT LTD**

**Under the Guidance of : Junaid Khateeb (Director, Khateeb Insitute of Technical Education)**

**Certificate Of completion**

**This is to certify that , Mr. Amurto Basu has successfully implemented an application designed to study the data and generate insights for an app launch on Google PlayStore.**

**The Application has been accepted as a completed project as it meets all the requirements specified.**

**12th July 2019**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**(Khateeb Institute of Technical Education)**

**Acknowledgements :**

I would like to express my sincere gratitude to my Computer Engineering faculty for providing their invaluable guidance, comments and suggestions throughout the course of the project. I would specially thank Professor Kalpana for constantly motivating me to work harder and providing right guidance for our project . Also I would like to thank Dr B.S Daga for his assistance in the ML Data Visualization .**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SR.NO** | **TOPIC** | **PAGE NO.** |
| **1** | **SPECIFICATIONS** | **5** |
| **2** | **TECHNOLOGY USED** | **5** |
| **3** | **DATABASE SPECIFICATIONS** | **13** |
| **4** | **SCREENSHOTS** | **14** |
| **5** | **VALIDATIONS** | **44** |
| **6** | **SOURCE CODE** | **45** |

**System Requirement Specifications:**

We are given a dataset of A leading software development and distribution firm ,that helps individuals /Firms in designing and launching comercially viable mobile apps. They conducted a survey of the user download behaviour on the apps across all categories on the google play store. They also have pursued the user review for the contents that the users have downloaded and used over a period of time. The Company aims to take crucial decisions before their future launches on

playstore. Therefore as a research analysts to provide appropriate results we have used following

modules:

**Technology used:**

As a research analysts to provide appropriate results we have used following modules:

1)**Pandas**- **pandas** is a **software library** written for the for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and [time series](https://en.wikipedia.org/wiki/Time_series)

Library features

* DataFrame object for data manipulation with integrated indexing.
* Tools for reading and writing data between in-memory data structures and different file formats.
* Data alignment and integrated handling of missing data.
* Reshaping and pivoting of data sets.
* Label-based slicing, fancy indexing, and subsetting of large data sets.
* Data structure column insertion and deletion.

2) **Matplotlib**-

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

Conventionally, the package is imported into the Python script by adding the following statement −

from matplotlib import pyplot as plt

3**)Seaborn-**Seaborn is a library for making statistical infographics in Python. It is built on top of matplotlib and also supports numpy and pandas data structures. It also supports statistical units from Scipy

Visualization plays an important role when we try to explore and understand data, Seaborn is aimed to make it easier and centre of the process. To put in perspective, if we say matplotlib makes things easier and hard things possible, seaborn tries to make that hard easy too, that too in a well-defined way. But seaborn is not an alternative to matplotlib, think of it as a complement to the previous.

As it is built on top of matplotlib, we will often invoke matplotlib functions directly for simple plots at matplotlib has already created highly efficient programs for it.

The high-level interface of seaborn and customizability and variety of backends for matplotlib combined together makes it easy to generate publication-quality figures.

4)plotly-plotly allows users to import, copy and paste, or stream data to be analyzed and visualized. For analysis and styling graphs, Plotly offers a Python sandbox (numpy supported), datagrid, and GUI. Python scripts can be saved, shared, and collaboratively edited in Plotly.

Plotly Offline allows you to create graphs offline and save them locally. There are also two methods for plotting offline: plotly.offline.plot().and plotly.offline.iplot().

* Use plotly.offline.plot() to create and standalone HTML that is saved locally and opened inside your web browser.

5)datetime- In Python, **date, time and datetime**classes provides a number of function to deal with dates, times and time intervals. Date and datetime are an object in Python, so when we manipulate them, we are actually manipulating objects and not string or timestamps. To manipulate dates or time, we need to import datetime function.

The datetime classes in Python are categorized into main 5 classes.

* date – Manipulate just date ( Month, day, year)
* time – Time independent of the day (Hour, minute, second, microsecond)
* datetime – Combination of time and date (Month, day, year, hour, second, microsecond)
* timedelta— A duration of time used for manipulating dates
* tzinfo— An abstract class for dealing with time zones

6) Python Gui Tkinter-

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.  
**To create a tkinter:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

Importing tkinter is same as importing any other module in the python code. Note that the name of the module in Python 2.x is ‘Tkinter’ and in Python 3.x is ‘tkinter’.

import tkinter

There are two main methods used you the user need to remember while creating the Python application with GUI.

1. **Tk(screenName=None,  baseName=None,  className=’Tk’,  useTk=1):** To create a main window, tkinter offers a method ‘Tk(screenName=None,  baseName=None,  className=’Tk’,  useTk=1)’. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

m=tkinter.Tk() where m is the name of the main window object

1. **mainloop():** There is a method known by the name mainloop() is used when you are ready for the application to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event till the window is not closed.

m.mainloop()

we are provided with two csv files,containing the results of the survey carried out by the companys.The dataset consists of 10842 entries with columns such as(Name,category,ratings,reviews,size,installs,type,price,content rating,price,genres,last updated,current version ,Android version).All these factors affect the success of the apps.

**Using the modules mentioned above we have analysed:**

a)Percentage download in each category on the playstore using piechart in plotly module

b)Apps that have managed to get downloads between some range such as(10,000 and 50,000, 50,000 and 150000, 150000 and 500000, 150000 and 500000, More than 5000000)using cut function which converts downloads to groups of downloads ranges.

c) category of apps have managed to get the most,least and an average of 2,50,000 downloads atleast by calculating the average of the installs(using sum() and len() functions)

d) category of apps have managed to get the highest maximum average ratings from the users

e) The download trends of the apps category wise by extracting the year from the Last updated column(using datetime module) in the dataset and plotting using plotly.

f)The downloads in the year 2016,2017,2018 by extracting the years form the last updated attribute and calculating the maximum amd minimum downloads in the above years by using (max and min functions.)

g) Percentage change in the downloads of the apps over the years using pct\_change() function of pandas module

h) The percentage change in the downloads of the apps that can work with varying devices (by truncating the varies with device entry in thr last updated column in the datframe )

i) Across all the years ,the month that has seen the maximum downloads in each of the category.(by extracting month and sorting them)

h)prediction of download trend of the apps category such as sports, entertainment,social media,news,events,travel and games using trend forecasting modules ***Arima(AutoRegressive Integrated Moving Averages )*** and ***Sarimax(Seasonal AutoRegressive Integrated Moving Averages with eXogenous regressors)***

ARIMA is a model that can be fitted to time series data in order to better understand or predict future points in the series.

There are three distinct integers (p, d, q) that are used to parametrize ARIMA models. Because of that, ARIMA models are denoted with the notation ARIMA(p, d, q). Together these three parameters account for seasonality, trend, and noise in datasets:

* p is the *auto-regressive* part of the model. It allows us to incorporate the effect of past values into our model. Intuitively, this would be similar to stating that it is likely to be warm tomorrow if it has been warm the past 3 days.
* d is the *integrated* part of the model. This includes terms in the model that incorporate the amount of differencing (i.e. the number of past time points to subtract from the current value) to apply to the time series. Intuitively, this would be similar to stating that it is likely to be same temperature tomorrow if the difference in temperature in the last three days has been very small.
* q is the *moving average* part of the model. This allows us to set the error of our model as a linear combination of the error values observed at previous time points in the past.

When dealing with seasonal effects, we make use of the *seasonal* ARIMA, which is denoted as ARIMA(p,d,q)(P,D,Q)s. Here, (p, d, q) are the non-seasonal parameters described above, while (P, D, Q) follow the same definition but are applied to the seasonal component of the time series. The term s is the periodicity of the time series (4 for quarterly periods, 12 for yearly periods, etc.).

The seasonal ARIMA method can appear daunting because of the multiple tuning parameters involved. In the next section, we will describe how to automate the process of identifying the optimal set of parameters for the seasonal ARIMA time series model.

When looking to fit time series data with a seasonal ARIMA model, our first goal is to find the values of ARIMA(p,d,q)(P,D,Q)s that optimize a metric of interest. There are many guidelines and best practices to achieve this goal, yet the correct parametrization of ARIMA models can be a painstaking manual process that requires domain expertise and time. Other statistical programming languages such as R provide [automated ways to solve this issue](https://www.rdocumentation.org/packages/forecast/versions/7.3/topics/auto.arima), but those have yet to be ported over to Python. In this section, we will resolve this issue by writing Python code to programmatically select the optimal parameter values for our ARIMA(p,d,q)(P,D,Q)s time series model.

We use a "grid search" to iteratively explore different combinations of parameters. For each combination of parameters, we fit a new seasonal ARIMA model with the SARIMAX() function from the statsmodels module and assess its overall quality. Once we have explored the entire landscape of parameters, our optimal set of parameters will be the one that yields the best performance for our criteria of interest. Let's begin by generating the various combination of parameters that we wish to assess:

We use the triplets of parameters defined above to automate the process of training and evaluating ARIMA models on different combinations. In Statistics and Machine Learning, this process is known as grid search (or hyperparameter optimization) for model selection.

When evaluating and comparing statistical models fitted with different parameters, each can be ranked against one another based on how well it fits the data or its ability to accurately predict future data points. We will use the AIC (Akaike Information Criterion) value, which is conveniently returned with ARIMA models fitted using statsmodels. The AIC measures how well a model fits the data while taking into account the overall complexity of the model. A model that fits the data very well while using lots of features will be assigned a larger AIC score than a model that uses fewer features to achieve the same goodness-of-fit. Therefore, we are interested in finding the model that yields the lowest AIC value.

The code chunk below iterates through combinations of parameters and uses the SARIMAXfunction from statsmodels to fit the corresponding Seasonal ARIMA model. Here, the orderargument specifies the (p, d, q) parameters, while the seasonal\_order argument specifies the (P, D, Q, S) seasonal component of the Seasonal ARIMA model. After fitting each SARIMAX()model, the code prints out its respective AIC score.

Using grid search, we have identified the set of parameters that produces the best fitting model to our time series data. We can proceed to analyze this particular model in more depth.

We'll start by plugging the optimal parameter values into a new SARIMAX model:

mod = sm.tsa.statespace.SARIMAX(y,

order=(1, 1, 1),

seasonal\_order=(1, 1, 1, 12),

enforce\_stationarity=**False**,

enforce\_invertibility=**False**)

results = mod.fit()

print(results.summary().tables[1])

Output

==============================================================================

coef std err z P>|z| [0.025 0.975]

------------------------------------------------------------------------------

ar.L1 0.3182 0.092 3.443 0.001 0.137 0.499

ma.L1 -0.6255 0.077 -8.165 0.000 -0.776 -0.475

ar.S.L12 0.0010 0.001 1.732 0.083 -0.000 0.002

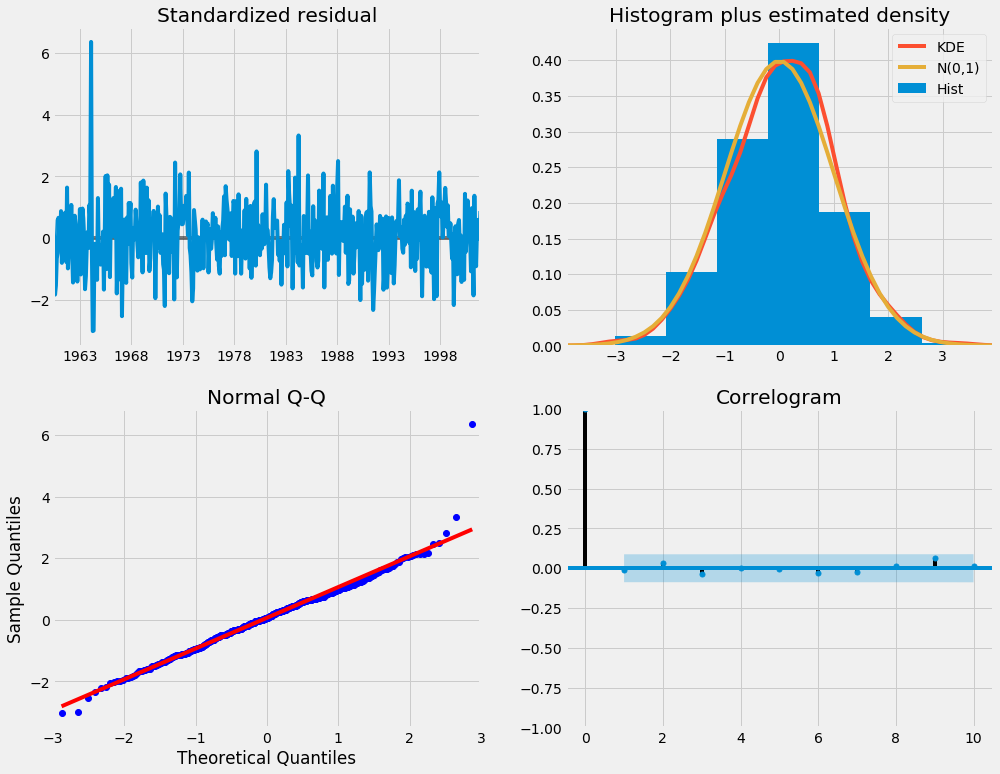
ma.S.L12 -0.8769 0.026 -33.811 0.000 -0.928 -0.826

sigma2 0.0972 0.004 22.634 0.000 0.089 0.106

==============================================================================

The summary attribute that results from the output of SARIMAX returns a significant amount of information, but we'll focus our attention on the table of coefficients. The coef column shows the weight (i.e. importance) of each feature and how each one impacts the time series. The P>|z|column informs us of the significance of each feature weight. Here, each weight has a p-value lower or close to 0.05, so it is reasonable to retain all of them in our model.

When fitting seasonal ARIMA models (and any other models for that matter), it is important to run model diagnostics to ensure that none of the assumptions made by the model have been violated. The plot\_diagnostics object allows us to quickly generate model diagnostics and investigate for any unusual behavior.



Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved.

In this case, our model diagnostics suggests that the model residuals are normally distributed based on the following:

* In the top right plot, we see that the red KDE line follows closely with the N(0,1) line (whereN(0,1)) is the standard notation for a normal distribution with mean 0 and standard deviation of 1). This is a good indication that the residuals are normally distributed.
* The [qq-plot](https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot) on the bottom left shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). Again, this is a strong indication that the residuals are normally distributed.
* The residuals over time (top left plot) don't display any obvious seasonality and appear to be white noise. This is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom right, which shows that the time series residuals have low correlation with lagged versions of itself.
* The get\_prediction() and conf\_int() attributes allow us to obtain the values and associated confidence intervals for forecasts of the time series.
* pred = results.get\_prediction(start=pd.to\_datetime('1998-01-01'), dynamic=**False**)
* pred\_ci = pred.conf\_int()

j) Sentiment Analysis of the apps using Textblob module. Sentiment analysis is basically the process of determining the attitude or the emotion of the writer, i.e., whether it is positive or negative or neutral.

The *sentiment* function of textblob returns two properties, **polarity**, and **subjectivity**. poarity is float which lies in the range of [-1,1] where 1 means positive statement and -1 means a negative statement. Subjective sentences generally refer to personal opinion, emotion or judgment whereas objective refers to factual information. Subjectivity is also a float which lies in the range of [0,1].We have also used the text translation function in textblob to analyse the reviews which are in other languages. TextBlob is an open source text processing library. It is built on top of Natural Language ToolKit (NLTK) and Pattern libraries and provides easy-to-use functions that simplifies text processing tasks. It has an advantage of being faster than NLTK. In the last post we have looked at text procblob. TextBlob integrates well with other Natural Language Processing (NLP) tasks such as Part-of-Speech tagging, sentiment analysis, text classification, language translation among others.Using textBlob translation function we can detect the language using blob.detect\_language() and blob.translate(from\_lang='ar', to ='en') from one language to other language

**Database Software used:**

The database software used is pymysql

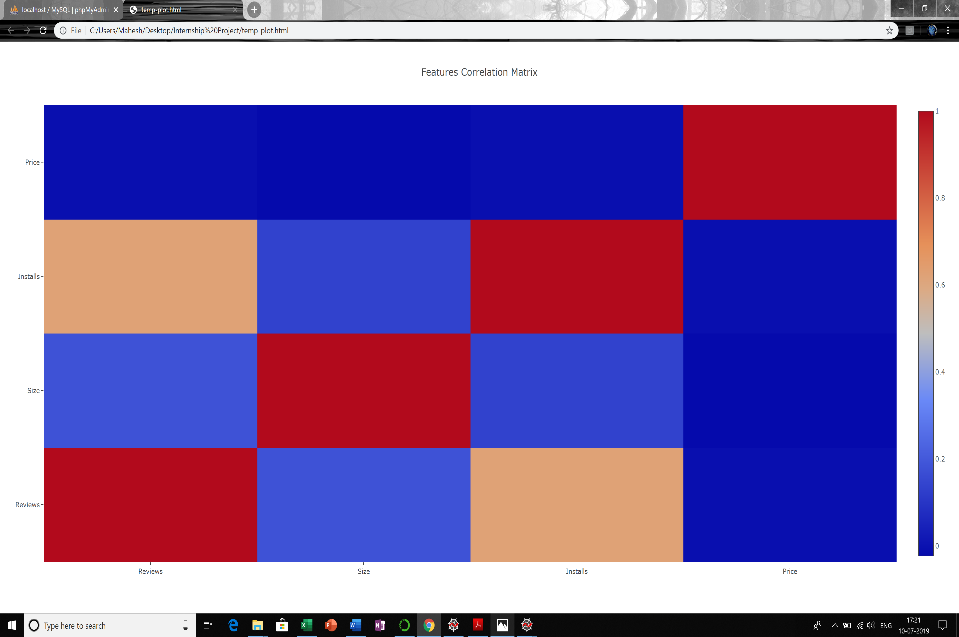
# pymysql

A simple database interface forpython that builds on top FreeTDs to provide a Python DB-API (pep\_249)interface to Microsoft SQL Server.

## Features

* Unicode friendly
* Python 3 friendly
* Works on most popular operating systems
* Written in Cython for performance
* Includes a supported and documented low-level module (mssql) that you can use instead of the DB-API
* Supports stored procedures with both return values and output parameters
* A comprehensive test suite
* Compatible with cooperative multi-tasking systems (gevent, etc.)
* Can be used to connect to Azure

***Screenshots*:-**



apps=common()

corrmat = apps.corr()

trace1 = {"x": corrmat.columns, "y": corrmat.columns,

"z": corrmat.values,

"type": "heatmap"

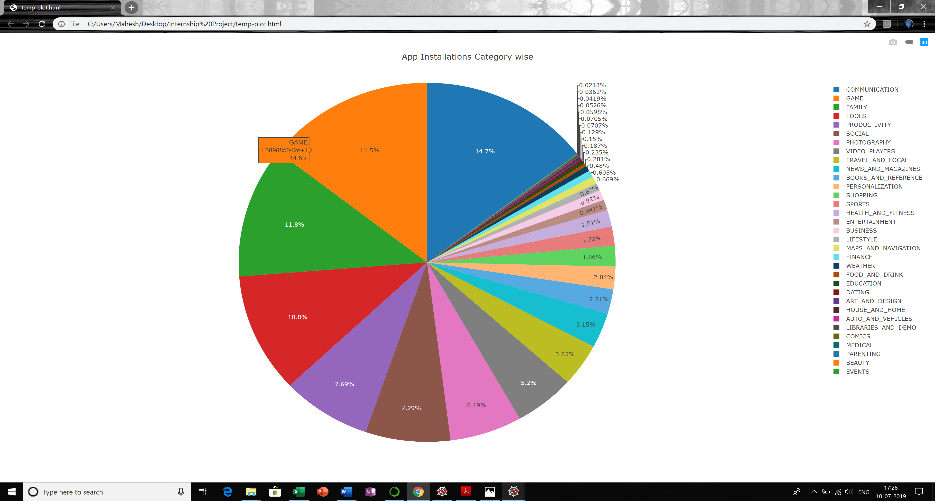
}

data = [trace1]

layout = go.Layout(title='Features Correlation Matrix')

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig)

1. 

apps=common()

category\_count=apps['Category'].tolist()

installs\_count=apps['Installs'].tolist()

category=apps['Category'].unique()

downloads=list()

for i in category:

dl=0

for j,k in zip(category\_count,installs\_count):

if j==i:

dl=dl+k

downloads.append(dl)

fig = {'data': [{'labels': category,

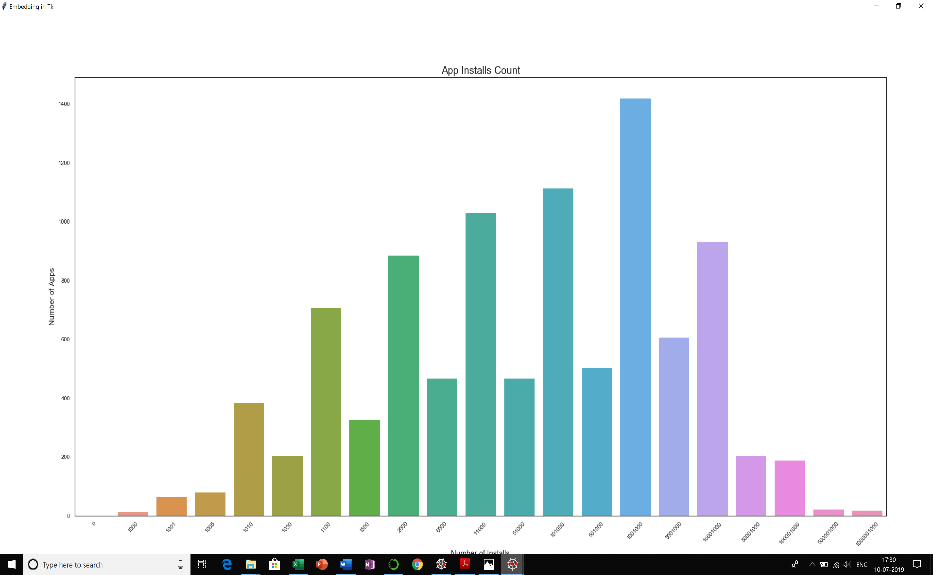
'values': downloads,

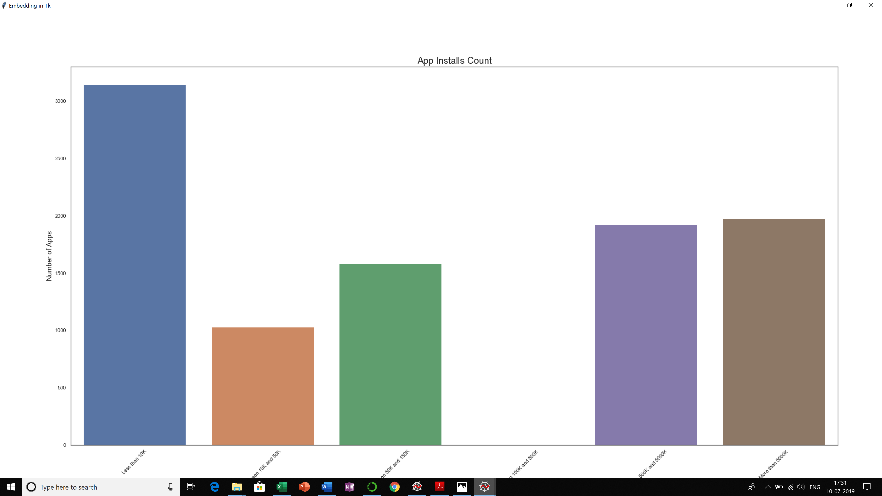
'type': 'pie'}],

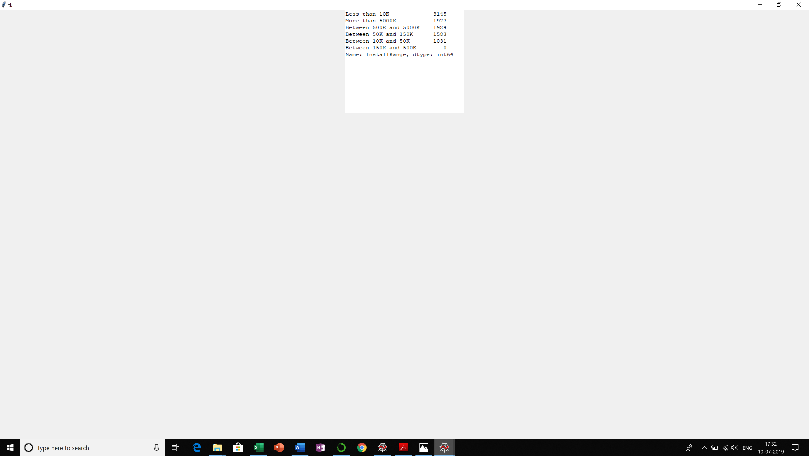
'layout': {'title': 'App Installations Category wise'}

}

plotly.offline.plot(fig)

1. 





apps=common()

sns.set(style="white")

f=plt.figure(figsize = (14,12))

g1 = sns.countplot(x='Installs', data=apps)

g1.set\_title("App Installs Count", fontsize=20)

g1.set\_xlabel("Number of Installs", fontsize=15)

g1.set\_ylabel("Number of Apps", fontsize=15)

plt.xticks(rotation=45)

root = tkinter.Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f,root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

apps=common()

sns.set(style="white")

installs\_categorical=pd.DataFrame({'InstallRange': apps['Installs']})

installs\_categorical=pd.cut(installs\_categorical['InstallRange'], bins=[0, 10000, 50000, 150000, 500000, 5000000, 999999999999], include\_lowest=True, labels=['Less than 10K','Between 10K and 50K','Between 50K and 150K','Between 150K and 500K','Between 500K and 5000K','More than 5000K'])

apps=pd.concat([apps,installs\_categorical],axis=1)

f2=plt.figure(figsize = (14,12))

g1 = sns.countplot(x='InstallRange', data=apps)

g1.set\_title("App Installs Count", fontsize=20)

g1.set\_xlabel("Number of Installs", fontsize=15)

g1.set\_ylabel("Number of Apps", fontsize=15)

plt.xticks(rotation=45)

root = tkinter.Tk()

root.wm\_title("Embedding in Tk")

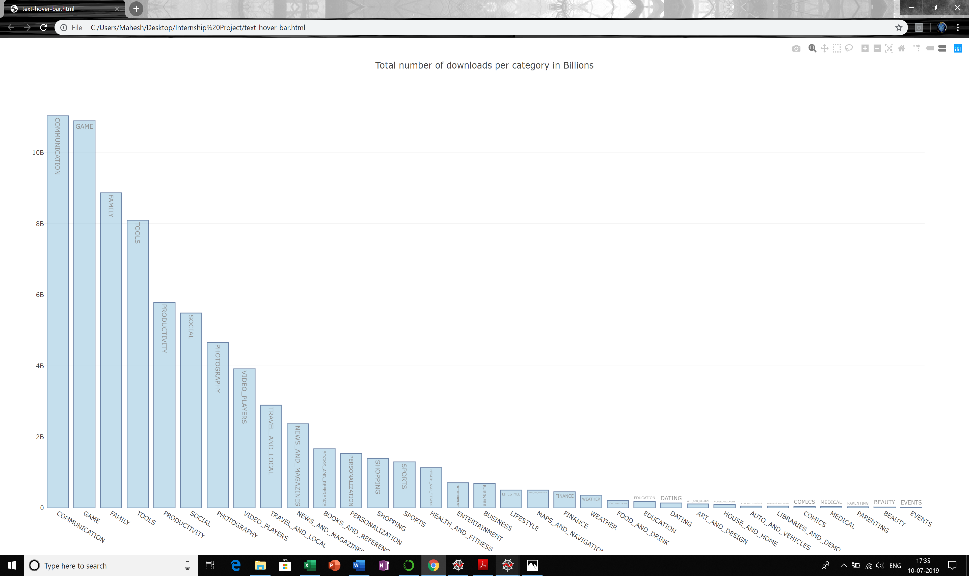
root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

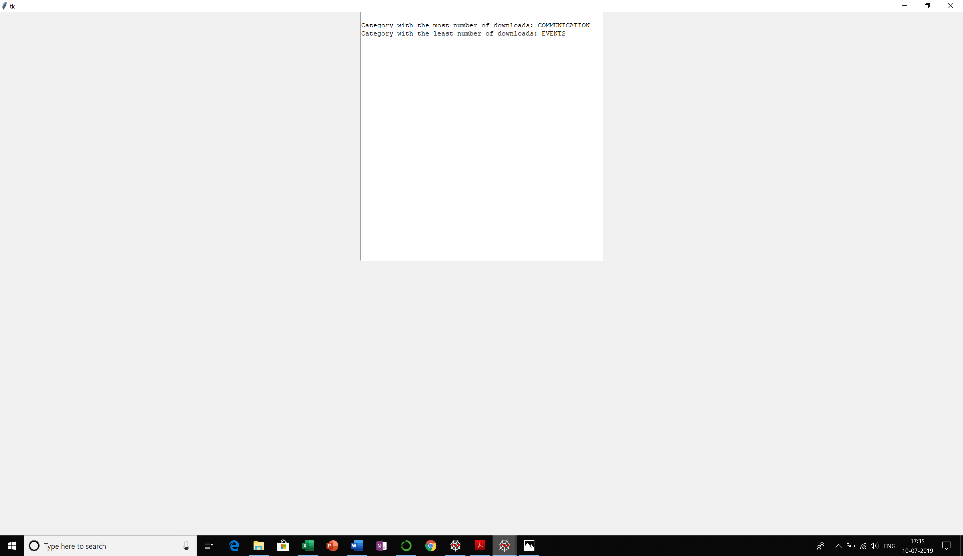
canvas = FigureCanvasTkAgg(f2,root) # A tk.DrawingArea.

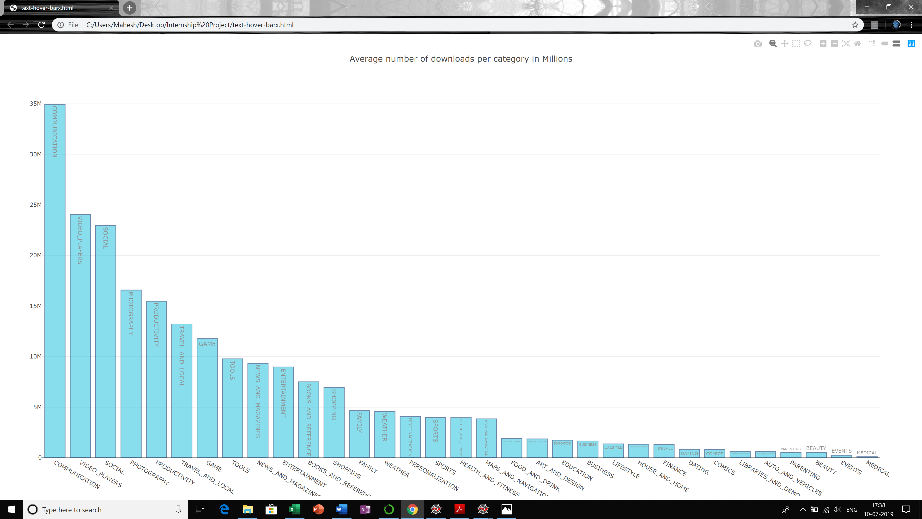
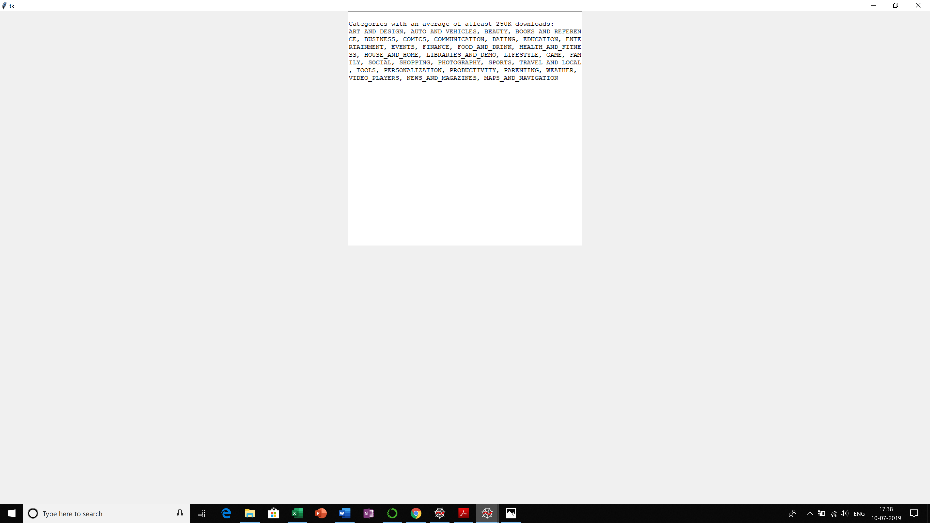
canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

1. 



apps=common()

cat\_list = list(apps.Category.unique())

dls = []

mldl = []

names=[]

for category in cat\_list:

x = apps[apps.Category == category]

dl\_count = x.Installs.sum()//len(x)

maxleast\_count=x.Installs.sum()

mldl.append(maxleast\_count)

dls.append(dl\_count)

if dl\_count>=250000:

names.append(category)

ml=pd.DataFrame({'Category':cat\_list, 'Downloads':mldl})

new\_index = (ml['Downloads'].sort\_values(ascending=False)).index.values

sorted\_data = ml.reindex(new\_index)

print(sorted\_data['Category'].iloc[0]," Category has the most number of downloads")

print(sorted\_data['Category'].iloc[-1]," Category has the least number of downloads")

trace1 = go.Bar(x=sorted\_data.Category,y=sorted\_data.Downloads,text=sorted\_data.Category,textposition='auto',marker=dict(color='rgb(158,202,225)',line=dict(color='rgb(8,48,107)',width=1.5,)),opacity=0.6)

data = [trace1]

layout = go.Layout(title='Total number of downloads per category in Billions',)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='text-hover-bar.html')

root = tk.Tk()

T = tk.Text(root, height=30, width=60)

T.pack()

T.insert(tk.END, "\nCategory with the most number of downloads: ")

T.insert(tk.END, sorted\_data['Category'].iloc[0],"\n")

T.insert(tk.END, "\nCategory with the least number of downloads: ")

T.insert(tk.END, sorted\_data['Category'].iloc[-1])

tk.mainloop()

apps=common()

cat\_list = list(apps.Category.unique())

dls = []

mldl = []

names=[]

for category in cat\_list:

x = apps[apps.Category == category]

dl\_count = x.Installs.sum()//len(x)

maxleast\_count=x.Installs.sum()

mldl.append(maxleast\_count)

dls.append(dl\_count)

if dl\_count>=250000:

names.append(category)

data=pd.DataFrame({'Category':cat\_list, 'Downloads':dls})

new\_index = (data['Downloads'].sort\_values(ascending=False)).index.values

sorted\_data = data.reindex(new\_index)

trace0 = go.Bar(x=sorted\_data.Category,y=sorted\_data.Downloads,text=sorted\_data.Category,textposition='auto',marker=dict(color='rgb(58,200,225)',line=dict(color='rgb(8,48,107)',width=1.5,)),opacity=0.6)

data = [trace0]

layout = go.Layout(title='Average number of downloads per category in Millions',)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='text-hover-barx.html')

root = tk.Tk()

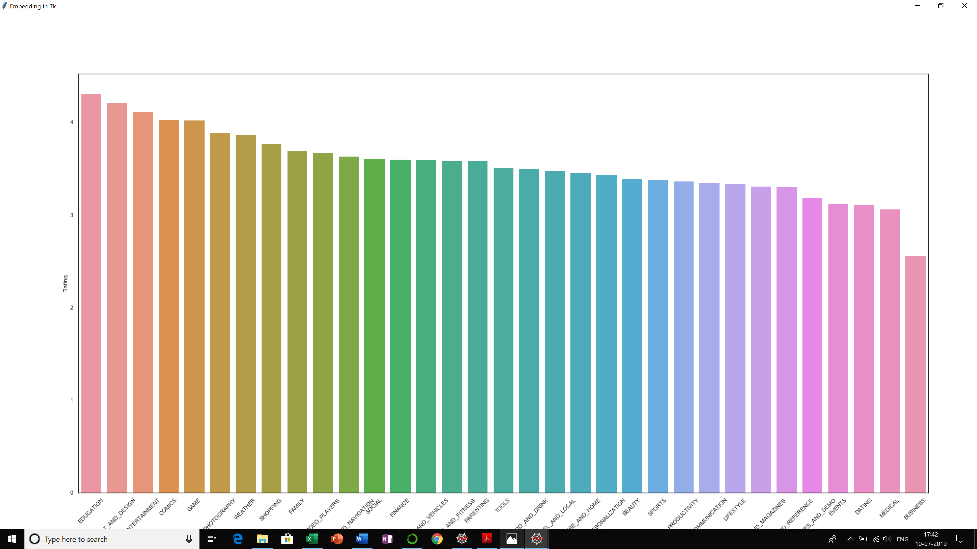
T = tk.Text(root, height=30, width=60)

T.pack()

T.insert(tk.END, "\nCategories with an average of atleast 250K downloads: \n")

T.insert(tk.END,', '.join(names),"\n")

tk.mainloop()

1. 

apps=common()

category\_list = list(apps.Category.unique())

ratings = []

for category in category\_list:

x = apps[apps.Category == category]

rating\_rate = x.Rating.sum()/len(x)

ratings.append(rating\_rate)

data=pd.DataFrame({'Category':category\_list, 'Rating':ratings})

new\_index = (data['Rating'].sort\_values(ascending=False)).index.values

sorted\_data = data.reindex(new\_index)

sns.set(style="white")

# Set up the matplotlib figure

f, ax = plt.subplots(figsize=(15, 9))

sns.barplot(x=sorted\_data.Category, y=sorted\_data.Rating)

plt.xticks(rotation=45)

plt.show()

root = tkinter.Tk()

root.wm\_title("Embedding in Tk")

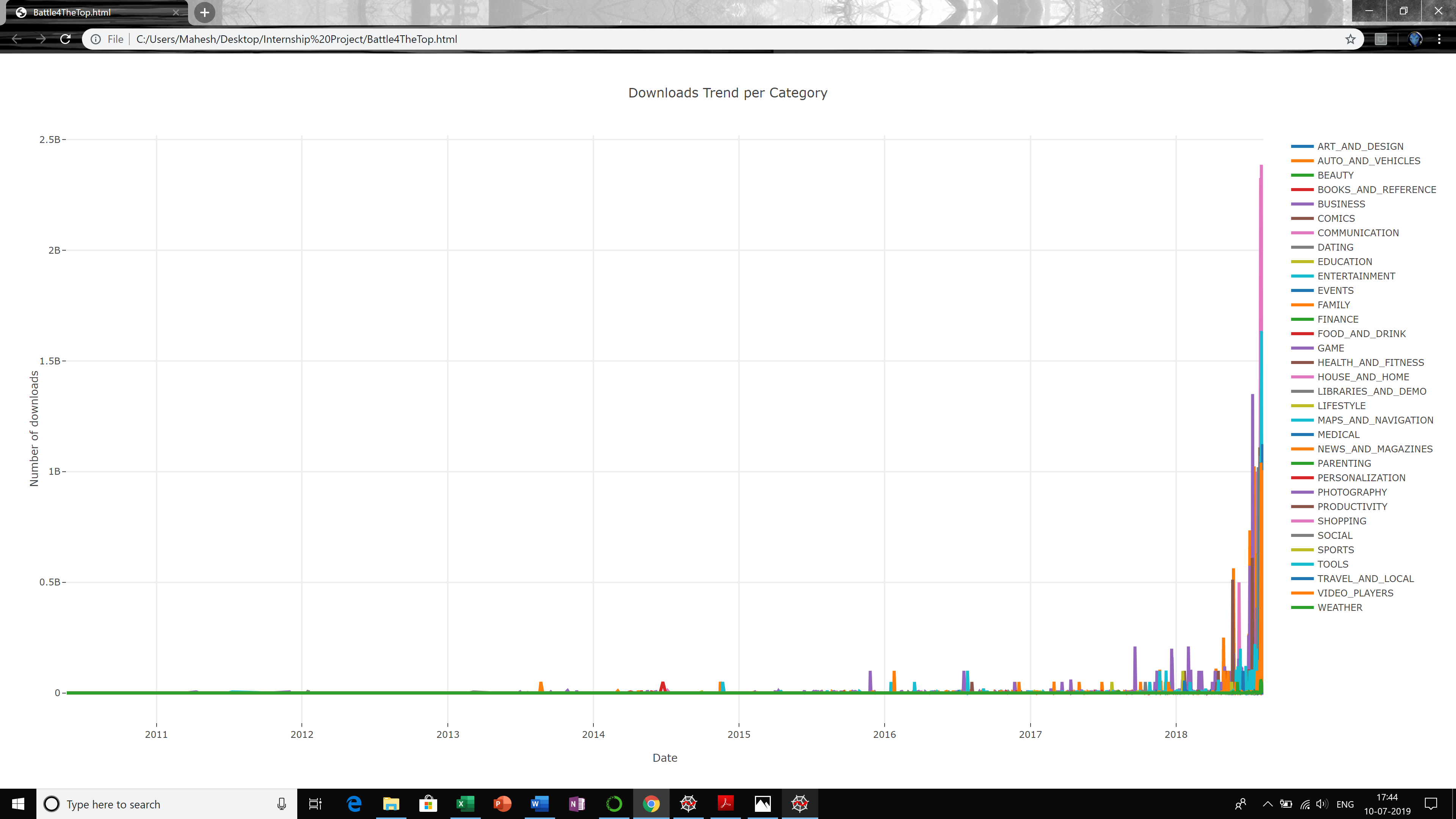
root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f, root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

1. 

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Date']).aggregate(aggregation\_functions).reset\_index()

data.sort\_values(by=['Date'],inplace=True)

data=data.pivot(index='Date', columns='Category', values='Installs')

data=data.fillna(0)

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Date', ticklen=5, zeroline=False, gridwidth=2),

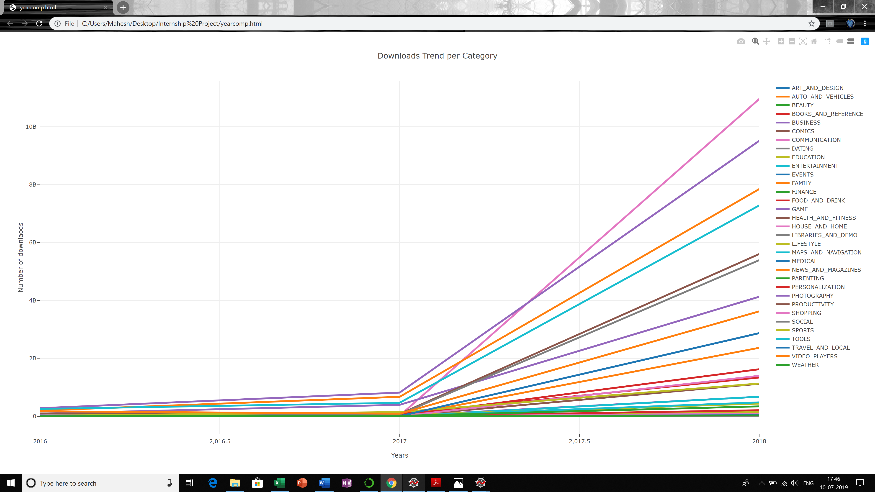
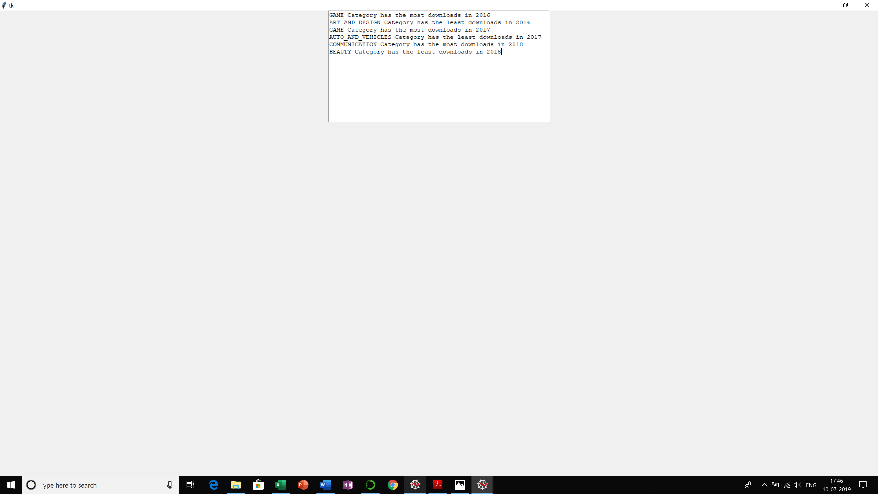
yaxis=dict(title='Number of downloads', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='Battle4TheTop.html')

1.   

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Year"}, inplace=True)

data['Year'] = pd.DatetimeIndex(data['Year']).year

data=data[data.Year > 2015]

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Year']).aggregate(aggregation\_functions).reset\_index()

data.sort\_values(by=['Year'],inplace=True)

data=data.pivot(index='Year', columns='Category', values='Installs')

data=data.fillna(1)

category=data.columns.tolist()

d2016=data.values[0].tolist()

d2017=data.values[0].tolist()

d2018=data.values[2].tolist()

dict2016 = dict(zip(category, data.values[0].tolist()))

maximum1 = max(dict2016, key=dict2016.get)

a=maximum1+" Category has the most downloads in 2016"

minimum1 = min(dict2016, key=dict2016.get)

b=minimum1+" Category has the least downloads in 2016"

dict2017 = dict(zip(category, data.values[1].tolist()))

maximum2 = max(dict2017, key=dict2017.get)

c=maximum2+" Category has the most downloads in 2017"

minimum2 = min(dict2017, key=dict2017.get)

d=minimum2+" Category has the least downloads in 2017"

dict2018 = dict(zip(category, data.values[2].tolist()))

maximum3 = max(dict2018, key=dict2018.get)

e=maximum3+" Category has the most downloads in 2018"

minimum3 = min(dict2018, key=dict2018.get)

f=minimum3+" Category has the least downloads in 2018"

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Years', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downlaods', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='yearcomp.html')

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,a)

T.insert(tk.END,"\n")

T.insert(tk.END,b)

T.insert(tk.END,"\n")

T.insert(tk.END,c)

T.insert(tk.END,"\n")

T.insert(tk.END,d)

T.insert(tk.END,"\n")

T.insert(tk.END,e)

T.insert(tk.END,"\n")

T.insert(tk.END,f)

root.mainloop()

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Year"}, inplace=True)

data['Year'] = data['Year'].apply(lambda x: x.replace('2017','2018') if '2017' in str(x) else x)

data['Year'] = pd.DatetimeIndex(data['Year']).year

data=data[data.Year > 2015]

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Year']).aggregate(aggregation\_functions).reset\_index()

data.sort\_values(by=['Year'],inplace=True)

data=data.pivot(index='Year', columns='Category', values='Installs')

data=data.fillna(1)

data=data.pct\_change()

category=data.columns.tolist()

pct=data.values[1].tolist()

pct = [i \* 100 for i in pct]

root=Tk()

T = tk.Text(root, height=100, width=150)

T.pack()

for i in range(len(category)):

T.insert(tk.END, category[i])

T.insert(tk.END, " : ")

if pct[i]<0:

T.insert(tk.END, "-")

T.insert(tk.END, pct[i])

T.insert(tk.END, "%")

T.insert(tk.END, "\n")

else:

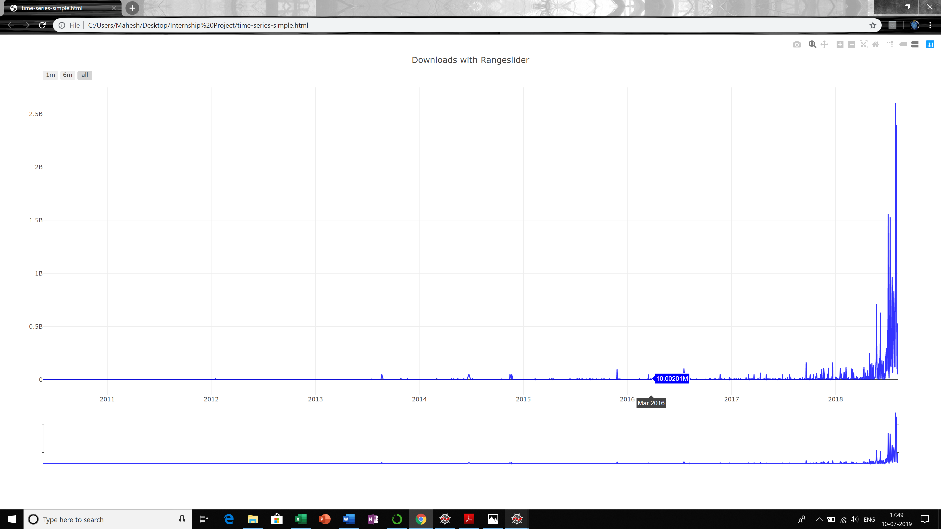
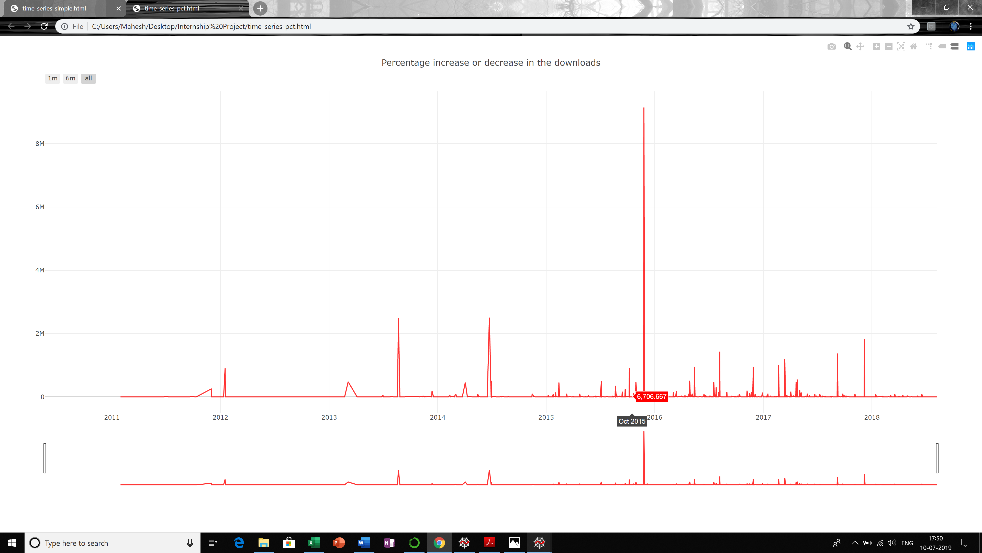
T.insert(tk.END, "+")

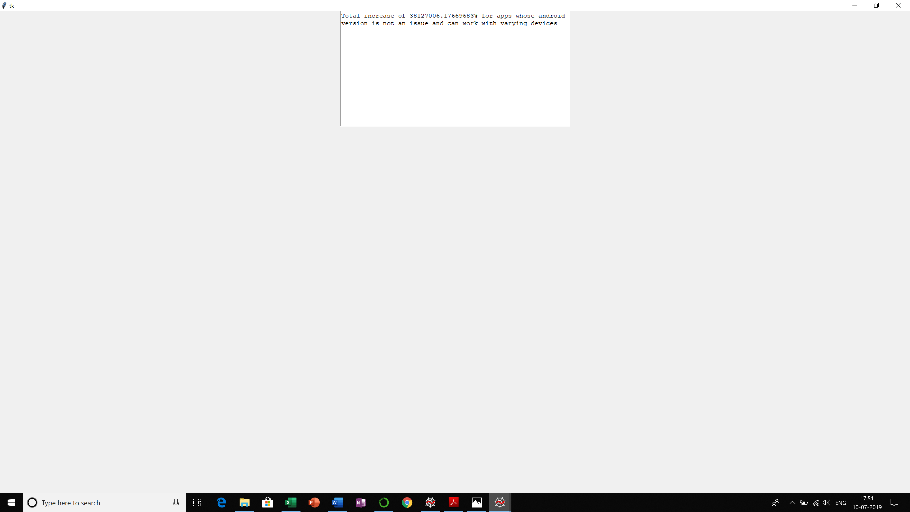
T.insert(tk.END, pct[i])

T.insert(tk.END, "%")

T.insert(tk.END, "\n")

root.mainloop()

1.  



apps=common()

apps=apps[apps['Android Ver'] != "Varies with device"]

data=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

data['Date'] = pd.DatetimeIndex(data['Date'])

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Date']).aggregate(aggregation\_functions).reset\_index()

trace0 = go.Scatter(

x=data.Date,

y=data.Installs,

name = "Downloads",

line = dict(color = 'blue'),

opacity = 0.8)

data2 = [trace0]

layout = dict(title='Downloads with Rangeslider',xaxis=dict(rangeselector=dict(buttons=list([dict(count=1,label='1m',step='month',stepmode='backward'),dict(count=6,label='6m',step='month',stepmode='backward'),dict(step='all')])),rangeslider=dict(visible = True),type='date'))

fig = dict(data=data2, layout=layout)

plotly.offline.plot(fig, filename = 'time-series-simple.html')

apps=common()

apps=apps[apps['Android Ver'] != "Varies with device"]

data=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

data['Date'] = pd.DatetimeIndex(data['Date'])

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Date']).aggregate(aggregation\_functions).reset\_index()

data['Installs']=data['Installs'].pct\_change()

data['Installs']=data['Installs']\*100

change=data.Installs.sum()

trace1 = go.Scatter(x=data.Date,y=data.Installs,name = "Percentage",line = dict(color = 'red'),opacity = 0.8)

data3 = [trace1]

layout = dict(title='Percentage increase or decrease in the downloads',xaxis=dict(rangeselector=dict(buttons=list([dict(count=1,label='1m',step='month',stepmode='backward'),dict(count=6,label='6m',step='month',stepmode='backward'),dict(step='all')])),rangeslider=dict(visible = True),type='date'))

fig = dict(data=data3, layout=layout)

plotly.offline.plot(fig, filename = 'time-series-pct.html')

root=Tk()

T = tk.Text(root, height=15, width=60)

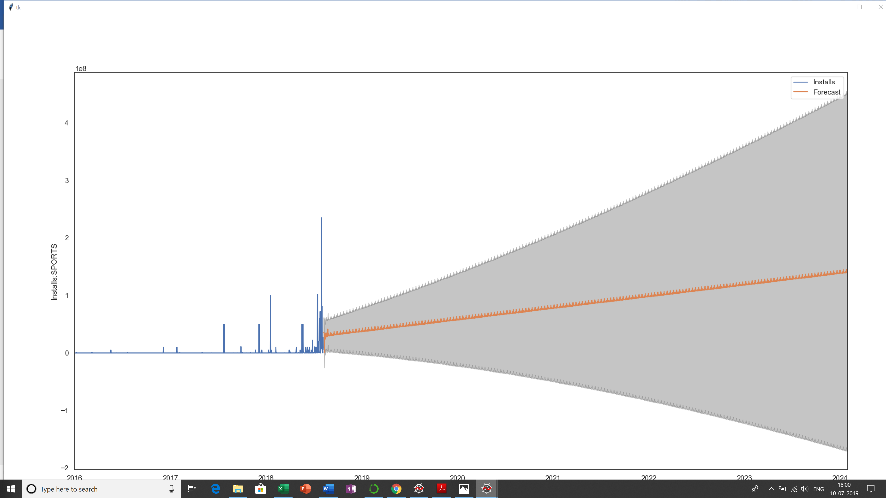
T.pack()

T.insert(tk.END,"Total increase of ")

T.insert(tk.END,change)

T.insert(tk.END,"% for apps whose android version is not an issue and can work with varying devices")

root.mainloop()

1. 

apps=common()

def conclusion():

root = tk.Tk()

T = tk.Text(root, height=30, width=100)

T.pack()

T.insert(tk.END,"According to the forecasting by SARIMAX Model,\n apps of SPORTS Category is most likely to be downloaded in the upcoming years")

tk.mainloop()

def prd():

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

data['Year'] = pd.DatetimeIndex(data['Last Updated']).year

data=data[data.Year > 2015]

del data['Year']

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Date']).aggregate(aggregation\_functions).reset\_index()

y=data[data.Category==var.get()]

del y['Category']

groupby\_day = y.groupby(pd.Grouper(key="Date", freq='D'))

results = groupby\_day.sum()

idx = pd.date\_range("2016-01-01 00:00:00", max(y.Date))

y=results.reindex(idx, fill\_value=0)

mod = sm.tsa.statespace.SARIMAX(y,

order=(1, 1, 1),

seasonal\_order=(1, 1, 1, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

pred\_uc = results.get\_forecast(steps=2000)

pred\_ci = pred\_uc.conf\_int()

root=tk.Tk()

figure3 = plt.Figure(figsize=(20,15), dpi=100)

ax=figure3.add\_subplot(111)

y.plot(ax=ax, label='Observed')

pred\_uc.predicted\_mean.plot(ax=ax, label='Forecast')

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1], color='k', alpha=.25)

scatter3 = FigureCanvasTkAgg(figure3, root)

scatter3.get\_tk\_widget().pack(side=tk.LEFT, fill=tk.BOTH)

ax.legend()

ax.set\_xlabel('Date')

ax.set\_ylabel('Installs:'+var.get())

root.mainloop()

root = Tk()

root.geometry("%dx%d+%d+%d" % (350, 360, 150, 150))

root.title("View Forecast")

tk.Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

category=['SPORTS','ENTERTAINMENT','SOCIAL','NEWS\_AND\_MAGAZINES','EVENTS','TRAVEL\_AND\_LOCAL','GAME']

var = tk.StringVar(root)

var.set(category[0])

option = tk.OptionMenu(root, var, \*category)

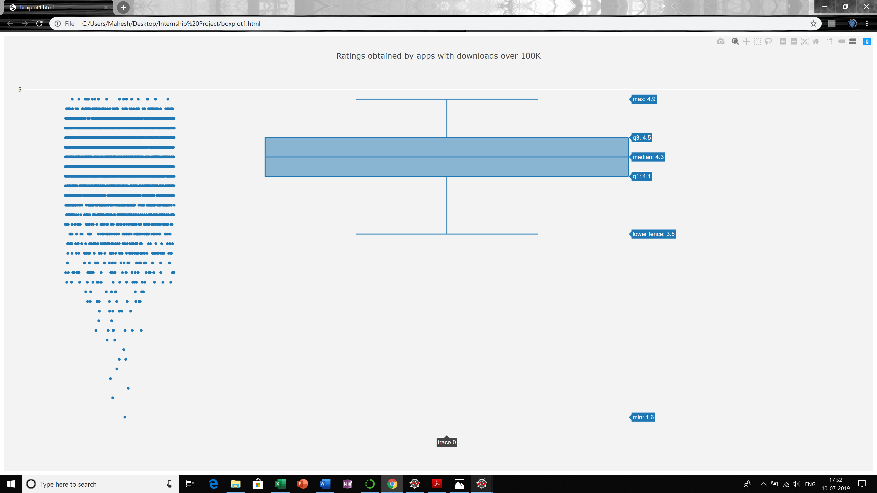
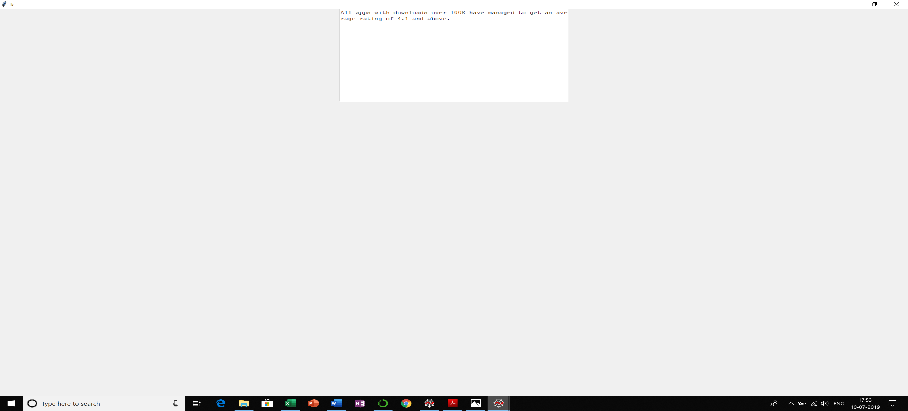
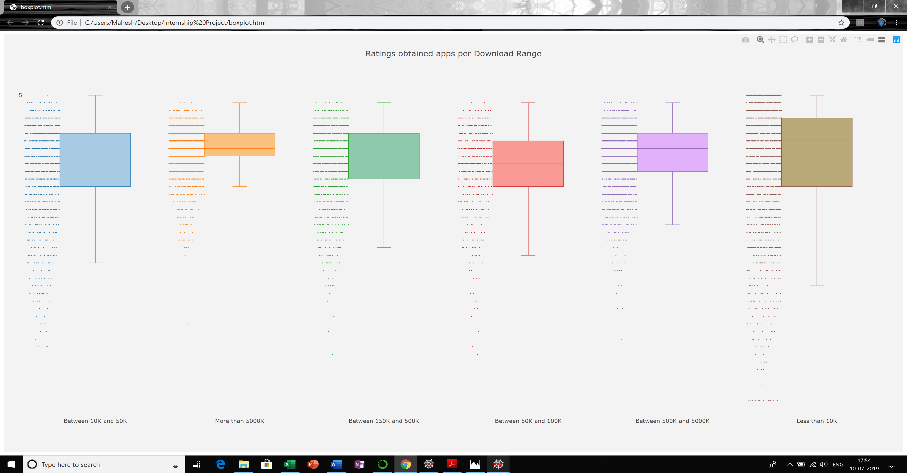
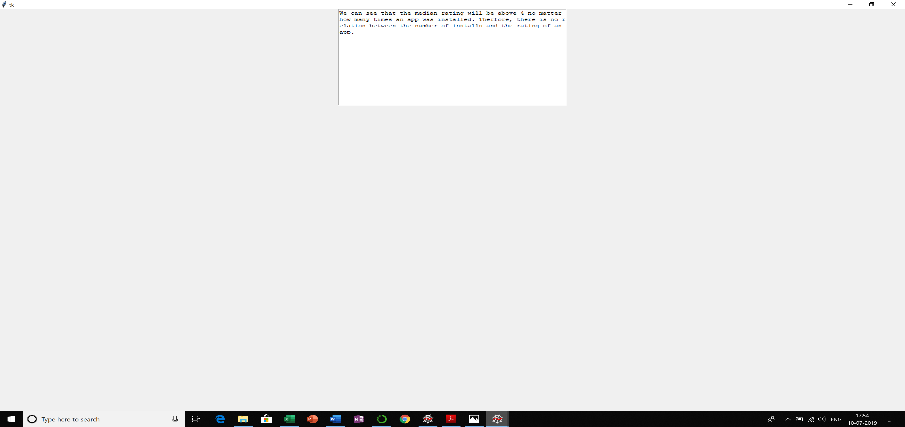
option.pack(side='left', padx=20, pady=20)

button = tk.Button(root, text="FORECAST",font=("Calibri",12,'bold'), bg="goldenrod1", fg="black", command=prd)

button.pack(side='left', padx=20, pady=20)

tk.Button(root,text="Conclusion",bg="goldenrod1",width=10,height=1,font=(" Times",12,'bold'),fg='black',command=conclusion).place(x=100,y=300)

root.mainloop()

9.   

apps=common()

def conclusion():

root = tk.Tk()

T = tk.Text(root, height=30, width=100)

T.pack()

T.insert(tk.END,"According to the forecasting by SARIMAX Model,\n apps of SPORTS Category is most likely to be downloaded in the upcoming years")

tk.mainloop()

def prd():

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

data['Year'] = pd.DatetimeIndex(data['Last Updated']).year

data=data[data.Year > 2015]

del data['Year']

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Date']).aggregate(aggregation\_functions).reset\_index()

y=data[data.Category==var.get()]

del y['Category']

groupby\_day = y.groupby(pd.Grouper(key="Date", freq='D'))

results = groupby\_day.sum()

idx = pd.date\_range("2016-01-01 00:00:00", max(y.Date))

y=results.reindex(idx, fill\_value=0)

mod = sm.tsa.statespace.SARIMAX(y,

order=(1, 1, 1),

seasonal\_order=(1, 1, 1, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

pred\_uc = results.get\_forecast(steps=2000)

pred\_ci = pred\_uc.conf\_int()

root=tk.Tk()

figure3 = plt.Figure(figsize=(20,15), dpi=100)

ax=figure3.add\_subplot(111)

y.plot(ax=ax, label='Observed')

pred\_uc.predicted\_mean.plot(ax=ax, label='Forecast')

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1], color='k', alpha=.25)

scatter3 = FigureCanvasTkAgg(figure3, root)

scatter3.get\_tk\_widget().pack(side=tk.LEFT, fill=tk.BOTH)

ax.legend()

ax.set\_xlabel('Date')

ax.set\_ylabel('Installs:'+var.get())

root.mainloop()

root = Tk()

root.geometry("%dx%d+%d+%d" % (350, 360, 150, 150))

root.title("View Forecast")

tk.Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

category=['SPORTS','ENTERTAINMENT','SOCIAL','NEWS\_AND\_MAGAZINES','EVENTS','TRAVEL\_AND\_LOCAL','GAME']

var = tk.StringVar(root)

var.set(category[0])

option = tk.OptionMenu(root, var, \*category)

option.pack(side='left', padx=20, pady=20)

button = tk.Button(root, text="FORECAST",font=("Calibri",12,'bold'), bg="goldenrod1", fg="black", command=prd)

button.pack(side='left', padx=20, pady=20)

tk.Button(root,text="Conclusion",bg="goldenrod1",width=10,height=1,font=(" Times",12,'bold'),fg='black',command=conclusion).place(x=100,y=300)

root.mainloop()

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"All apps with downloads over 100K have managed to get an average rating of 4.1 and above.")

root.mainloop()

apps=common()

apps=apps.filter(['Installs','Rating'])

apps=apps.dropna()

apps['Installs']=pd.cut(apps['Installs'], bins=[0, 10000, 50000, 100000, 500000, 5000000, 999999999999], include\_lowest=True, labels=['Less than 10K','Between 10K and 50K','Between 50K and 100K','Between 150K and 500K','Between 500K and 5000K','More than 5000K'])

a=apps.loc[apps['Installs'] == 'Between 10K and 50K']

range\_list = list(apps.Installs.unique())

ratings = []

for range in range\_list:

x = apps[apps.Installs == range]

ratings.append(x.Rating.tolist())

colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)', 'rgba(44, 160, 101, 0.5)', 'rgba(255, 65, 54, 0.5)', 'rgba(207, 114, 255, 0.5)', 'rgba(127, 96, 0, 0.5)']

traces = []

for xd, yd, cls in zip(range\_list, ratings, colors):

traces.append(go.Box(

y=yd,

name=xd,

boxpoints='all',

jitter=0.5,

whiskerwidth=0.2,

fillcolor=cls,

marker=dict(

size=2,

),

line=dict(width=1),

))

layout = go.Layout(

title='Ratings obtained apps per Download Range',

yaxis=dict(

autorange=True,

showgrid=True,

zeroline=True,

dtick=5,

gridcolor='rgb(255, 255, 255)',

gridwidth=1,

zerolinecolor='rgb(255, 255, 255)',

zerolinewidth=2,

),

margin=dict(

l=40,

r=30,

b=80,

t=100,

),

paper\_bgcolor='rgb(243, 243, 243)',

plot\_bgcolor='rgb(243, 243, 243)',

showlegend=False

)

fig = go.Figure(data=traces, layout=layout)

plotly.offline.plot(fig, filename="boxplot.html")

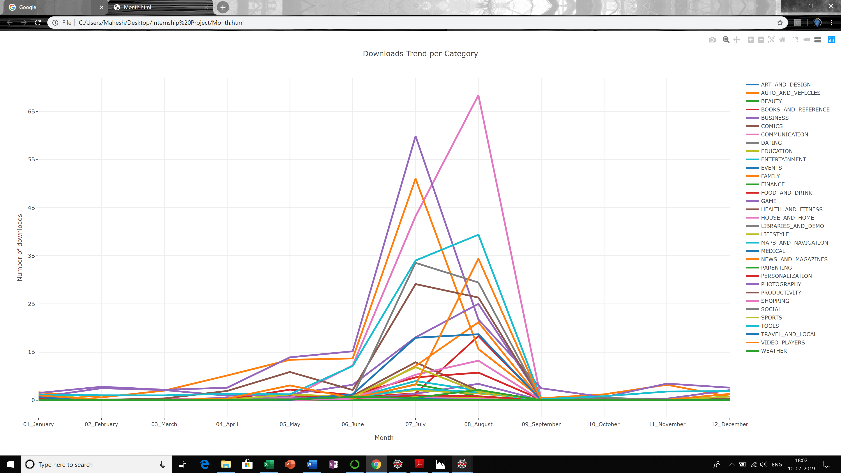
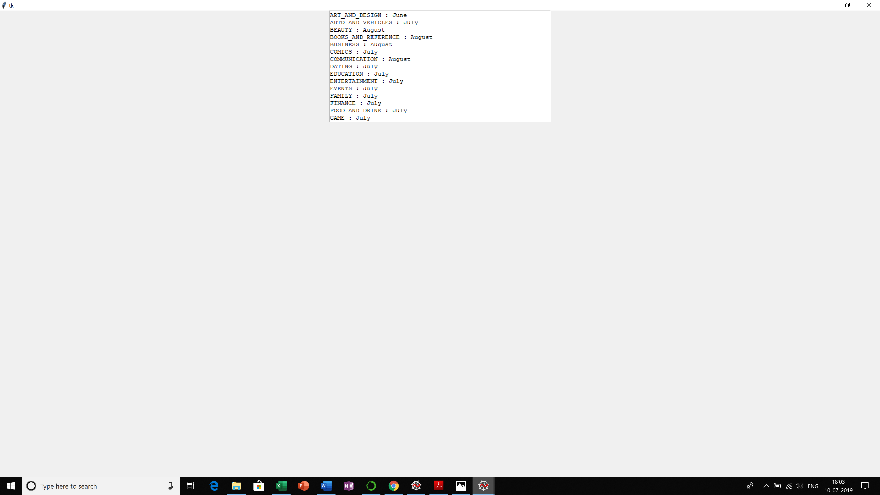
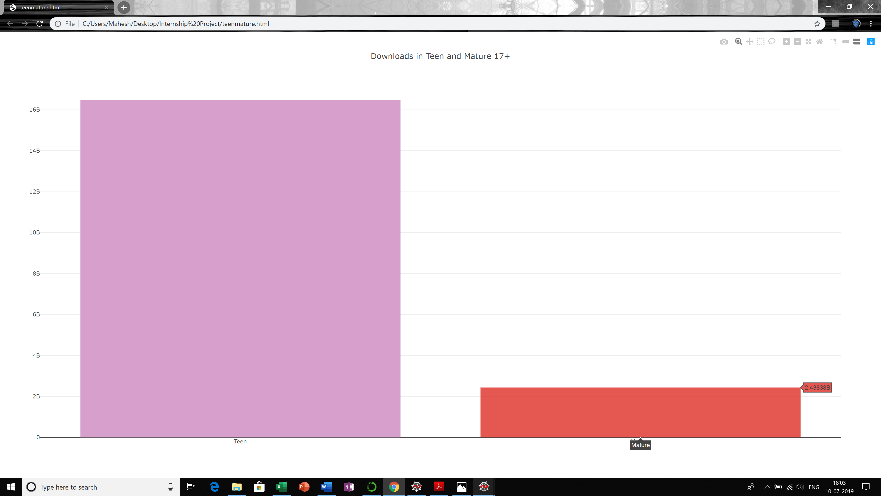
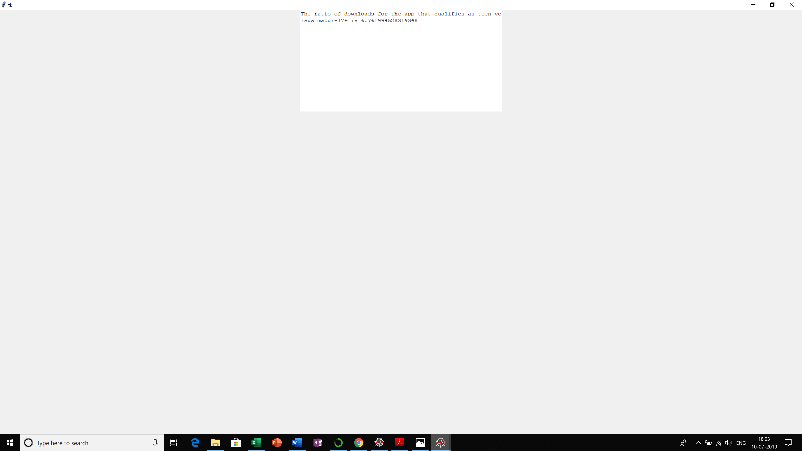
root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"We can see that the median rating will be above 4 no matter how many times an app was installed. Therfore, there is no relation between the number of installs and the rating of an app.")

root.mainloop()

10)    

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

month=[]

for i in range(len(data)):

month.append((data['Last Updated'].iloc[i]).month)

data.drop('Last Updated', axis=1, inplace=True)

data['Month']=month

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Month']).aggregate(aggregation\_functions).reset\_index()

data2=data

data['Month'].replace(to\_replace =[1,2,3,4,5,6,7,8,9,10,11,12],

value =["01\_January","02\_February","03\_March","04\_April","05\_May","06\_June","07\_July","08\_August","09\_September","10\_October","11\_November","12\_December"],inplace=True)

data=data.pivot(index='Month', columns='Category', values='Installs')

data=data.fillna(0)

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Month', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downloads', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='Month.html')

data2['Month'].replace(to\_replace =["01\_January","02\_February","03\_March","04\_April","05\_May","06\_June","07\_July","08\_August","09\_September","10\_October","11\_November","12\_December"],

value =["January","February","March","April","May","June","July","August","September","October","November","December"],inplace=True)

category\_list=list(data2.Category.unique())

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

for category in category\_list:

x=data2[data2.Category == category]

pos=list(x.Installs)

idx = pos.index(max(pos))

T.insert(tk.END,category)

T.insert(tk.END," : ")

T.insert(tk.END,x['Month'].iloc[idx])

T.insert(tk.END,"\n")

root.mainloop()

apps=common()

x=apps[apps['Content Rating']=='Teen']

d1=x.Installs.sum()

y=apps[apps['Content Rating']=='Mature 17+']

d2=y.Installs.sum()

ratio=d1/d2

trace0 = go.Bar(

x=['Teen','Mature'],

y=[d1,d2],

marker=dict(

color=['rgba(215,160,204,1)', 'rgba(222,45,38,0.8)']),

)

data = [trace0]

layout = go.Layout(

title="Downloads in Teen and Mature 17+",

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='teenmature.html')

root=Tk()

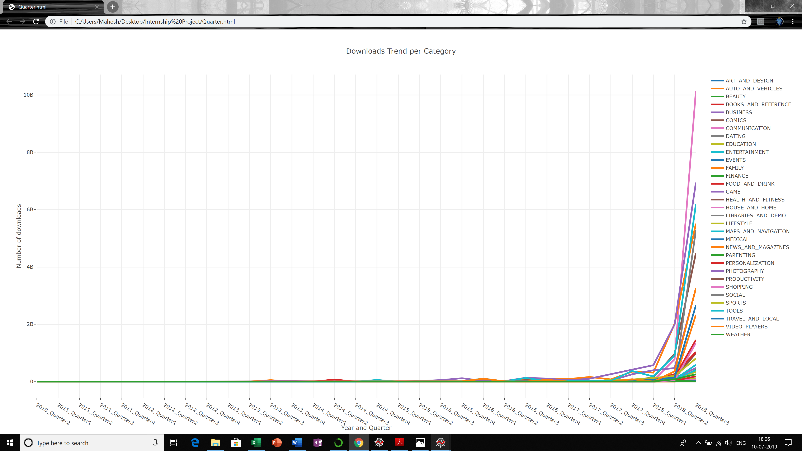
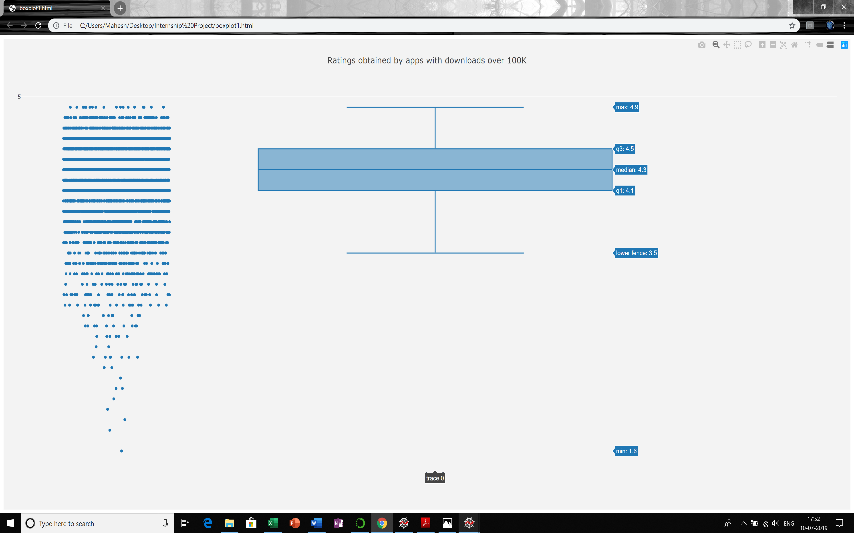
T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"The ratio of downloads for the app that qualifies as teen versus mature17+ is ")

T.insert(tk.END,ratio)

root.mainloop()

11)  

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

def get\_quarter(d):

return "%d\_Quarter%d" % (d.year, math.ceil(d.month/3))

quarter=[]

for i in range(len(data)):

quarter.append(get\_quarter(data['Last Updated'].iloc[i]))

data.drop('Last Updated', axis=1, inplace=True)

data['Quarter']=quarter

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Quarter']).aggregate(aggregation\_functions).reset\_index()

data2=data

data=data.pivot(index='Quarter', columns='Category', values='Installs')

data=data.fillna(0)

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Year and Quarter', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downloads', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='Quarter.html')

category\_list=list(data2.Category.unique())

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

for category in category\_list:

x=data2[data2.Category == category]

pos=list(x.Installs)

idx = pos.index(max(pos))

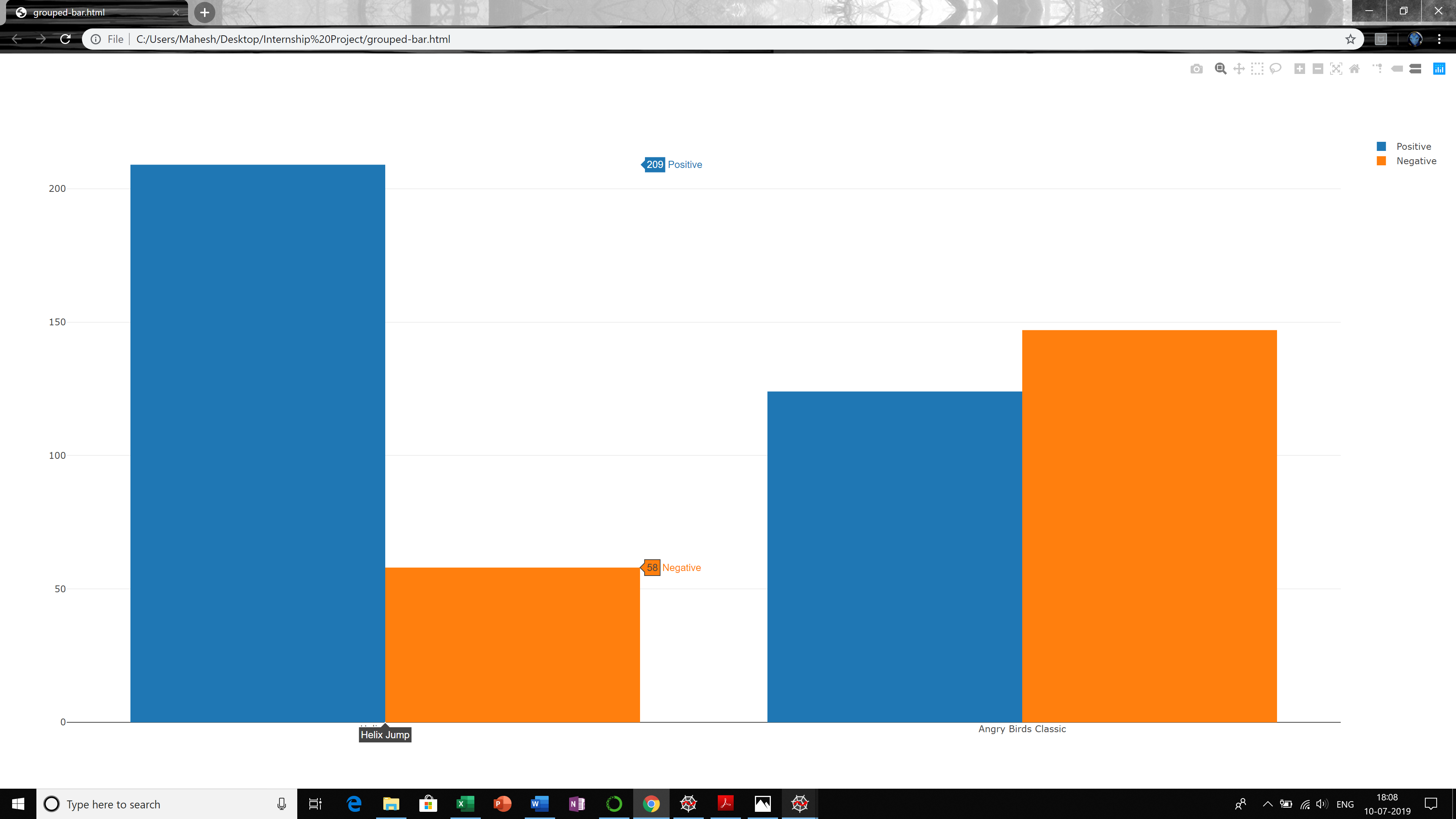
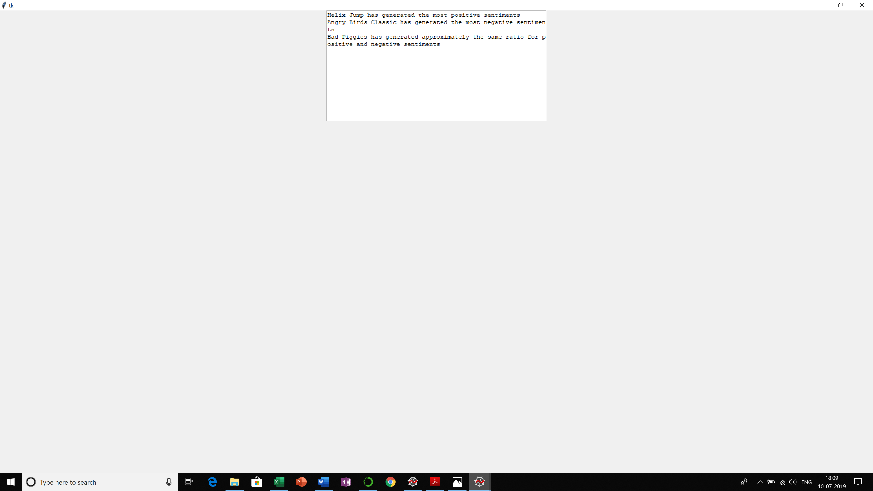
T.insert(tk.END,category)

T.insert(tk.END," : ")

T.insert(tk.END,x['Quarter'].iloc[idx])

T.insert(tk.END,"\n")

root.mainloop()

12)  

apprev=common2()

apprev=apprev.dropna()

app\_list=apprev['App'].unique().tolist()

pos=[]

neg=[]

for app in app\_list:

x = apprev[apprev.App == app]

pos.append(x.Sentiment.str.count("Positive").sum())

neg.append(x.Sentiment.str.count("Negative").sum())

d={'App':app\_list,'Positive':pos,'Negative':neg}

app=pd.DataFrame(d)

app['Ratio'] = app['Positive'] / app['Negative']

app=app.dropna()

array=np.asarray(app['Ratio'])

idx = (np.abs(array - 1)).argmin()

maxpos = pos.index(max(pos))

maxpos1 = neg.index(max(neg))

app\_plot=[app\_list[maxpos],app\_list[maxpos1]]

positive=[pos[maxpos],pos[maxpos1]]

negative=[neg[maxpos],neg[maxpos1]]

trace1 = go.Bar(

x=app\_plot,

y=positive,

name='Positive'

)

trace2 = go.Bar(

x=app\_plot,

y=negative,

name='Negative'

)

data = [trace1, trace2]

layout = go.Layout(

barmode='group'

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='grouped-bar.html')

apprev=common2()

apprev=apprev.dropna()

app\_list=apprev['App'].unique().tolist()

pos=[]

neg=[]

for app in app\_list:

x = apprev[apprev.App == app]

pos.append(x.Sentiment.str.count("Positive").sum())

neg.append(x.Sentiment.str.count("Negative").sum())

d={'App':app\_list,'Positive':pos,'Negative':neg}

app=pd.DataFrame(d)

app['Ratio'] = app['Positive'] / app['Negative']

app=app.dropna()

array=np.asarray(app['Ratio'])

idx = (np.abs(array - 1)).argmin()

maxpos = pos.index(max(pos))

maxpos1 = neg.index(max(neg))

app\_plot=[app\_list[maxpos],app\_list[maxpos1]]

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,app\_plot[0])

T.insert(tk.END," has generated the most positive sentiments")

T.insert(tk.END,"\n")

T.insert(tk.END,app\_plot[1])

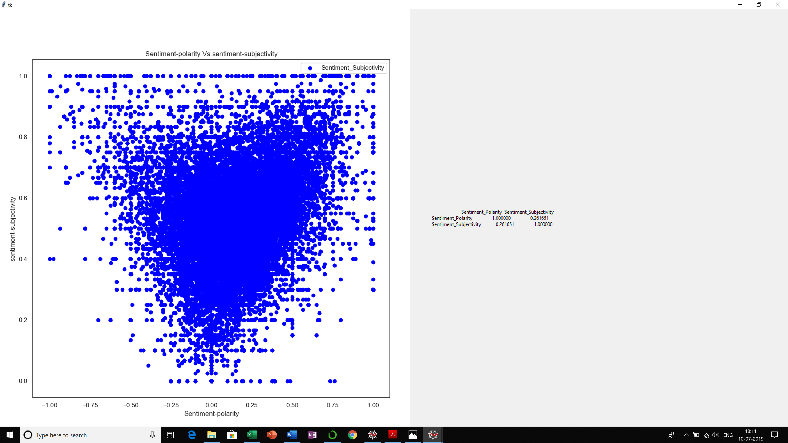
T.insert(tk.END," has generated the most negative sentiments")

T.insert(tk.END,"\n")

T.insert(tk.END,app['App'].iloc[idx])

T.insert(tk.END," has generated approximately the same ratio for positive and negative sentiments")

root.mainloop()

13) 

data=common2()

pogo = data[['Sentiment\_Polarity','Sentiment\_Subjectivity']]

correlation = pogo.corr(method='pearson')

root= tk.Tk()

figure3 = plt.Figure(figsize=(10,10), dpi=100)

ax3 = figure3.add\_subplot(111)

ax3.scatter(data['Sentiment\_Polarity'],data['Sentiment\_Subjectivity'], color = 'blue')

scatter3 = FigureCanvasTkAgg(figure3, root)

scatter3.get\_tk\_widget().pack(side=tk.LEFT, fill=tk.BOTH)

ax3.legend()

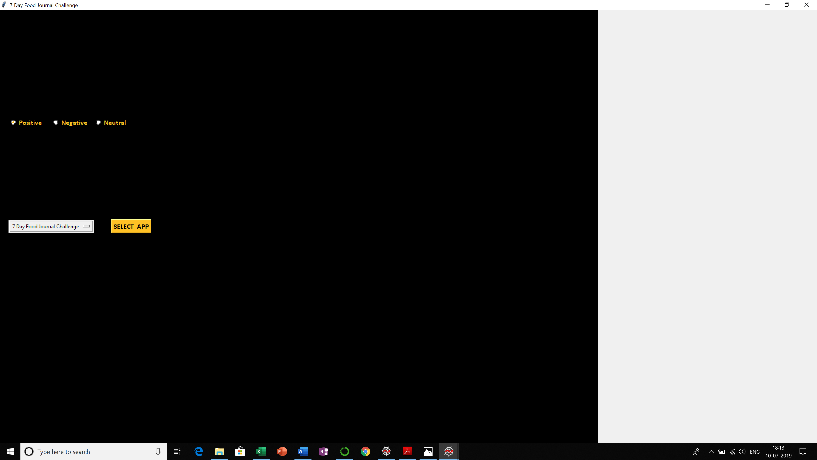
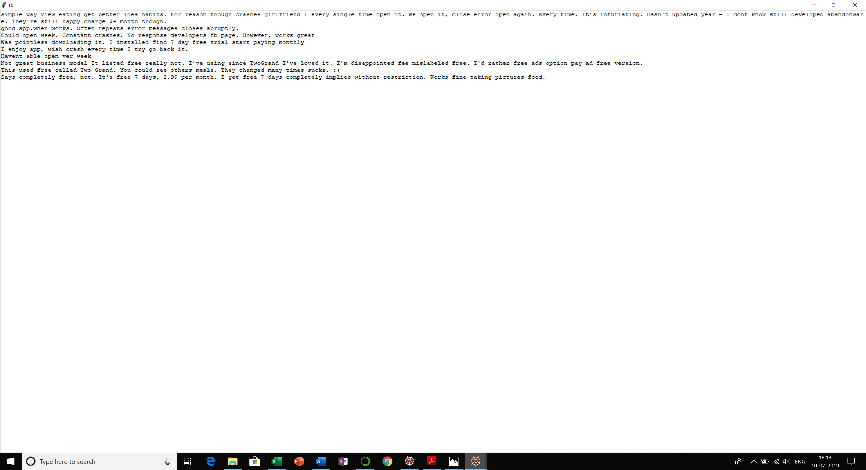
ax3.set\_xlabel('Sentiment-polarity')

ax3.set\_ylabel('sentiment-subjectivity')

ax3.set\_title('Sentiment-polarity Vs.sentiment-subjectivity')

w2 = tk.Label(root, justify=tk.LEFT,padx = 50, text=correlation).pack(side="left")

root.mainloop()

14)  

apprev=common2()

def select():

sf = " %s" % var.get()

root.title(sf)

def display\_review():

print(v.get())

papp=apprev[apprev['App']==var.get()]

papp=papp[papp['Sentiment']==v.get()]

trlist=papp['Translated\_Review'].tolist()

print("\n".join(trlist))

root1=Tk()

T = tk.Text(root1, height=800, width=800)

T.pack()

T.insert(tk.END,"\n".join(trlist))

root1.mainloop()

Radiobutton(root, text="Positive", variable=v, value="Positive", bg='black', fg="goldenrod1",font=("Calibri",12,'bold'), command=display\_review).place(x=20, y=250)

Radiobutton(root, text="Negative", variable=v, value="Negative", bg='black', fg="goldenrod1",font=("Calibri",12,'bold'), command=display\_review).place(x=120, y=250)

Radiobutton(root, text="Neutral", variable=v, value="Neutral", bg='black', fg="goldenrod1",font=("Calibri",12,'bold'), command=display\_review).place(x=220, y=250)

root = Tk()

root.geometry("%dx%d+%d+%d" % (350, 360, 150, 150))

root.title("View Translated Reviews")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

category=apprev['App'].unique()

var = tk.StringVar(root)

v=tk.StringVar(root)

var.set(category[0])

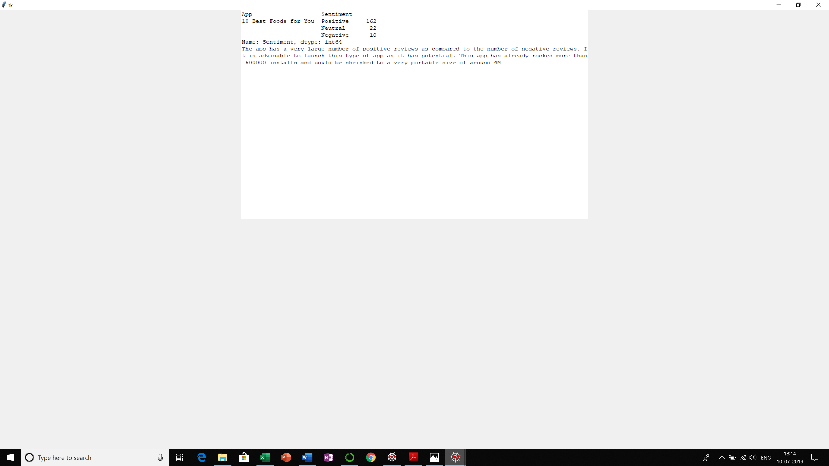
option = tk.OptionMenu(root, var, \*category)

option.pack(side='left', padx=20, pady=20)

button = tk.Button(root, text="SELECT APP",font=("Calibri",12,'bold'), bg="goldenrod1", fg="black", command=select)

button.pack(side='left', padx=20, pady=20)

root.mainloop()

15) 

data=common2()

res = data.groupby(['App','Sentiment']).first()

print(res)

category=data['App'].unique()

variable=category[0]

p=data.groupby('App')

gk=p.get\_group(variable)

print(gk)

gg=gk.groupby('App')['Sentiment'].value\_counts()

print(gg)

print(gg.tolist())

a="The app has a very large number of positive reviews as compared to the number of negative reviews. It is advisable to launch this type of app as it has potential. This app has already rocked more than 500000 installs and could be shrinked to a very portable size of around 4M"

root = tk.Tk()

T = tk.Text(root, height=30, width=100)

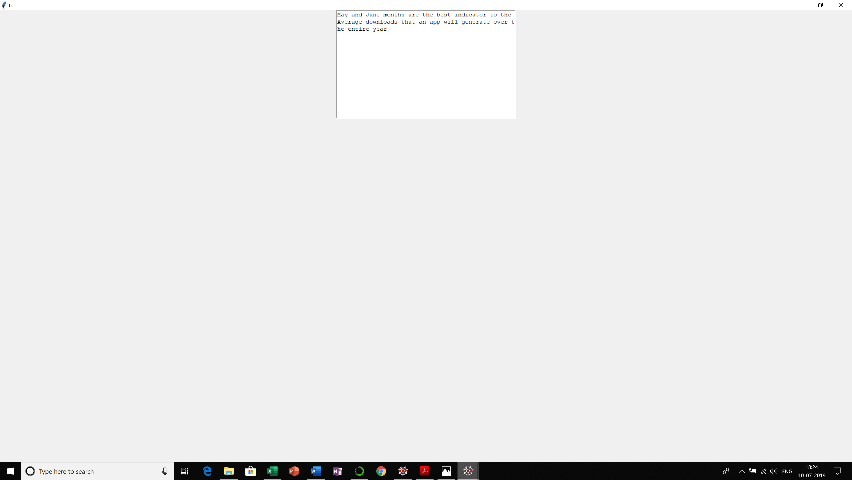
T.pack()

T.insert(tk.END, gg)

T.insert(tk.END, "\n")

T.insert(tk.END,a)

tk.mainloop()

16) 

data=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Year"}, inplace=True)

data['Year'] = pd.DatetimeIndex(data['Year']).year

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Year']).aggregate(aggregation\_functions).reset\_index()

#print(data)

data1=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data1.rename(index=str, columns={"Last Updated": "Month"}, inplace=True)

data1['Month'] = pd.DatetimeIndex(data1['Month']).month

aggregation\_functions = {'Installs': 'sum'}

data1= data1.groupby(['Month']).aggregate(aggregation\_functions).reset\_index()

#print(data1)

x=data['Installs'].sum()/len(data)

x=x/12

print(x)

data1['Installs']=data1['Installs']//len(data)

array=np.asarray(data1['Installs'])

idx = (np.abs(array - x)).argmin()

root=Tk()

T = tk.Text(root, height=15, width=50)

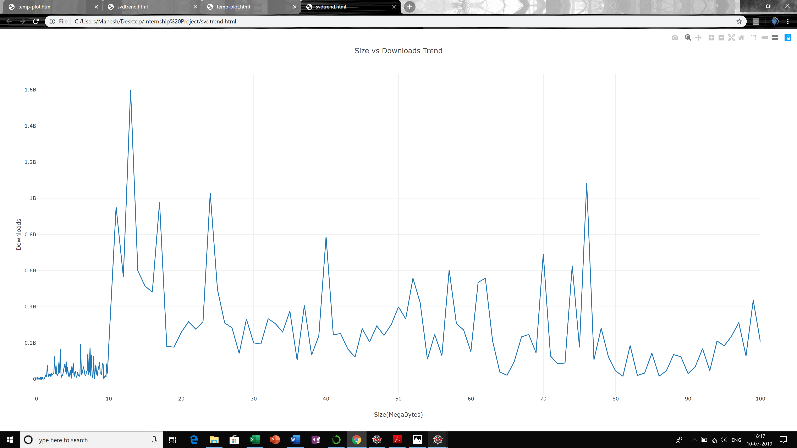
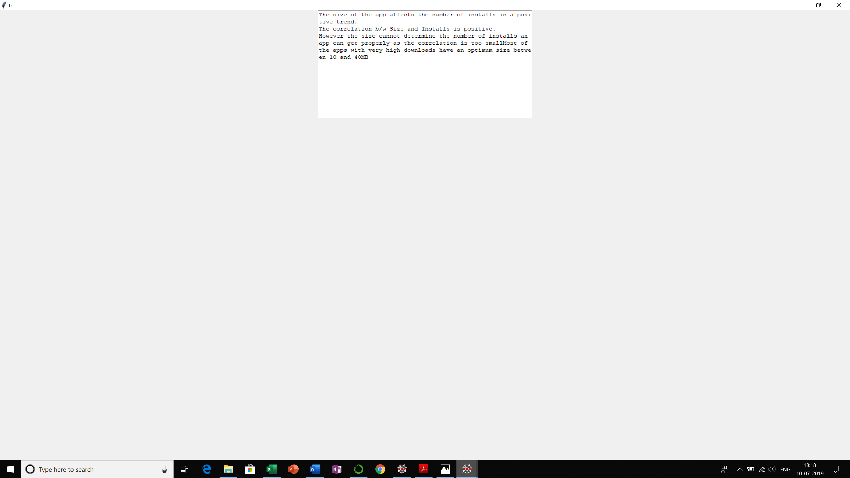
T.pack()

a=calendar.month\_name[idx]+" and "+calendar.month\_name[idx+1]+" months are the best indicator to the"

T.insert(tk.END,a)

T.insert(tk.END,"\nAverage downloads that an app will generate over the entire year")

root.mainloop()

17)  

apps=common()

apps=apps[apps['Size'].notnull()]

trace = [go.Scatter(x=apps["Size"], y=apps["Installs"], mode="markers")]

layout = {"title": "Size vs Downloads",

"xaxis": {"title": "Size in MB"},

"yaxis": {"title": "Downloads"},

"plot\_bgcolor": "rgb(0,0,0)"}

plotly.offline.plot({'data': trace, 'layout': layout})

size\_list=list(apps.Size.unique())

size\_list=sorted(size\_list)

downloads=[]

for size in size\_list:

x = apps[apps.Size == size]

count = x.Installs.sum()

downloads.append(count)

trace0 = go.Scatter(

x = size\_list,

y = downloads,

mode = 'lines',

name = 'lines'

)

data = [trace0]

layout = dict(title = 'Size vs Downloads Trend',

xaxis = dict(title = 'Size(MegaBytes)'),

yaxis = dict(title = 'Downloads'),

)

fig = dict(data=data, layout=layout)

plotly.offline.plot(fig, filename='svdtrend.html')

root=Tk()

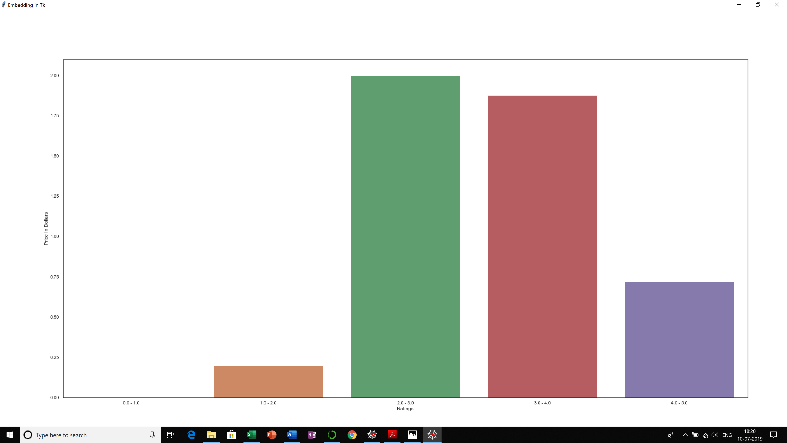
T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"The size of the app affects the number of installs in a positive trend.\nThe correlation b/w Size and Installs is positive.\nHowever the size cannot determine the number of installs an app can get properly as the correlation is too small")

T.insert(tk.END,"Most of the apps with very high downloads have an optimum size between 10 and 40MB")

root.mainloop()

EXTRA1) 

apps=pd.read\_excel("appdatabook.xlsx")

apps['Price In Dollars'] = apps['Price']

apps['Price In Dollars'] = apps['Price In Dollars'].str.replace('Everyone', '$0')

apps['Price In Dollars'] = apps['Price In Dollars'].str.replace('$', '')

apps['Price In Dollars'] = apps['Price In Dollars'].astype(np.float)

apps['Ratings'] = ''

apps.loc[(apps['Rating']>=0.0) & (apps['Rating']<=1.0), 'Ratings'] = '0.0 - 1.0'

apps.loc[(apps['Rating']>=1.0) & (apps['Rating']<=2.0), 'Ratings'] = '1.0 - 2.0'

apps.loc[(apps['Rating']>=2.0) & (apps['Rating']<=3.0), 'Ratings'] = '2.0 - 3.0'

apps.loc[(apps['Rating']>=3.0) & (apps['Rating']<=4.0), 'Ratings'] = '3.0 - 4.0'

apps.loc[(apps['Rating']>=4.0) & (apps['Rating']<=5.0), 'Ratings'] = '4.0 - 5.0'

f=plt.figure(figsize=(25, 8))

sns.barplot(x='Ratings', y='Price In Dollars', data=apps, ci=None, order=['0.0 - 1.0','1.0 - 2.0','2.0 - 3.0','3.0 - 4.0','4.0 - 5.0'])

root = Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f, root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

EXTRA2) 

apps=common()

f=plt.figure(figsize=(22,8))

plt.title('Number of Apps on the basis of Android version required to run them')

sns.countplot(x='Android Ver',data = apps.sort\_values(by = 'Android Ver'),palette='hls')

plt.xticks(rotation = 90)

root = Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f, root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

**VALIDATIONS:**

* Check whether all the fields are filled by the user. If not, display a field empty message to the user.
* Check whether the app rating entered is between 0 and 5.
* Check whether the number of reviews is a valid digit or not.
* Check the validity of the size entry, Display message if the entry is not a float or int.
* The number of Installs should be a digit.
* If the android version or current version entries contain any character except digits or “.”, display an error message.
* Specified Price value should be a float or int.
* Entered Date should be valid. For example, display message if the date entered contains a day value out of range for the specified month and year (or leap year).
* Detect the language of the review entered by the user and translate it to English.
* Calculate the sentiment polarity and subjectivity of the translated review.
* Categorize the Sentiment field as Positive, Negative or Neutral as per the Sentiment Polarity value of the review.

**DATABASE MANAGEMENT:**

The newly entered App data is stored in the mysql database using pymysql. The database contains the same columns as the original datasets. During the execution of the program, the dataset and the mysql database are merged to get the results.

SOURCE CODE:-

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import math

import random

import plotly

import plotly.graph\_objs as go

from string import ascii\_letters

import tkinter

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import time

from datetime import date

from tkinter import\*

from tkinter import messagebox

from tkinter import Tk

import tkinter as tk

import pymysql

from PIL import ImageTk, Image

def adjustscreen(window):

x = 700

y = 700

dx = window.winfo\_screenwidth()

dy = window.winfo\_screenheight()

w = (dx/2) - (x/2)

h = (dy/2) - (y/2)

window.geometry('%dx%d+%d+%d' %(x, y, w, h))

window.resizable(False, False)

window.configure(background = 'white')

def adjustscreen1(window):

x = 700

y = 300

dx = window.winfo\_screenwidth()

dy = window.winfo\_screenheight()

w = (dx/2) - (x/2)

h = (dy/2) - (y/2)

window.geometry('%dx%d+%d+%d' %(x, y, w, h))

window.resizable(False, False)

window.configure(background = 'white')

import time

def is\_date\_valid(year, month, day):

this\_date = '%d/%s/%s' % (month, day, year)

try:

time.strptime(this\_date, '%m/%d/%Y')

except ValueError:

return False

else:

return True

def common():

apps=pd.read\_excel("appdatabook.xlsx")

apps\_columns=apps.columns.tolist()

db\_connection = pymysql.connect(host="localhost", database="playstore", user="root", password=None)

db\_cursor = db\_connection.cursor()

db\_cursor.execute('SELECT \* FROM app\_details')

table\_rows = db\_cursor.fetchall()

db\_connection.commit()

db\_connection.close()

mysql\_data1 = pd.DataFrame(list(table\_rows), columns = apps\_columns)

apps=pd.concat([apps,mysql\_data1],axis=0)

apps=apps.drop\_duplicates(['App'],keep='last')

apps = apps[apps['Android Ver'] != np.nan]

apps = apps[apps['Android Ver'] != 'NaN']

apps = apps[apps['Installs'] != 'Free']

apps = apps[apps['Installs'] != 'Paid']

apps['Installs'] = apps['Installs'].apply(lambda x: x.replace(',', '') if ',' in str(x) else x)

apps['Installs']=apps['Installs'].apply(lambda x: int(str(x).replace('+','')) + 1000 if '+' in str(x) else x)

apps['Installs'] = apps['Installs'].apply(lambda x: int(x))

apps['Size'] = apps['Size'].apply(lambda x: str(x).replace('Varies with device', 'NaN') if 'Varies with device' in str(x) else x)

apps['Size'] = apps['Size'].apply(lambda x: str(x).replace('M', '') if 'M' in str(x) else x)

apps['Size'] = apps['Size'].apply(lambda x: str(x).replace(',', '') if 'M' in str(x) else x)

apps['Size'] = apps['Size'].apply(lambda x: float(str(x).replace('k', '')) / 1000 if 'k' in str(x) else x)

apps['Size'] = apps['Size'].apply(lambda x: float(x))

apps['Price'] = apps['Price'].apply(lambda x: str(x).replace('$', '') if '$' in str(x) else str(x))

apps['Price'] = apps['Price'].apply(lambda x: float(x))

apps['Reviews'] = apps['Reviews'].apply(lambda x: int(x))

return apps

def common2():

apprev=pd.read\_excel("appreview.xlsx")

apprev\_columns=apprev.columns.tolist()

db\_connection = pymysql.connect(host="localhost", database="playstore", user="root", password=None)

db\_cursor = db\_connection.cursor()

db\_cursor.execute('SELECT \* FROM app\_reviews')

table\_rows = db\_cursor.fetchall()

db\_connection.commit()

db\_connection.close()

mysql\_data2 = pd.DataFrame(list(table\_rows), columns = apprev\_columns)

apprev=pd.concat([apprev,mysql\_data2],axis=0)

return apprev

def enter\_dataset1():

def enable\_size():

size\_entry.configure(state="normal")

def disable\_size():

size\_entry.configure(state="disabled")

def enable\_price():

price\_entry.configure(state="normal")

def disable\_price():

price\_entry.configure(state="disabled")

def enable\_cver():

cver\_entry.configure(state="normal")

def disable\_cver():

cver\_entry.configure(state="disabled")

def enable\_aver():

aver\_entry.configure(state="normal")

def disable\_aver():

aver\_entry.configure(state="disabled")

apps=common()

x='disabled'

global dataset1

dataset1 = Toplevel(myroot)

dataset1.title("Data Entry")

adjustscreen(dataset1)

global appname, category, rating, reviews, size, size\_value, installs, atype, price, content\_rating, genres, day, month, year, c\_ver, c\_verog, a\_ver, a\_verog

appname=StringVar()

category=StringVar()

rating=StringVar()

reviews=StringVar()

size=StringVar()

size\_value=StringVar()

installs=StringVar()

atype=StringVar()

price=StringVar()

content\_rating=StringVar()

genres=StringVar()

day=StringVar()

month=StringVar()

year=StringVar()

c\_ver=StringVar()

c\_verog=StringVar()

a\_ver=StringVar()

a\_verog=StringVar()

photo = ImageTk.PhotoImage(Image.open("blacktheme.jpg")) # opening left side image - Note: If image is in same folder then no need to mention the full path

label = Label(dataset1, image=photo, text="") # attaching image to the label

label.place(x=0, y=0)

label.image = photo

Label(dataset1, text="Fill App Data", width='40', height="2", font=("Times New Roman", 24,'bold'), fg='black', bg='goldenrod1',).place(x=0, y=0)

Label(dataset1, text="App Name:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=110)

Entry(dataset1, textvar=appname, width='25').place(x=100, y=110)

Label(dataset1, text="Category:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=150)

list1 = apps['Category'].unique().tolist()

droplist = OptionMenu(dataset1, category, \*list1)

droplist.config(width=17)

category.set('--Select Category--')

droplist.place(x=100, y=150)

Label(dataset1, text="App Rating:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=200)

Entry(dataset1, textvar=rating, width='5').place(x=100, y=200)

Label(dataset1, text="\*Enter a value between 0 and 5", font=("Open Sans", 9, 'bold'), fg='yellow',bg='black', anchor=W).place(x=10, y=220)

Label(dataset1, text="No. of reviews:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=250)

Entry(dataset1, textvar=reviews, width='6').place(x=130, y=250)

Label(dataset1, text="Size:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=280)

Label(dataset1, text="Specify Size:", font=("Open Sans", 10, 'bold'), fg='gray64',bg='black', anchor=W).place(x=10, y=345)

Label(dataset1, text="MB", font=("Open Sans", 10, 'bold'), fg='gray64',bg='black', anchor=W).place(x=140, y=345)

size\_entry=Entry(dataset1, textvar=size\_value, width='4', state=x)

size\_entry.place(x=110, y=345)

Radiobutton(dataset1, text="Varies with device", variable=size, value="Varies",bg='gainsboro',command=disable\_size).place(x=80, y=280)

Radiobutton(dataset1, text="Specify Size", variable=size, value=" ",bg='gainsboro',command=enable\_size).place(x=80, y=310)

Label(dataset1, text="Number of Installs:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=380)

Entry(dataset1, textvar=installs, width='11').place(x=160, y=380)

Label(dataset1, text="Type:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=350, y=110)

Radiobutton(dataset1, text="Free", variable=atype, value="Free", bg='gainsboro',command=disable\_price).place(x=400, y=110)

Radiobutton(dataset1, text="Paid", variable=atype, value="Paid", bg='gainsboro',command=enable\_price).place(x=400, y=150)

Label(dataset1, text="Enter Price:", font=("Open Sans", 10, 'bold'), fg='gray64',bg='black', anchor=W).place(x=350, y=190)

Label(dataset1, text="$", font=("Open Sans", 10, 'bold'), fg='gray64',bg='black', anchor=W).place(x=430, y=190)

price\_entry=Entry(dataset1, textvar=price, width='5', state=x)

price\_entry.place(x=450, y=190)

Label(dataset1, text="Content Rating:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=350, y=230)

list1 = apps['Content Rating'].unique().tolist()

droplist = OptionMenu(dataset1, content\_rating, \*list1)

droplist.config(width=20)

content\_rating.set('--Select Content Rating--')

droplist.place(x=490, y=230)

Label(dataset1, text="Genre:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=350, y=280)

list1 = apps['Genres'].unique().tolist()

droplist = OptionMenu(dataset1, genres, \*list1)

droplist.config(width=17)

genres.set('--Select Genre--')

droplist.place(x=430, y=280)

Label(dataset1, text="Enter Date of Last Update:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=350, y=350)

day\_list=list(range(1, 31))

import calendar

month\_list=calendar.month\_name[1:13]

year\_list=list(range(2014,2019))

droplista = OptionMenu(dataset1, day, \*day\_list)

droplista.config(width=3)

day.set('--Day--')

droplista.place(x=350, y=380)

droplistb = OptionMenu(dataset1, month, \*month\_list)

droplistb.config(width=9)

month.set('--Month--')

droplistb.place(x=410, y=380)

droplistc = OptionMenu(dataset1, year, \*year\_list)

droplistc.config(width=4)

year.set('--Year--')

droplistc.place(x=510, y=380)

Label(dataset1, text="Current Version:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=420)

Radiobutton(dataset1, text="Varies with device", variable=c\_ver, value="Varies with device", bg='gainsboro',command=disable\_cver).place(x=10, y=450)

Radiobutton(dataset1, text="Specify current app version:", variable=c\_ver, value=0, bg='gainsboro',command=enable\_cver).place(x=10, y=480)

cver\_entry=Entry(dataset1, textvar=c\_verog, width='16', state=x)

cver\_entry.place(x=200, y=485)

Label(dataset1, text="\*eg:-6.5.1", font=("Open Sans", 9, 'bold'), fg='yellow',bg='black', anchor=W).place(x=10, y=505)

Label(dataset1, text="Android Version:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=350, y=420)

Radiobutton(dataset1, text="Varies with device", variable=a\_ver, value="Varies with device", bg='gainsboro',command=disable\_aver).place(x=350, y=450)

Radiobutton(dataset1, text="Specify Android version:", variable=a\_ver, value=0, bg='gainsboro',command=enable\_aver).place(x=350, y=480)

Label(dataset1, text="and up", font=("Open Sans", 10, 'bold'), fg='white',bg='black', anchor=W).place(x=615, y=485)

aver\_entry=Entry(dataset1, textvar=a\_verog, width='10', state=x)

aver\_entry.place(x=540, y=485)

Label(dataset1, text="\*eg:-2.3.3 and up", font=("Open Sans", 9, 'bold'), fg='yellow',bg='black', anchor=W).place(x=350, y=505)

Button(dataset1, text='Submit', width=20, font=("Open Sans", 18, 'bold'), bg='goldenrod1', fg='black',command=input\_dataset1).place(x=200, y=580)

def enter\_dataset2():

global dataset2, appn, t

appn=StringVar()

dataset2 = Toplevel(myroot)

dataset2.title("Data Entry")

adjustscreen1(dataset2)

photo = ImageTk.PhotoImage(Image.open("blacktheme.jpg")) # opening left side image - Note: If image is in same folder then no need to mention the full path

label = Label(dataset2, image=photo, text="") # attaching image to the label

label.place(x=0, y=0)

label.image = photo

Label(dataset2, text="Fill App Data", width='50', height="2", font=("Calibri", 22,'bold'), fg='black', bg='goldenrod1').place(x=0, y=0)

Label(dataset2, text="App Name:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=110)

Entry(dataset2, textvar=appn, width='45', font=("Calibri", 15)).place(x=100, y=110)

Label(dataset2, text="Write Review:", font=("Open Sans", 11, 'bold'), fg='white',bg='black', anchor=W).place(x=10, y=160)

t=Text(dataset2, height=3, width=60, font=("Calibri", 11))

t.place(x=10, y=190)

Button(dataset2, text='Submit', width=10, font=("Open Sans", 18, 'bold'), bg='goldenrod1', fg='black',command=input\_dataset2).place(x=500, y=190)

def input\_dataset1():

if appname.get() and rating.get() and reviews.get() and size.get() and installs.get() and atype.get() and c\_ver.get() and a\_ver.get():

if category.get() == "--Select Category--":

Label(dataset1, text="\*Please select Category", fg="DarkOrange2", font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

elif content\_rating.get() == "--Select Content Rating--":

Label(dataset1, text="\*Please select Content Rating", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

elif genres.get() == "--Select Genre--":

Label(dataset1, text="\*Please select Genre", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

elif day.get() == "--Day--" or month.get() == "--Month--" or year.get() == "--Year--":

Label(dataset1, text="\*Please fill Date field", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

if not(rating.get().replace('.','',1).isdigit()):

Label(dataset1, text="\*App Rating entered is invalid", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

if not(0<=float(rating.get())<=5):

Label(dataset1, text="\*App Rating entered is out of bounds(should be between 0 and 5)", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

if not(reviews.get().isdigit()):

Label(dataset1, text="\*Number of reviews entered should be numeric", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

if size.get()=="Varies":

size\_var="Varies with device"

else:

if not(size\_value.get()):

Label(dataset1, text="\*Size not entered", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

if not(size\_value.get().replace('.','',1).isdigit()):

Label(dataset1, text="\*Size entered should be numeric", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

size\_var=size\_value.get()+"M"

if not(installs.get().isdigit()):

Label(dataset1, text="\*Number of installs entered should be numeric", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

installs\_count=installs.get()+"+"

if c\_ver.get()=="Varies with device":

cur\_ver=c\_ver.get()

elif c\_ver.get()=="0":

if not c\_verog.get():

Label(dataset1, text="\*Current version not entered", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

elif not(c\_verog.get().replace('.','').isdigit()):

Label(dataset1, text="\*Current version entered contains invalid characters", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

cur\_ver=c\_verog.get()

if a\_ver.get()=="Varies with device":

add\_ver=a\_ver.get()

elif a\_ver.get()=="0":

if not a\_verog.get():

Label(dataset1, text="\*Android version not entered", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

elif not(a\_verog.get().replace('.','').isdigit()):

Label(dataset1, text="\*Android version entered contains invalid characters", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

add\_ver=a\_verog.get()+" and up"

if atype.get()=="Free":

price\_value="0"

else:

if not price.get():

Label(dataset1, text="\*Price not entered", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

elif not(price.get().replace('.','',1).isdigit()):

Label(dataset1, text="\*Price entered is invalid", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

price\_value="$"+price.get()

import calendar

m=list(calendar.month\_name).index(month.get())

if not is\_date\_valid(year.get(), m, day.get()):

Label(dataset1, text="\*entered Date is invalid", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

else:

date\_var=month.get()+" "+day.get()+", "+year.get()

connection = pymysql.connect(host="localhost", user="root", passwd=None, database="playstore")

cursor = connection.cursor()

insert\_query = "INSERT INTO app\_details (Name,Category,Rating,Reviews,Size,Installs,Type,Price,ContentRating,Genres,LastUpdated,CurrentVer,AndroidVer) VALUES('"+ appname.get() + "', '"+ category.get() + "', '"+ rating.get() +"', '"+ reviews.get() + "', '"+ size\_var + "', '"+ installs\_count + "', '"+ atype.get() + "', '"+ price\_value + "', '"+ content\_rating.get() + "', '"+ genres.get() + "', '"+ date\_var + "', '"+ cur\_ver + "', '"+ add\_ver + "' );"

cursor.execute(insert\_query)

connection.commit()

connection.close()

Label(dataset1, text="Entry Success", fg="lime green", font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

ok=Tk()

ok.title(" Confirm ")

ok.geometry("200x70+600+400")

Label(ok, text="", bg='black', width='120', height='60').place(x=0, y=0)

Label(ok, text="Data Entered", fg="white", font=("calibri", 17, 'bold'), width='60', anchor=CENTER, bg='black').pack()

Button(ok,text="OK",bg="goldenrod1",width=6,height=1,font=(" Times",11,'bold'),fg='black',command=lambda:[ok.destroy(),dataset1.destroy()]).pack()

ok.mainloop()

else:

Label(dataset1, text="Please fill all the details", fg="DarkOrange2",font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=650)

return

def input\_dataset2():

global write\_rev

write\_rev=t.get('1.0','end-1c')

if write\_rev=="":

Label(dataset2, text="\*Empty review field", fg="DarkOrange2", font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=280)

return

else:

if not appn.get():

Label(dataset2, text="\*Empty name field", fg="DarkOrange2", font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=280)

return

else:

from textblob import TextBlob

tr = TextBlob(write\_rev)

try:

tr=tr.translate(to='en')

except Exception as e:

print(e.\_\_doc\_\_)

sen\_pol=tr.polarity

sen\_sub=tr.subjectivity

trev=str(tr)

if sen\_pol>0:

sentiment="Positive"

elif sen\_pol<0:

sentiment="Negative"

else:

sentiment="Neutral"

connection=pymysql.connect(host="localhost",user="root",passwd=None,database="playstore")

cursor=connection.cursor()

insert\_query= "INSERT INTO app\_reviews (App,Translated\_Review,Sentiment,Sentiment\_Polarity,Sentiment\_Subjectivity) VALUES('"+ appn.get() + "', '"+ trev + "', '"+ sentiment + "', '"+ str(sen\_pol) + "', '"+ str(sen\_sub) + "' );"

cursor.execute(insert\_query)

connection.commit()

connection.close()

Label(dataset2, text="Entry Success", fg="lime green", font=("calibri", 11), width='60', anchor=W, bg='black').place(x=0, y=280)

ok=Tk()

ok.title(" Confirm ")

ok.geometry("200x70+600+400")

Label(ok, text="", bg='black', width='120', height='60').place(x=0, y=0)

Label(ok, text="Data Entered", fg="white", font=("calibri", 17, 'bold'), width='60', anchor=CENTER, bg='black').pack()

Button(ok,text="OK",bg="goldenrod1",width=6,height=1,font=(" Times",11,'bold'),fg='black',command=lambda:[ok.destroy(),dataset2.destroy()]).pack()

ok.mainloop()

def dataentry():

myroot=Tk()

Label(myroot, text="", bg='black', width='150', height='100').place(x=0, y=0)

myroot.title("DATA ENTRY")

myroot.geometry("600x400+200+200")

Label(myroot, text="Enter data in Playstore App database", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=30, y=100)

Button(myroot,text="ENTER",bg="goldenrod1",width=8,height=1,font=(" Times",13,'bold'),fg='black',command=enter\_dataset1).place(x=470,y=100)

Label(myroot, text="Enter data in App Reviews/Sentiments database", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=30, y=200)

Button(myroot,text="ENTER",bg="goldenrod1",width=8,height=1,font=(" Times",13,'bold'),fg='black',command=enter\_dataset2).place(x=470,y=200)

Button(myroot,text="EXIT",bg="goldenrod1",width=8,height=1,font=(" Times",13,'bold'),fg='black',command=myroot.destroy).place(x=450,y=320)

myroot.mainloop()

def forecast():

apps=common()

def conclusion():

root = tk.Tk()

T = tk.Text(root, height=30, width=100)

T.pack()

T.insert(tk.END,"According to the forecasting by SARIMAX Model,\n apps of SPORTS Category is most likely to be downloaded in the upcoming years")

tk.mainloop()

def prd():

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

data['Year'] = pd.DatetimeIndex(data['Last Updated']).year

data=data[data.Year > 2015]

del data['Year']

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Date']).aggregate(aggregation\_functions).reset\_index()

y=data[data.Category==var.get()]

del y['Category']

groupby\_day = y.groupby(pd.Grouper(key="Date", freq='D'))

results = groupby\_day.sum()

idx = pd.date\_range("2016-01-01 00:00:00", max(y.Date))

y=results.reindex(idx, fill\_value=0)

mod = sm.tsa.statespace.SARIMAX(y,

order=(1, 1, 1),

seasonal\_order=(1, 1, 1, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

pred\_uc = results.get\_forecast(steps=2000)

pred\_ci = pred\_uc.conf\_int()

root=tk.Tk()

figure3 = plt.Figure(figsize=(20,15), dpi=100)

ax=figure3.add\_subplot(111)

y.plot(ax=ax, label='Observed')

pred\_uc.predicted\_mean.plot(ax=ax, label='Forecast')

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1], color='k', alpha=.25)

scatter3 = FigureCanvasTkAgg(figure3, root)

scatter3.get\_tk\_widget().pack(side=tk.LEFT, fill=tk.BOTH)

ax.legend()

ax.set\_xlabel('Date')

ax.set\_ylabel('Installs:'+var.get())

root.mainloop()

root = Tk()

root.geometry("%dx%d+%d+%d" % (350, 360, 150, 150))

root.title("View Forecast")

tk.Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

category=['SPORTS','ENTERTAINMENT','SOCIAL','NEWS\_AND\_MAGAZINES','EVENTS','TRAVEL\_AND\_LOCAL','GAME']

var = tk.StringVar(root)

var.set(category[0])

option = tk.OptionMenu(root, var, \*category)

option.pack(side='left', padx=20, pady=20)

button = tk.Button(root, text="FORECAST",font=("Calibri",12,'bold'), bg="goldenrod1", fg="black", command=prd)

button.pack(side='left', padx=20, pady=20)

tk.Button(root,text="Conclusion",bg="goldenrod1",width=10,height=1,font=(" Times",12,'bold'),fg='black',command=conclusion).place(x=100,y=300)

root.mainloop()

def que15():

data=common2()

res = data.groupby(['App','Sentiment']).first()

print(res)

category=data['App'].unique()

variable=category[0]

p=data.groupby('App')

gk=p.get\_group(variable)

print(gk)

gg=gk.groupby('App')['Sentiment'].value\_counts()

print(gg)

print(gg.tolist())

a="The app has a very large number of positive reviews as compared to the number of negative reviews. It is advisable to launch this type of app as it has potential. This app has already rocked more than 500000 installs and could be shrinked to a very portable size of around 4M"

root = tk.Tk()

T = tk.Text(root, height=30, width=100)

T.pack()

T.insert(tk.END, gg)

T.insert(tk.END, "\n")

T.insert(tk.END,a)

tk.mainloop()

def que14():

apprev=common2()

def select():

sf = " %s" % var.get()

root.title(sf)

def display\_review():

print(v.get())

papp=apprev[apprev['App']==var.get()]

papp=papp[papp['Sentiment']==v.get()]

trlist=papp['Translated\_Review'].tolist()

print("\n".join(trlist))

root1=Tk()

T = tk.Text(root1, height=800, width=800)

T.pack()

T.insert(tk.END,"\n".join(trlist))

root1.mainloop()

Radiobutton(root, text="Positive", variable=v, value="Positive", bg='black', fg="goldenrod1",font=("Calibri",12,'bold'), command=display\_review).place(x=20, y=250)

Radiobutton(root, text="Negative", variable=v, value="Negative", bg='black', fg="goldenrod1",font=("Calibri",12,'bold'), command=display\_review).place(x=120, y=250)

Radiobutton(root, text="Neutral", variable=v, value="Neutral", bg='black', fg="goldenrod1",font=("Calibri",12,'bold'), command=display\_review).place(x=220, y=250)

root = Tk()

root.geometry("%dx%d+%d+%d" % (350, 360, 150, 150))

root.title("View Translated Reviews")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

category=apprev['App'].unique()

var = tk.StringVar(root)

v=tk.StringVar(root)

var.set(category[0])

option = tk.OptionMenu(root, var, \*category)

option.pack(side='left', padx=20, pady=20)

button = tk.Button(root, text="SELECT APP",font=("Calibri",12,'bold'), bg="goldenrod1", fg="black", command=select)

button.pack(side='left', padx=20, pady=20)

root.mainloop()

def que13():

data=common2()

pogo = data[['Sentiment\_Polarity','Sentiment\_Subjectivity']]

correlation = pogo.corr(method='pearson')

root= tk.Tk()

figure3 = plt.Figure(figsize=(10,10), dpi=100)

ax3 = figure3.add\_subplot(111)

ax3.scatter(data['Sentiment\_Polarity'],data['Sentiment\_Subjectivity'], color = 'blue')

scatter3 = FigureCanvasTkAgg(figure3, root)

scatter3.get\_tk\_widget().pack(side=tk.LEFT, fill=tk.BOTH)

ax3.legend()

ax3.set\_xlabel('Sentiment-polarity')

ax3.set\_ylabel('sentiment-subjectivity')

ax3.set\_title('Sentiment-polarity Vs.sentiment-subjectivity')

w2 = tk.Label(root, justify=tk.LEFT,padx = 50, text=correlation).pack(side="left")

root.mainloop()

def extra1():

apps=pd.read\_excel("appdatabook.xlsx")

apps['Price In Dollars'] = apps['Price']

apps['Price In Dollars'] = apps['Price In Dollars'].str.replace('Everyone', '$0')

apps['Price In Dollars'] = apps['Price In Dollars'].str.replace('$', '')

apps['Price In Dollars'] = apps['Price In Dollars'].astype(np.float)

apps['Ratings'] = ''

apps.loc[(apps['Rating']>=0.0) & (apps['Rating']<=1.0), 'Ratings'] = '0.0 - 1.0'

apps.loc[(apps['Rating']>=1.0) & (apps['Rating']<=2.0), 'Ratings'] = '1.0 - 2.0'

apps.loc[(apps['Rating']>=2.0) & (apps['Rating']<=3.0), 'Ratings'] = '2.0 - 3.0'

apps.loc[(apps['Rating']>=3.0) & (apps['Rating']<=4.0), 'Ratings'] = '3.0 - 4.0'

apps.loc[(apps['Rating']>=4.0) & (apps['Rating']<=5.0), 'Ratings'] = '4.0 - 5.0'

f=plt.figure(figsize=(25, 8))

sns.barplot(x='Ratings', y='Price In Dollars', data=apps, ci=None, order=['0.0 - 1.0','1.0 - 2.0','2.0 - 3.0','3.0 - 4.0','4.0 - 5.0'])

root = Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f, root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

def extra2():

apps=common()

f=plt.figure(figsize=(22,8))

plt.title('Number of Apps on the basis of Android version required to run them')

sns.countplot(x='Android Ver',data = apps.sort\_values(by = 'Android Ver'),palette='hls')

plt.xticks(rotation = 90)

root = Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f, root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

def que17():

root=Tk()

root.title(" Analysis ")

root.geometry("500x200+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="View Correlation Matrix", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=correlation).place(x=390,y=50)

Label(root, text="Size vs Downloads", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que17a).place(x=390,y=100)

Label(root, text="Conclusion", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=150)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que17b).place(x=390,y=150)

def que17a():

apps=common()

apps=apps[apps['Size'].notnull()]

trace = [go.Scatter(x=apps["Size"], y=apps["Installs"], mode="markers")]

layout = {"title": "Size vs Downloads",

"xaxis": {"title": "Size in MB"},

"yaxis": {"title": "Downloads"},

"plot\_bgcolor": "rgb(0,0,0)"}

plotly.offline.plot({'data': trace, 'layout': layout})

size\_list=list(apps.Size.unique())

size\_list=sorted(size\_list)

downloads=[]

for size in size\_list:

x = apps[apps.Size == size]

count = x.Installs.sum()

downloads.append(count)

trace0 = go.Scatter(

x = size\_list,

y = downloads,

mode = 'lines',

name = 'lines'

)

data = [trace0]

layout = dict(title = 'Size vs Downloads Trend',

xaxis = dict(title = 'Size(MegaBytes)'),

yaxis = dict(title = 'Downloads'),

)

fig = dict(data=data, layout=layout)

plotly.offline.plot(fig, filename='svdtrend.html')

def que17b():

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"The size of the app affects the number of installs in a positive trend.\nThe correlation b/w Size and Installs is positive.\nHowever the size cannot determine the number of installs an app can get properly as the correlation is too small")

T.insert(tk.END,"Most of the apps with very high downloads have an optimum size between 10 and 40MB")

root.mainloop()

def que12():

root=Tk()

root.title(" Analysis ")

root.geometry("650x200+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="App Category with most Positive and Negative Sentiments", justify=LEFT, fg="white", font=("calibri",13,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que12a).place(x=540,y=50)

Label(root, text="App Category with same ratio for Positive and Negative Sentiments", justify=LEFT, fg="white", font=("calibri",13,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que12b).place(x=540,y=100)

def que12a():

apprev=common2()

apprev=apprev.dropna()

app\_list=apprev['App'].unique().tolist()

pos=[]

neg=[]

for app in app\_list:

x = apprev[apprev.App == app]

pos.append(x.Sentiment.str.count("Positive").sum())

neg.append(x.Sentiment.str.count("Negative").sum())

d={'App':app\_list,'Positive':pos,'Negative':neg}

app=pd.DataFrame(d)

app['Ratio'] = app['Positive'] / app['Negative']

app=app.dropna()

array=np.asarray(app['Ratio'])

idx = (np.abs(array - 1)).argmin()

maxpos = pos.index(max(pos))

maxpos1 = neg.index(max(neg))

app\_plot=[app\_list[maxpos],app\_list[maxpos1]]

positive=[pos[maxpos],pos[maxpos1]]

negative=[neg[maxpos],neg[maxpos1]]

trace1 = go.Bar(

x=app\_plot,

y=positive,

name='Positive'

)

trace2 = go.Bar(

x=app\_plot,

y=negative,

name='Negative'

)

data = [trace1, trace2]

layout = go.Layout(

barmode='group'

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='grouped-bar.html')

def que12b():

apprev=common2()

apprev=apprev.dropna()

app\_list=apprev['App'].unique().tolist()

pos=[]

neg=[]

for app in app\_list:

x = apprev[apprev.App == app]

pos.append(x.Sentiment.str.count("Positive").sum())

neg.append(x.Sentiment.str.count("Negative").sum())

d={'App':app\_list,'Positive':pos,'Negative':neg}

app=pd.DataFrame(d)

app['Ratio'] = app['Positive'] / app['Negative']

app=app.dropna()

array=np.asarray(app['Ratio'])

idx = (np.abs(array - 1)).argmin()

maxpos = pos.index(max(pos))

maxpos1 = neg.index(max(neg))

app\_plot=[app\_list[maxpos],app\_list[maxpos1]]

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,app\_plot[0])

T.insert(tk.END," has generated the most positive sentiments")

T.insert(tk.END,"\n")

T.insert(tk.END,app\_plot[1])

T.insert(tk.END," has generated the most negative sentiments")

T.insert(tk.END,"\n")

T.insert(tk.END,app['App'].iloc[idx])

T.insert(tk.END," has generated approximately the same ratio for positive and negative sentiments")

root.mainloop()

def que11():

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

def get\_quarter(d):

return "%d\_Quarter%d" % (d.year, math.ceil(d.month/3))

quarter=[]

for i in range(len(data)):

quarter.append(get\_quarter(data['Last Updated'].iloc[i]))

data.drop('Last Updated', axis=1, inplace=True)

data['Quarter']=quarter

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Quarter']).aggregate(aggregation\_functions).reset\_index()

data2=data

data=data.pivot(index='Quarter', columns='Category', values='Installs')

data=data.fillna(0)

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Year and Quarter', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downloads', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='Quarter.html')

category\_list=list(data2.Category.unique())

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

for category in category\_list:

x=data2[data2.Category == category]

pos=list(x.Installs)

idx = pos.index(max(pos))

T.insert(tk.END,category)

T.insert(tk.END," : ")

T.insert(tk.END,x['Quarter'].iloc[idx])

T.insert(tk.END,"\n")

root.mainloop()

def que10():

root=Tk()

root.title(" Analysis ")

root.geometry("500x200+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="Monthwise Classification of downloads", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que10a).place(x=390,y=50)

Label(root, text="Ratio of downloads for the App that\nqualifies as Teen versus Mature 17+", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que10b).place(x=390,y=100)

def que10a():

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

month=[]

for i in range(len(data)):

month.append((data['Last Updated'].iloc[i]).month)

data.drop('Last Updated', axis=1, inplace=True)

data['Month']=month

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Month']).aggregate(aggregation\_functions).reset\_index()

data2=data

data['Month'].replace(to\_replace =[1,2,3,4,5,6,7,8,9,10,11,12],

value =["01\_January","02\_February","03\_March","04\_April","05\_May","06\_June","07\_July","08\_August","09\_September","10\_October","11\_November","12\_December"],inplace=True)

data=data.pivot(index='Month', columns='Category', values='Installs')

data=data.fillna(0)

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Month', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downloads', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='Month.html')

data2['Month'].replace(to\_replace =["01\_January","02\_February","03\_March","04\_April","05\_May","06\_June","07\_July","08\_August","09\_September","10\_October","11\_November","12\_December"],

value =["January","February","March","April","May","June","July","August","September","October","November","December"],inplace=True)

category\_list=list(data2.Category.unique())

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

for category in category\_list:

x=data2[data2.Category == category]

pos=list(x.Installs)

idx = pos.index(max(pos))

T.insert(tk.END,category)

T.insert(tk.END," : ")

T.insert(tk.END,x['Month'].iloc[idx])

T.insert(tk.END,"\n")

root.mainloop()

def que10b():

apps=common()

x=apps[apps['Content Rating']=='Teen']

d1=x.Installs.sum()

y=apps[apps['Content Rating']=='Mature 17+']

d2=y.Installs.sum()

ratio=d1/d2

trace0 = go.Bar(

x=['Teen','Mature'],

y=[d1,d2],

marker=dict(

color=['rgba(215,160,204,1)', 'rgba(222,45,38,0.8)']),

)

data = [trace0]

layout = go.Layout(

title="Downloads in Teen and Mature 17+",

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='teenmature.html')

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"The ratio of downloads for the app that qualifies as teen versus mature17+ is ")

T.insert(tk.END,ratio)

root.mainloop()

def que9():

root=Tk()

root.title(" Analysis ")

root.geometry("600x300+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="Ratings obtained by Apps with downloads over 100k", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que9a).place(x=490,y=50)

Label(root, text="Conclusion", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que9a1).place(x=490,y=100)

Label(root, text="Ratings obtained by apps per download range", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=150)

Button(root,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que9b).place(x=490,y=150)

Label(root, text="Conclusion", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=200)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que9b1).place(x=490,y=200)

def que9a():

apps=common()

apps=apps.filter(['Installs','Rating'])

apps1=apps[apps.Installs>100000]

data = [

go.Box(

y=apps1['Rating'],

boxpoints='all',

jitter=0.3,

pointpos=-1.8

)

]

layout = go.Layout(

title='Ratings obtained by apps with downloads over 100K',

yaxis=dict(

autorange=True,

showgrid=True,

zeroline=True,

dtick=5,

gridcolor='rgb(255, 255, 255)',

gridwidth=1,

zerolinecolor='rgb(255, 255, 255)',

zerolinewidth=2,

),

margin=dict(

l=40,

r=30,

b=80,

t=100,

),

paper\_bgcolor='rgb(243, 243, 243)',

plot\_bgcolor='rgb(243, 243, 243)',

showlegend=False

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename="boxplot1.html")

def que9a1():

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"All apps with downloads over 100K have managed to get an average rating of 4.1 and above.")

root.mainloop()

def que9b():

apps=common()

apps=apps.filter(['Installs','Rating'])

apps=apps.dropna()

apps['Installs']=pd.cut(apps['Installs'], bins=[0, 10000, 50000, 100000, 500000, 5000000, 999999999999], include\_lowest=True, labels=['Less than 10K','Between 10K and 50K','Between 50K and 100K','Between 150K and 500K','Between 500K and 5000K','More than 5000K'])

a=apps.loc[apps['Installs'] == 'Between 10K and 50K']

range\_list = list(apps.Installs.unique())

ratings = []

for range in range\_list:

x = apps[apps.Installs == range]

ratings.append(x.Rating.tolist())

colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)', 'rgba(44, 160, 101, 0.5)', 'rgba(255, 65, 54, 0.5)', 'rgba(207, 114, 255, 0.5)', 'rgba(127, 96, 0, 0.5)']

traces = []

for xd, yd, cls in zip(range\_list, ratings, colors):

traces.append(go.Box(

y=yd,

name=xd,

boxpoints='all',

jitter=0.5,

whiskerwidth=0.2,

fillcolor=cls,

marker=dict(

size=2,

),

line=dict(width=1),

))

layout = go.Layout(

title='Ratings obtained apps per Download Range',

yaxis=dict(

autorange=True,

showgrid=True,

zeroline=True,

dtick=5,

gridcolor='rgb(255, 255, 255)',

gridwidth=1,

zerolinecolor='rgb(255, 255, 255)',

zerolinewidth=2,

),

margin=dict(

l=40,

r=30,

b=80,

t=100,

),

paper\_bgcolor='rgb(243, 243, 243)',

plot\_bgcolor='rgb(243, 243, 243)',

showlegend=False

)

fig = go.Figure(data=traces, layout=layout)

plotly.offline.plot(fig, filename="boxplot.html")

def que9b1():

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"We can see that the median rating will be above 4 no matter how many times an app was installed. Therfore, there is no relation between the number of installs and the rating of an app.")

root.mainloop()

def que7():

root=Tk()

root.title(" Analysis ")

root.geometry("500x200+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="Download trend ", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que7a).place(x=390,y=50)

Label(root, text="Percentage increase or decrease in downloads", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que7b).place(x=390,y=100)

def que7a():

apps=common()

apps=apps[apps['Android Ver'] != "Varies with device"]

data=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

data['Date'] = pd.DatetimeIndex(data['Date'])

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Date']).aggregate(aggregation\_functions).reset\_index()

trace0 = go.Scatter(

x=data.Date,

y=data.Installs,

name = "Downloads",

line = dict(color = 'blue'),

opacity = 0.8)

data2 = [trace0]

layout = dict(title='Downloads with Rangeslider',xaxis=dict(rangeselector=dict(buttons=list([dict(count=1,label='1m',step='month',stepmode='backward'),dict(count=6,label='6m',step='month',stepmode='backward'),dict(step='all')])),rangeslider=dict(visible = True),type='date'))

fig = dict(data=data2, layout=layout)

plotly.offline.plot(fig, filename = 'time-series-simple.html')

def que7b():

apps=common()

apps=apps[apps['Android Ver'] != "Varies with device"]

data=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

data['Date'] = pd.DatetimeIndex(data['Date'])

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Date']).aggregate(aggregation\_functions).reset\_index()

data['Installs']=data['Installs'].pct\_change()

data['Installs']=data['Installs']\*100

change=data.Installs.sum()

trace1 = go.Scatter(x=data.Date,y=data.Installs,name = "Percentage",line = dict(color = 'red'),opacity = 0.8)

data3 = [trace1]

layout = dict(title='Percentage increase or decrease in the downloads',xaxis=dict(rangeselector=dict(buttons=list([dict(count=1,label='1m',step='month',stepmode='backward'),dict(count=6,label='6m',step='month',stepmode='backward'),dict(step='all')])),rangeslider=dict(visible = True),type='date'))

fig = dict(data=data3, layout=layout)

plotly.offline.plot(fig, filename = 'time-series-pct.html')

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,"Total increase of ")

T.insert(tk.END,change)

T.insert(tk.END,"% for apps whose android version is not an issue and can work with varying devices")

root.mainloop()

def que6a():

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Year"}, inplace=True)

data['Year'] = pd.DatetimeIndex(data['Year']).year

data=data[data.Year > 2015]

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Year']).aggregate(aggregation\_functions).reset\_index()

data.sort\_values(by=['Year'],inplace=True)

data=data.pivot(index='Year', columns='Category', values='Installs')

data=data.fillna(1)

category=data.columns.tolist()

d2016=data.values[0].tolist()

d2017=data.values[0].tolist()

d2018=data.values[2].tolist()

dict2016 = dict(zip(category, data.values[0].tolist()))

maximum1 = max(dict2016, key=dict2016.get)

a=maximum1+" Category has the most downloads in 2016"

minimum1 = min(dict2016, key=dict2016.get)

b=minimum1+" Category has the least downloads in 2016"

dict2017 = dict(zip(category, data.values[1].tolist()))

maximum2 = max(dict2017, key=dict2017.get)

c=maximum2+" Category has the most downloads in 2017"

minimum2 = min(dict2017, key=dict2017.get)

d=minimum2+" Category has the least downloads in 2017"

dict2018 = dict(zip(category, data.values[2].tolist()))

maximum3 = max(dict2018, key=dict2018.get)

e=maximum3+" Category has the most downloads in 2018"

minimum3 = min(dict2018, key=dict2018.get)

f=minimum3+" Category has the least downloads in 2018"

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Years', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downlaods', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='yearcomp.html')

root=Tk()

T = tk.Text(root, height=15, width=60)

T.pack()

T.insert(tk.END,a)

T.insert(tk.END,"\n")

T.insert(tk.END,b)

T.insert(tk.END,"\n")

T.insert(tk.END,c)

T.insert(tk.END,"\n")

T.insert(tk.END,d)

T.insert(tk.END,"\n")

T.insert(tk.END,e)

T.insert(tk.END,"\n")

T.insert(tk.END,f)

root.mainloop()

def que6b():

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Year"}, inplace=True)

data['Year'] = data['Year'].apply(lambda x: x.replace('2017','2018') if '2017' in str(x) else x)

data['Year'] = pd.DatetimeIndex(data['Year']).year

data=data[data.Year > 2015]

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Year']).aggregate(aggregation\_functions).reset\_index()

data.sort\_values(by=['Year'],inplace=True)

data=data.pivot(index='Year', columns='Category', values='Installs')

data=data.fillna(1)

data=data.pct\_change()

category=data.columns.tolist()

pct=data.values[1].tolist()

pct = [i \* 100 for i in pct]

root=Tk()

T = tk.Text(root, height=100, width=150)

T.pack()

for i in range(len(category)):

T.insert(tk.END, category[i])

T.insert(tk.END, " : ")

if pct[i]<0:

T.insert(tk.END, "-")

T.insert(tk.END, pct[i])

T.insert(tk.END, "%")

T.insert(tk.END, "\n")

else:

T.insert(tk.END, "+")

T.insert(tk.END, pct[i])

T.insert(tk.END, "%")

T.insert(tk.END, "\n")

root.mainloop()

def que6():

root=Tk()

root.title(" Analysis ")

root.geometry("500x200+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="Download trend from 2016 to 2018", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que6a).place(x=390,y=50)

Label(root, text="Percentage increase or decrease in downloads\n over the period of three years", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que6b).place(x=390,y=100)

def correlation():

apps=common()

corrmat = apps.corr()

trace1 = {

"x": corrmat.columns,

"y": corrmat.columns,

"z": corrmat.values,

"type": "heatmap"

}

data = [trace1]

layout = go.Layout(title='Features Correlation Matrix')

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig)

def que1():

apps=common()

category\_count=apps['Category'].tolist()

installs\_count=apps['Installs'].tolist()

category=apps['Category'].unique()

downloads=list()

for i in category:

dl=0

for j,k in zip(category\_count,installs\_count):

if j==i:

dl=dl+k

downloads.append(dl)

fig = {'data': [{'labels': category,

'values': downloads,

'type': 'pie'}],

'layout': {'title': 'App Installations Category wise'}

}

plotly.offline.plot(fig)

def que4():

apps=common()

category\_list = list(apps.Category.unique())

ratings = []

for category in category\_list:

x = apps[apps.Category == category]

rating\_rate = x.Rating.sum()/len(x)

ratings.append(rating\_rate)

data=pd.DataFrame({'Category':category\_list, 'Rating':ratings})

new\_index = (data['Rating'].sort\_values(ascending=False)).index.values

sorted\_data = data.reindex(new\_index)

sns.set(style="white")

# Set up the matplotlib figure

f, ax = plt.subplots(figsize=(15, 9))

sns.barplot(x=sorted\_data.Category, y=sorted\_data.Rating)

plt.xticks(rotation=45)

plt.show()

root = tkinter.Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f, root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

def que5():

apps=common()

data=pd.concat([apps['Last Updated'],apps['Category'],apps['Installs']],axis=1)

data['Last Updated'] =pd.to\_datetime(data['Last Updated'])

data.rename(index=str, columns={"Last Updated": "Date"}, inplace=True)

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Category','Date']).aggregate(aggregation\_functions).reset\_index()

data.sort\_values(by=['Date'],inplace=True)

data=data.pivot(index='Date', columns='Category', values='Installs')

data=data.fillna(0)

data = [

go.Scatter(

x=data.index,

y=data[name].values,

mode='lines',

name=name,

line=dict(width=4)

) for name in data.columns

]

layout = go.Layout(

title='Downloads Trend per Category',

xaxis=dict(title='Date', ticklen=5, zeroline=False, gridwidth=2),

yaxis=dict(title='Number of downloads', ticklen=5, gridwidth=2),

showlegend=True

)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='Battle4TheTop.html')

def que2a():

apps=common()

sns.set(style="white")

f=plt.figure(figsize = (14,12))

g1 = sns.countplot(x='Installs', data=apps)

g1.set\_title("App Installs Count", fontsize=20)

g1.set\_xlabel("Number of Installs", fontsize=15)

g1.set\_ylabel("Number of Apps", fontsize=15)

plt.xticks(rotation=45)

root = tkinter.Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f,root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

def que2b():

apps=common()

sns.set(style="white")

installs\_categorical=pd.DataFrame({'InstallRange': apps['Installs']})

installs\_categorical=pd.cut(installs\_categorical['InstallRange'], bins=[0, 10000, 50000, 150000, 500000, 5000000, 999999999999], include\_lowest=True, labels=['Less than 10K','Between 10K and 50K','Between 50K and 150K','Between 150K and 500K','Between 500K and 5000K','More than 5000K'])

apps=pd.concat([apps,installs\_categorical],axis=1)

f2=plt.figure(figsize = (14,12))

g1 = sns.countplot(x='InstallRange', data=apps)

g1.set\_title("App Installs Count", fontsize=20)

g1.set\_xlabel("Number of Installs", fontsize=15)

g1.set\_ylabel("Number of Apps", fontsize=15)

plt.xticks(rotation=45)

root = tkinter.Tk()

root.wm\_title("Embedding in Tk")

root.geometry("%dx%d+%d+%d" % (900, 400, 300, 300))

canvas = FigureCanvasTkAgg(f2,root) # A tk.DrawingArea.

canvas.draw()

canvas.get\_tk\_widget().pack(side="top",fill='both',expand=True)

root.mainloop()

def result():

apps=common()

installs\_categorical=pd.DataFrame({'InstallRange': apps['Installs']})

installs\_categorical=pd.cut(installs\_categorical['InstallRange'], bins=[0, 10000, 50000, 150000, 500000, 5000000, 999999999999], include\_lowest=True, labels=['Less than 10K','Between 10K and 50K','Between 50K and 150K','Between 150K and 500K','Between 500K and 5000K','More than 5000K'])

apps=pd.concat([apps,installs\_categorical],axis=1)

root=Tk()

T = tk.Text(root, height=15, width=35)

T.pack()

a=apps['InstallRange'].value\_counts()

T.insert(tk.END,a )

root.mainloop()

def que2():

r=Tk()

r.title(" Analysis ")

r.geometry("500x200+200+200")

Label(r, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(r, text="Downloads Count", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(r,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que2a).place(x=390,y=50)

Label(r, text="Downloads Count according to Range", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(r,text="VIEW PLOT",bg="goldenrod1",width=10,height=1,font=(" Times",11,'bold'),fg='black',command=que2b).place(x=390,y=100)

Label(r, text="Result", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=10, y=150)

Button(r,text="VIEW ",bg="goldenrod1",width=8,height=1,font=(" Times",11,'bold'),fg='black', command=result).place(x=390,y=150)

r.mainloop()

def que16():

data=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data.rename(index=str, columns={"Last Updated": "Year"}, inplace=True)

data['Year'] = pd.DatetimeIndex(data['Year']).year

aggregation\_functions = {'Installs': 'sum'}

data= data.groupby(['Year']).aggregate(aggregation\_functions).reset\_index()

#print(data)

data1=pd.concat([apps['Last Updated'],apps['Installs']],axis=1)

data1.rename(index=str, columns={"Last Updated": "Month"}, inplace=True)

data1['Month'] = pd.DatetimeIndex(data1['Month']).month

aggregation\_functions = {'Installs': 'sum'}

data1= data1.groupby(['Month']).aggregate(aggregation\_functions).reset\_index()

#print(data1)

x=data['Installs'].sum()/len(data)

x=x/12

print(x)

data1['Installs']=data1['Installs']//len(data)

array=np.asarray(data1['Installs'])

idx = (np.abs(array - x)).argmin()

root=Tk()

T = tk.Text(root, height=15, width=50)

T.pack()

a=calendar.month\_name[idx]+" and "+calendar.month\_name[idx+1]+" months are the best indicator to the"

T.insert(tk.END,a)

T.insert(tk.END,"\nAverage downloads that an app will generate over the entire year")

root.mainloop()

def que3a():

apps=common()

cat\_list = list(apps.Category.unique())

dls = []

mldl = []

names=[]

for category in cat\_list:

x = apps[apps.Category == category]

dl\_count = x.Installs.sum()//len(x)

maxleast\_count=x.Installs.sum()

mldl.append(maxleast\_count)

dls.append(dl\_count)

if dl\_count>=250000:

names.append(category)

ml=pd.DataFrame({'Category':cat\_list, 'Downloads':mldl})

new\_index = (ml['Downloads'].sort\_values(ascending=False)).index.values

sorted\_data = ml.reindex(new\_index)

print(sorted\_data['Category'].iloc[0]," Category has the most number of downloads")

print(sorted\_data['Category'].iloc[-1]," Category has the least number of downloads")

trace1 = go.Bar(x=sorted\_data.Category,y=sorted\_data.Downloads,text=sorted\_data.Category,textposition='auto',marker=dict(color='rgb(158,202,225)',line=dict(color='rgb(8,48,107)',width=1.5,)),opacity=0.6)

data = [trace1]

layout = go.Layout(title='Total number of downloads per category in Billions',)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='text-hover-bar.html')

root = tk.Tk()

T = tk.Text(root, height=30, width=60)

T.pack()

T.insert(tk.END, "\nCategory with the most number of downloads: ")

T.insert(tk.END, sorted\_data['Category'].iloc[0],"\n")

T.insert(tk.END, "\nCategory with the least number of downloads: ")

T.insert(tk.END, sorted\_data['Category'].iloc[-1])

tk.mainloop()

def que3b():

apps=common()

cat\_list = list(apps.Category.unique())

dls = []

mldl = []

names=[]

for category in cat\_list:

x = apps[apps.Category == category]

dl\_count = x.Installs.sum()//len(x)

maxleast\_count=x.Installs.sum()

mldl.append(maxleast\_count)

dls.append(dl\_count)

if dl\_count>=250000:

names.append(category)

data=pd.DataFrame({'Category':cat\_list, 'Downloads':dls})

new\_index = (data['Downloads'].sort\_values(ascending=False)).index.values

sorted\_data = data.reindex(new\_index)

trace0 = go.Bar(x=sorted\_data.Category,y=sorted\_data.Downloads,text=sorted\_data.Category,textposition='auto',marker=dict(color='rgb(58,200,225)',line=dict(color='rgb(8,48,107)',width=1.5,)),opacity=0.6)

data = [trace0]

layout = go.Layout(title='Average number of downloads per category in Millions',)

fig = go.Figure(data=data, layout=layout)

plotly.offline.plot(fig, filename='text-hover-barx.html')

root = tk.Tk()

T = tk.Text(root, height=30, width=60)

T.pack()

T.insert(tk.END, "\nCategories with an average of atleast 250K downloads: \n")

T.insert(tk.END,', '.join(names),"\n")

tk.mainloop()

def que3():

root=Tk()

root.title(" Analysis ")

root.geometry("500x150+200+200")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="App Category with most and least Downloads", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=50)

Button(root,text="VIEW",bg="goldenrod1",width=8,height=1,font=(" Times",11,'bold'),fg='black',command=que3a).place(x=390,y=50)

Label(root, text="Average Downloads of App Categories", justify=LEFT, fg="white", font=("calibri",14,'bold'), width='60', anchor=W, bg='black').place(x=10, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=8,height=1,font=(" Times",11,'bold'),fg='black',command=que3b).place(x=390,y=100)

root.mainloop()

def frame3():

root=Tk()

root.title(" Analysis ")

root.geometry("1000x730+20+40")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="Relation between the Sentiment-Polarity and\nSentiment-Subjectivity of all the apps", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que13).place(x=780,y=100)

Label(root, text="Check out the reviews categorized as positive,negative \nand neutral for any App", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=185)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que14).place(x=780,y=185)

Label(root, text=" Is it advisable to launch an app like ’10 Best foods for you’?\nDo the users like these apps?", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=270)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que15).place(x=780,y=270)

Label(root, text="Is there any influence of the size of an App over the number of Installs?", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=355)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que17).place(x=780,y=355)

Label(root, text="App Ratings vs Price(in Dollars)", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=440)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=extra1).place(x=780,y=440)

Label(root, text="Number of Apps per Android Version", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=525)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=extra2).place(x=780,y=525)

Label(root, text="Best Month Indicator to the average downloads per Year", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=600)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que16).place(x=780,y=600)

Button(root,text="PREVIOUS",bg="goldenrod1",width=12,height=1,font=(" Times",15,'bold'),fg='black',command=lambda:[root.destroy(),frame2()]).place(x=100,y=670)

Button(root,text="EXIT",bg="goldenrod1",width=12,height=1,font=(" Times",15,'bold'),fg='black',command=root.destroy).place(x=800,y=670)

root.mainloop()

def frame2():

root=Tk()

root.title(" Analysis ")

root.geometry("1000x730+20+40")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="Analysis of App Downloads over the past three years", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=100)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que6).place(x=780,y=100)

Label(root, text="Analysis of those apps whose android version is not an \nissue and can work with varying devices", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=185)

Button(root,text="GO",bg="goldenrod1",width=10, height=1,font=(" Times",13,'bold'),fg='black',command=que7).place(x=780,y=185)

Label(root, text="Analysis of co-relation between the number of downloads\nand the ratings received. ", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=270)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que9).place(x=780,y=270)

Label(root, text="Analysing App downloads per month and calculating the ratio\n of downloads for the app that qualifies as teen versus mature17+", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=355)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que10).place(x=780,y=355)

Label(root, text=" Which quarter of which year has generated the highest number\nof install for each app used in the study?", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=440)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que11).place(x=780,y=440)

Label(root, text="Categorizing the Apps according to the type of Sentiments", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=525)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que12).place(x=780,y=525)

Button(root,text="NEXT",bg="goldenrod1",width=12,height=1,font=(" Times",15,'bold'),fg='black',command=lambda:[root.destroy(),frame3()]).place(x=800,y=670)

Button(root,text="PREVIOUS",bg="goldenrod1",width=12,height=1,font=(" Times",15,'bold'),fg='black',command=lambda:[root.destroy(),frame1()]).place(x=100,y=670)

root.mainloop()

def frame1():

root=Tk()

root.title(" Analysis ")

root.geometry("1000x730+20+40")

Label(root, text="", bg='black', width='200', height='200').place(x=0, y=0)

Label(root, text="View Correlation Matrix", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=100)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=correlation).place(x=780,y=100)

Label(root, text="Percentage download in each Category on the Playstore", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=185)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que1).place(x=780,y=185)

Label(root, text="Categorization of Apps according to the number of downloads", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=270)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que2).place(x=780,y=270)

Label(root, text="Discovering App Categories with most, least and average Downloads", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=355)

Button(root,text="GO",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que3).place(x=780,y=355)

Label(root, text="Analysing and Calculating average Ratings of each App Category", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=440)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que4).place(x=780,y=440)

Label(root, text="Download Trend over the period of past few years", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=100, y=525)

Button(root,text="VIEW",bg="goldenrod1",width=10,height=1,font=(" Times",13,'bold'),fg='black',command=que5).place(x=780,y=525)

Button(root,text="NEXT",bg="goldenrod1",width=12,height=1,font=(" Times",15,'bold'),fg='black',command=lambda:[root.destroy(),frame2()]).place(x=800,y=670)

root.mainloop()

def mainscreen():

global myroot

myroot=Tk()

myroot.title("Contents")

myroot.geometry("800x600+20+40")

Label(myroot, text="", bg='black', width='150', height='100').place(x=0, y=0)

Label(myroot, text="TABLE OF CONTENTS", fg="white", font=("calibri",26,'bold'), width='24', anchor=CENTER, bg='black',relief="ridge",bd=4).place(x=150, y=70)

Label(myroot, text="Check out analysis of data using various data Visualization tools", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=70, y=200)

Button(myroot,text="Go",bg="goldenrod1",width=5,height=1,font=(" Times",12,'bold'),fg='black',command=frame1).place(x=710,y=200)

Label(myroot, text="Check out Forecasting of Data in future years using \nSARIMAX Time-Series Prediction Model", justify=LEFT, fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=70, y=300)

Button(myroot,text="Go",bg="goldenrod1",width=5,height=1,font=(" Times",12,'bold'),fg='black',command=forecast).place(x=710,y=300)

Label(myroot, text="Add new Data to existing Databases for Analyis and Forecasting", fg="white", font=("calibri",16,'bold'), width='60', anchor=W, bg='black').place(x=70, y=400)

Button(myroot,text="Go",bg="goldenrod1",width=5,height=1,font=(" Times",12,'bold'),fg='black',command=dataentry).place(x=710,y=400)

Button(myroot,text="EXIT",bg="goldenrod1",width=8,height=1,font=(" Times",16,'bold'),fg='black',command=myroot.destroy).place(x=560,y=520)

myroot.mainloop()

mainscreen()