Import Statements

In [89]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
import tkinter
from tkinter import *
from sklearn.ensemble import RandomForestClassifier
```

Reading the Data file. The data file size is 115mb, it has **434891** entries

In [90]:

```
steam_review = pd.read_csv("Datasets/steam_reviews.csv")
```

Data Preprocessing

Checking if there are any null values present in the Dataset. We will be cleaning them out as they can affect our predections.

In [91]:

```
steam_review.head()
```

Out[91]:

	date_posted	funny	helpful	hour_played	is_early_access_review	recommendation	review	title
0	2019-02-10	2	4	578	False	Recommended	&gt Played as German Reich&gt Declare war on B...	Expansion - Hearts of Iron IV: Man the Guns
1	2019-02-10	0	0	184	False	Recommended	yes.	Expansion - Hearts of Iron IV: Man the Guns
2	2019-02-07	0	0	892	False	Recommended	Very good game although a bit overpriced in my...	Expansion - Hearts of Iron IV: Man the Guns
3	2018-06-14	126	1086	676	False	Recommended	Out of all the reviews I wrote This one is pro...	Dead by Daylight
4	2017-06-20	85	2139	612	False	Recommended	Disclaimer I survivor main. I play games for f...	Dead by Daylight

In [92]:

```
steam_review.describe()
```

Out[92]:

	funny	helpful	hour_played
count	4.348910e+05	434891.000000	434891.000000
mean	5.333024e+05	1.004114	364.130773

	funny	helpful	hour_played
std	4.785640e+07	59.462935	545.961198
min	0.000000e+00	0.000000	0.000000
25%	0.000000e+00	0.000000	62.000000
50%	0.000000e+00	0.000000	190.000000
75%	0.000000e+00	0.000000	450.000000
max	4.294967e+09	28171.000000	31962.000000

In [93]:

```
steam_review.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 434891 entries, 0 to 434890
Data columns (total 8 columns):
 #   Column                Non-Null Count   Dtype 
---  ------                --------------   ----- 
 0   date_posted           434891 non-null  object
 1   funny                 434891 non-null  int64 
 2   helpful               434891 non-null  int64 
 3   hour_played           434891 non-null  int64 
 4   is_early_access_review  434891 non-null  bool  
 5   recommendation        434891 non-null  object
 6   review                433375 non-null  object
 7   title                 434891 non-null  object
dtypes: bool(1), int64(3), object(4)
memory usage: 23.6+ MB
```

In [94]:

```
steam_review.isnull().sum()
```

Out[94]:

```
date_posted                  0
funny                        0
helpful                      0
hour_played                  0
is_early_access_review       0
recommendation               0
review                    1516
title                        0
dtype: int64
```

In [95]:

```
steam_review = steam_review.dropna()
```

In [96]:

```
steam_review.isnull().sum()
```

Out[96]:

```
date_posted               0
funny                     0
helpful                   0
hour_played               0
is_early_access_review    0
recommendation            0
review                    0
title                     0
dtype: int64
```

In [97]:

```
steam_review.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 433375 entries, 0 to 434890
```

```
Int64Index: 433375 entries, 0 to 434890
Data columns (total 8 columns):
 #   Column                 Non-Null Count   Dtype 
---  ------                 --------------   ----- 
 0   date_posted            433375 non-null  object
 1   funny                  433375 non-null  int64 
 2   helpful                433375 non-null  int64 
 3   hour_played            433375 non-null  int64 
 4   is_early_access_review 433375 non-null  bool  
 5   recommendation         433375 non-null  object
 6   review                 433375 non-null  object
 7   title                  433375 non-null  object
dtypes: bool(1), int64(3), object(4)
memory usage: 26.9+ MB
```

In [98]:

```
steam_review.recommendation.value_counts()
```

Out[98]:

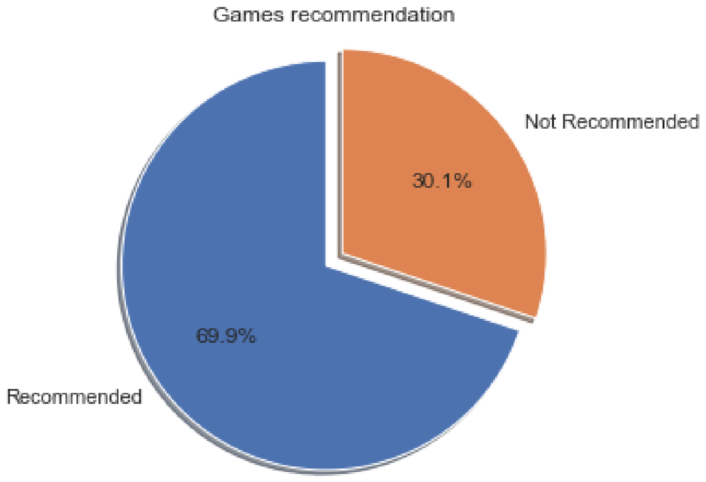
```
Recommended        302751
Not Recommended    130624
Name: recommendation, dtype: int64
```

Data Visualization

Games Recommended Vs Not Recommended besed on the review

In [99]:

```
graph_sizes = [steam_review.recommendation.value_counts()[0], steam_review.recommendatio
n.value_counts()[1]]
labels = ["Recommended", "Not Recommended"]

explode = (0, 0.1)
fig1, ax1 = plt.subplots()
ax1.set_title('Games recommendation')
ax1.pie(graph_sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)

ax1.axis('equal')
plt.tight_layout()
plt.show()
```



Hours Played Vs Number of Reviews

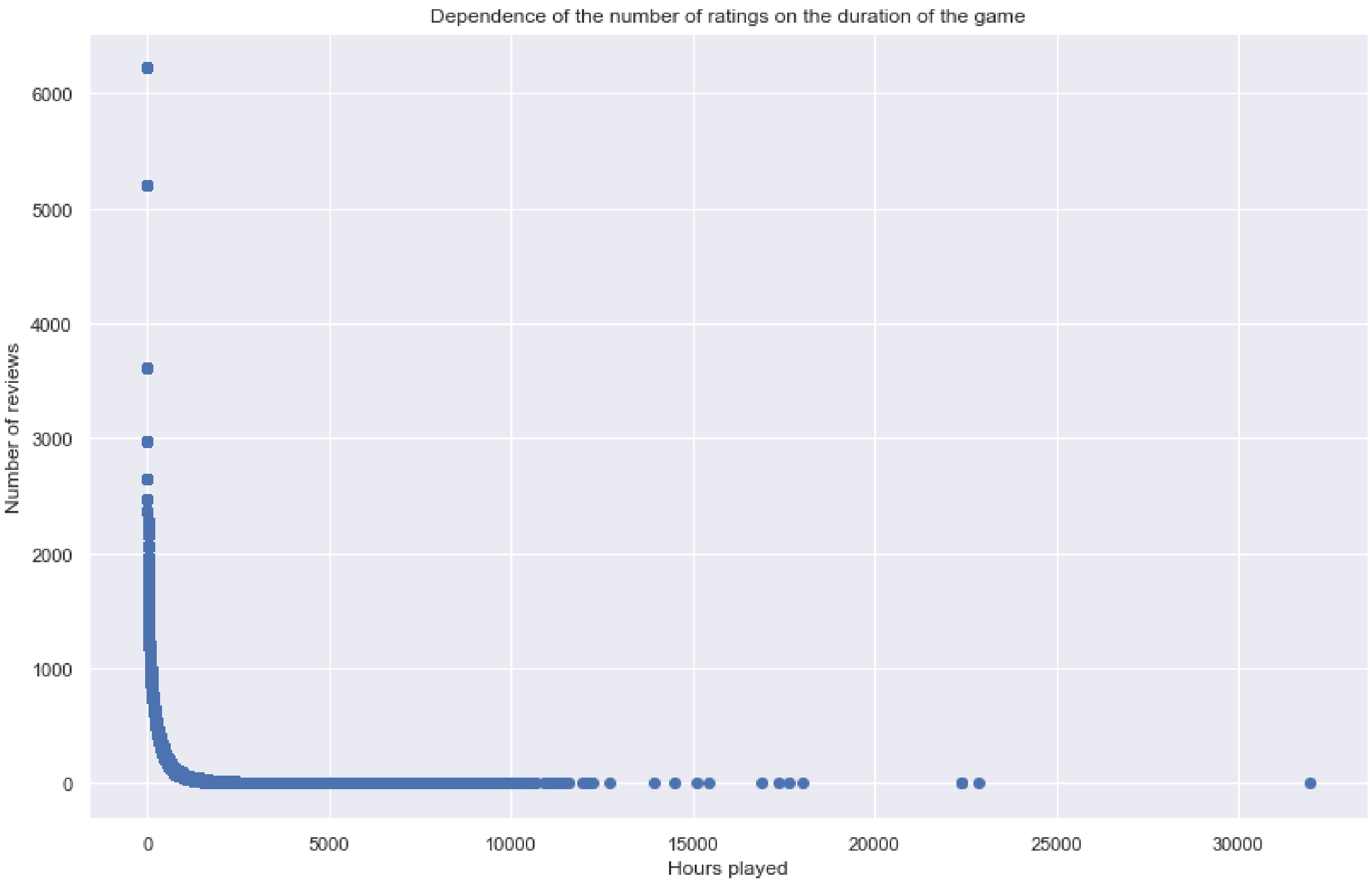
In [100]:

```
steam_review['hour_played_reviews'] = steam_review.groupby('hour_played')['hour_played']
.transform('count')
x = steam_review.hour_played
y = steam_review['hour_played_reviews']
```

```
fig = plt.figure(figsize = (13,8))
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
ax.scatter(x,y)
ax.set_title('Dependence of the number of ratings on the duration of the game')
ax.set_xlabel('Hours played')
ax.set_ylabel('Number of reviews')
```

Out[100]:

```
Text(0, 0.5, 'Number of reviews')
```



In [101]:

```
steam_review.dataframeName = 'steam_reviews.csv'
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have between 1 and 50 unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()

def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are
```

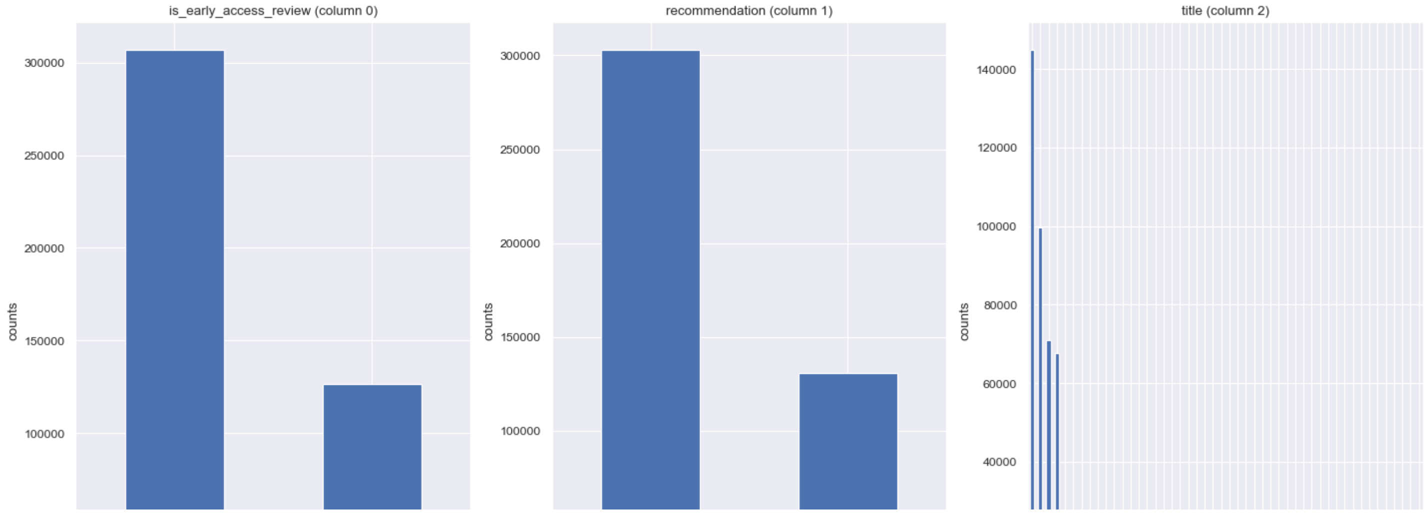
```
more than 1 unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()

def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

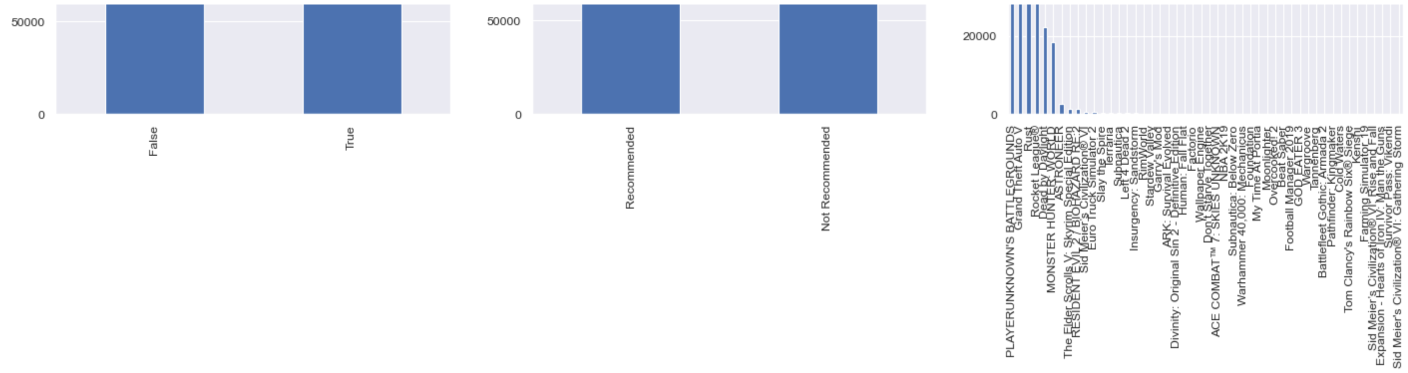
Plot per Column Distribution. Column name vs there couunt

In [102]:

```
plotPerColumnDistribution(steam_review, 10, 5)
```

```
<ipython-input-101-83bf43227d26>:10: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(nGraphRow, nGraphPerRow, i + 1)
<ipython-input-101-83bf43227d26>:10: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(nGraphRow, nGraphPerRow, i + 1)
<ipython-input-101-83bf43227d26>:10: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(nGraphRow, nGraphPerRow, i + 1)
```

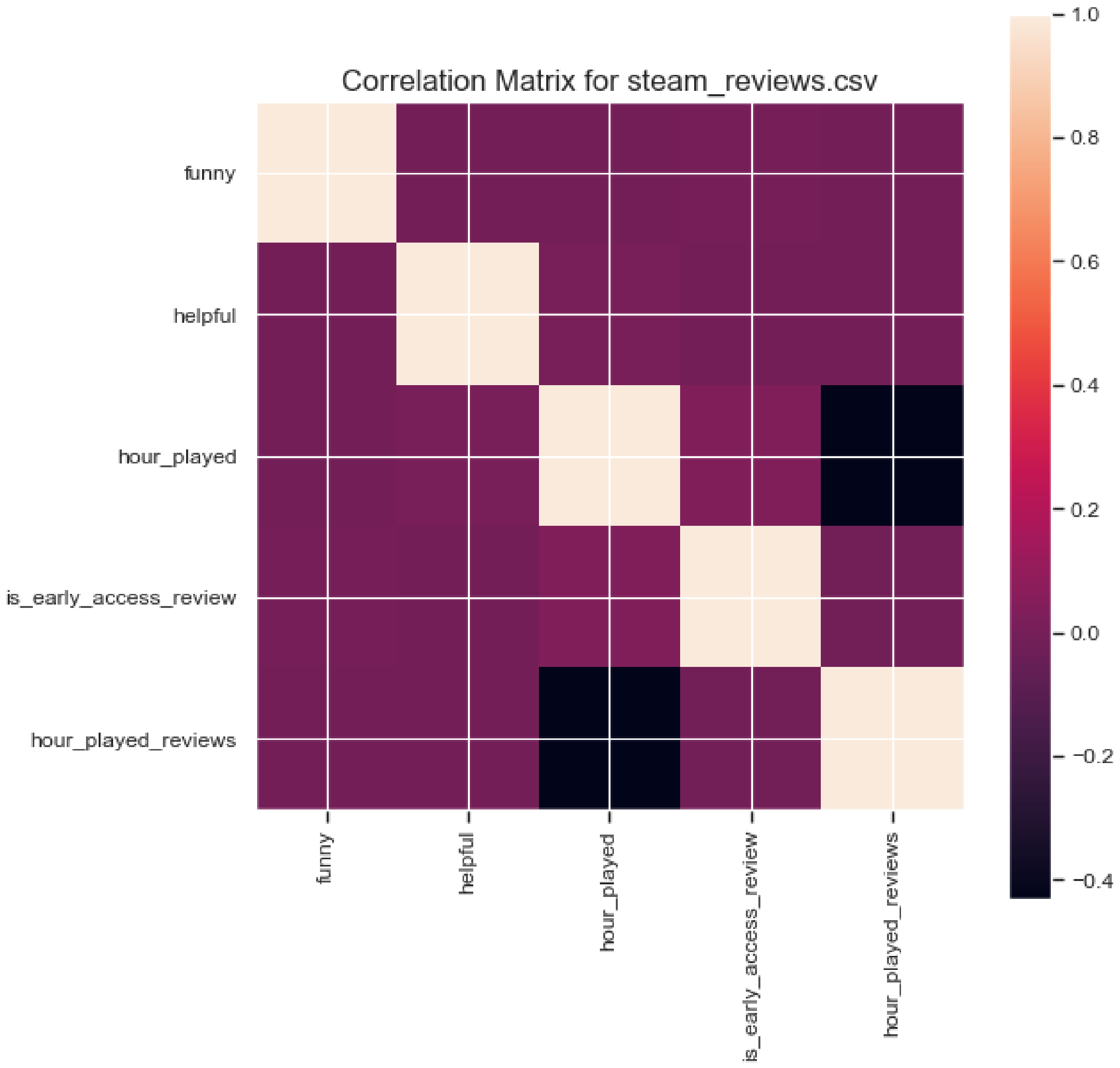




Correlation Matrix of the columns

In [103]:

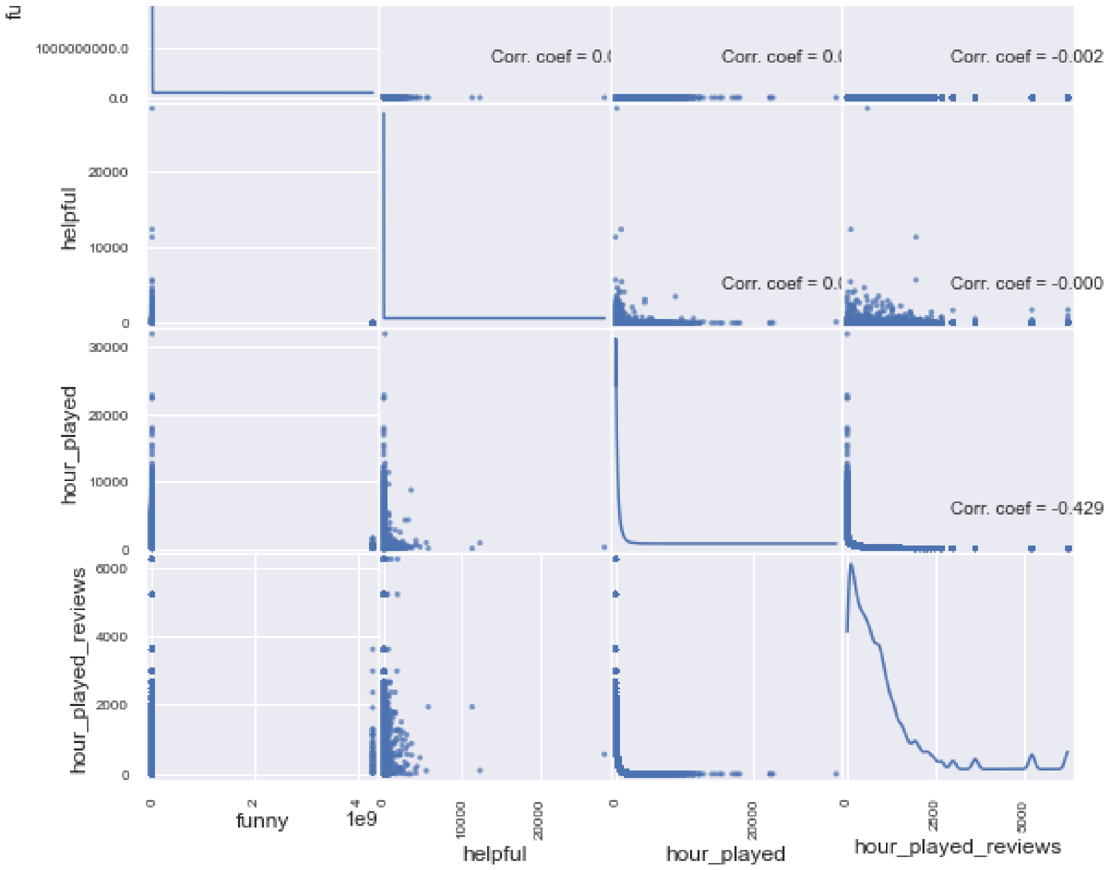
```
plotCorrelationMatrix(steam_review, 8)
```



Scatter Matrix of each column compared with other columns

In [104]:

```
plotScatterMatrix(steam_review, 9, 10)
```



Top 10 Most Reviewed Games

In [105]:

```
top_games = steam_review.title.value_counts()
print("Top 10 games are\n\n",steam_review.title.value_counts()[:10])
```

```
Top 10 games are

 PLAYERUNKNOWN'S BATTLEGROUNDS                      144846
Grand Theft Auto V                                 99677
Rust                                               70907
Rocket League®                                     67765
Dead by Daylight                                   22178
MONSTER HUNTER: WORLD                              18390
ASTRONEER                                           2658
The Elder Scrolls V: Skyrim Special Edition         1471
RESIDENT EVIL 2 / BIOHAZARD RE:2                    1384
Sid Meier’s Civilization® VI                         522
Name: title, dtype: int64
```

In [106]:

```
steam_review['review_length'] = steam_review.apply(lambda row: len(str(row['review'])), a
xis=1)

steam_review['recommendation_int'] = steam_review['recommendation'] == 'Recommended'
steam_review['recommendation_int'] = steam_review['recommendation_int'].astype(int)
```

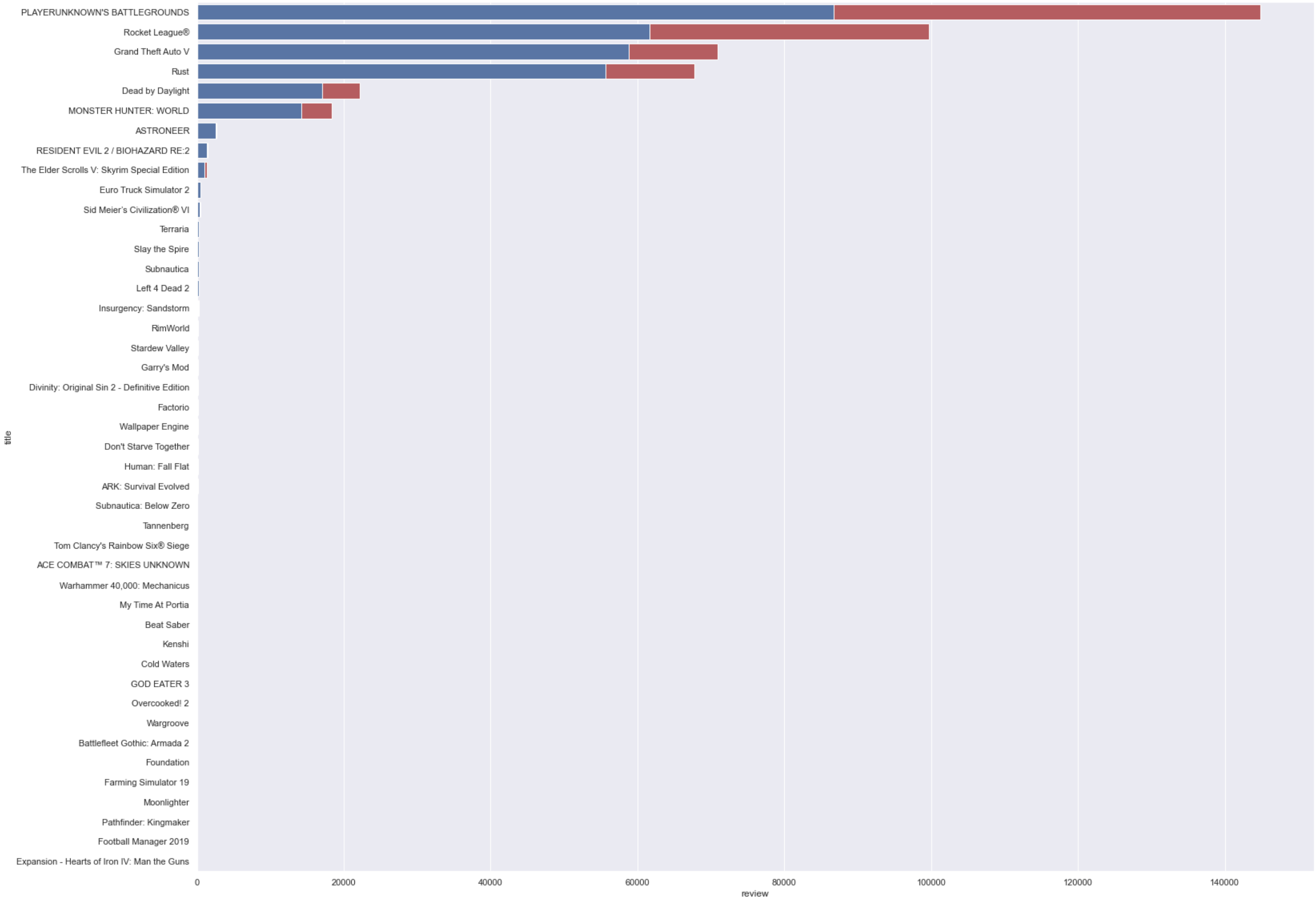
Plot of each game Vs the number of positive and negative reviews

In [107]:

```
reviews_count = steam_review.groupby(['title'])['review'].count().sort_values(ascending=
False)

reviews_count = reviews_count.reset_index()

sns.set(style="darkgrid")
plt.figure(figsize=(25,20))
sns.barplot(y=reviews_count['title'], x=reviews_count['review'], data=reviews_count,
             label="Total", color="r")

reviews_count_pos = steam_review.groupby(['title', 'recommendation_int'])['review'].coun
t().sort_values(ascending=False)
reviews_count_pos = reviews_count_pos.reset_index()
reviews_count_pos = reviews_count_pos[reviews_count_pos['recommendation_int'] == 1]
sns.barplot(y=reviews_count_pos['title'], x=reviews_count_pos['review'], data=reviews_co
unt_pos,
             label="Total", color="b")
```

Out[107]:

```
<AxesSubplot:xlabel='review', ylabel='title'>
```



Assigning integer value for recommended and not recommended as it will be easier for working.

In [108]:

```
steam_review['review_length'] = steam_review.apply(lambda row: len(str(row['review'])), a
xis=1)

steam_review['recommendation_int'] = steam_review['recommendation'] == 'Recommended'
steam_review['recommendation_int'] = steam_review['recommendation_int'].astype(int)
```

In [109]:

```
steam_review
```

Out[109]:

	date_posted	funny	helpful	hour_played	is_early_access_review	recommendation	review	title	hour_
0	2019-02-10	2	4	578	False	Recommended	&gt Played as German Reich&gt Declare war on B...	Expansion - Hearts of Iron IV: Man the Guns	
								Expansion - Hearts of	

	date_posted	funny	helpful	hour_played	is_early_access_review	recommendation	review	title	hour_
1	2019-02-10	0	0	184	False	Recommended	yes	Hearts of Iron IV: Man the Guns	
2	2019-02-07	0	0	892	False	Recommended	Very good game although a bit overpriced in my...	Expansion - Hearts of Iron IV: Man the Guns	
3	2018-06-14	126	1086	676	False	Recommended	Out of all the reviews I wrote This one is pro...	Dead by Daylight	
4	2017-06-20	85	2139	612	False	Recommended	Disclaimer I survivor main. I play games for f...	Dead by Daylight	
...	...	...	...	...	...	...	...	...	
434886	2018-11-17	1	37	10	False	Recommended	YOUR FLESH WILL ROT AND DECAY.STEEL IS IMMORTA...	Warhammer 40,000: Mechanicus	
434887	2018-11-17	3	41	38	False	Recommended	Domini and Dominae I believe what we are deali...	Warhammer 40,000: Mechanicus	
434888	2018-11-20	0	0	36	False	Recommended	First off if you like X Com style of games you...	Warhammer 40,000: Mechanicus	
434889	2018-11-18	1	44	12	False	Recommended	As a disclaimer I'm an AdMech player on the ta...	Warhammer 40,000: Mechanicus	
434890	2019-01-21	1	28	20	False	Recommended	Don't listen to people who claim it's the game...	Warhammer 40,000: Mechanicus	

433375 rows × 11 columns

In [110]:

```
steam_review.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 433375 entries, 0 to 434890
Data columns (total 11 columns):
 #   Column                  Non-Null Count   Dtype 
---  ------                  --------------   ----- 
 0   date_posted             433375 non-null  object
 1   funny                   433375 non-null  int64 
 2   helpful                 433375 non-null  int64 
 3   hour_played             433375 non-null  int64 
 4   is_early_access_review  433375 non-null  bool  
 5   recommendation          433375 non-null  object
 6   review                  433375 non-null  object
 7   title                   433375 non-null  object
 8   hour_played_reviews     433375 non-null  int64 
 9   review_length           433375 non-null  int64 
 10  recommendation_int      433375 non-null  int32 
dtypes: bool(1), int32(1), int64(5), object(4)
memory usage: 35.1+ MB
```

In [111]:

```
steam_review.duplicated().sum()
steam_review.duplicated().sum()
```

Out[111]:

```
0
```

Data Cleaning

Converting all the reviews to lower case and removing special characters

In [112]:

```
steam_review['review'] = [review.strip().lower() for review in steam_review['review']]
```

In [113]:

```
steam_review['review'] = steam_review['review'].replace(r"[^a-zA-Z\d\_\+\-\'\.\/\s]+", '
', regex = True)

steam_review['review'] = steam_review['review'].replace(["./ ", "' ", " '"], " ", regex
= True)
```

In [114]:

```
steam_review.head()
```

Out[114]:

	date_posted	funny	helpful	hour_played	is_early_access_review	recommendation	review	title	hour_played_revi
0	2019-02-10	2	4	578	False	Recommended	gt played as german reich gt declare war on b...	Expansion - Hearts of Iron IV: Man the Guns	
1	2019-02-10	0	0	184	False	Recommended	yes.	Expansion - Hearts of Iron IV: Man the Guns	
2	2019-02-07	0	0	892	False	Recommended	very good game although a bit overpriced in my...	Expansion - Hearts of Iron IV: Man the Guns	
3	2018-06-14	126	1086	676	False	Recommended	out of all the reviews i wrote this one is pro...	Dead by Daylight	
4	2017-06-20	85	2139	612	False	Recommended	disclaimer i survivor main. i play games for f...	Dead by Daylight	

Cleaned Data

In [115]:

```
steam_review.isnull().sum()
```

Out[115]:

```
date_posted                 0
funny                       0
funny                      0
helpful                    0
hour_played                0
is_early_access_review     0
recommendation             0
review                     0
title                      0
hour_played_reviews        0
review_length              0
recommendation_int         0
dtype: int64
```

Building Classification Model

We will be diving the data into Train data, Test data and Dev data using train_test_split.

In [116]:

```
data_frame = steam_review
```

In [117]:

```
data_frame, test_data = train_test_split(data_frame, test_size=0.20)
```

In [118]:

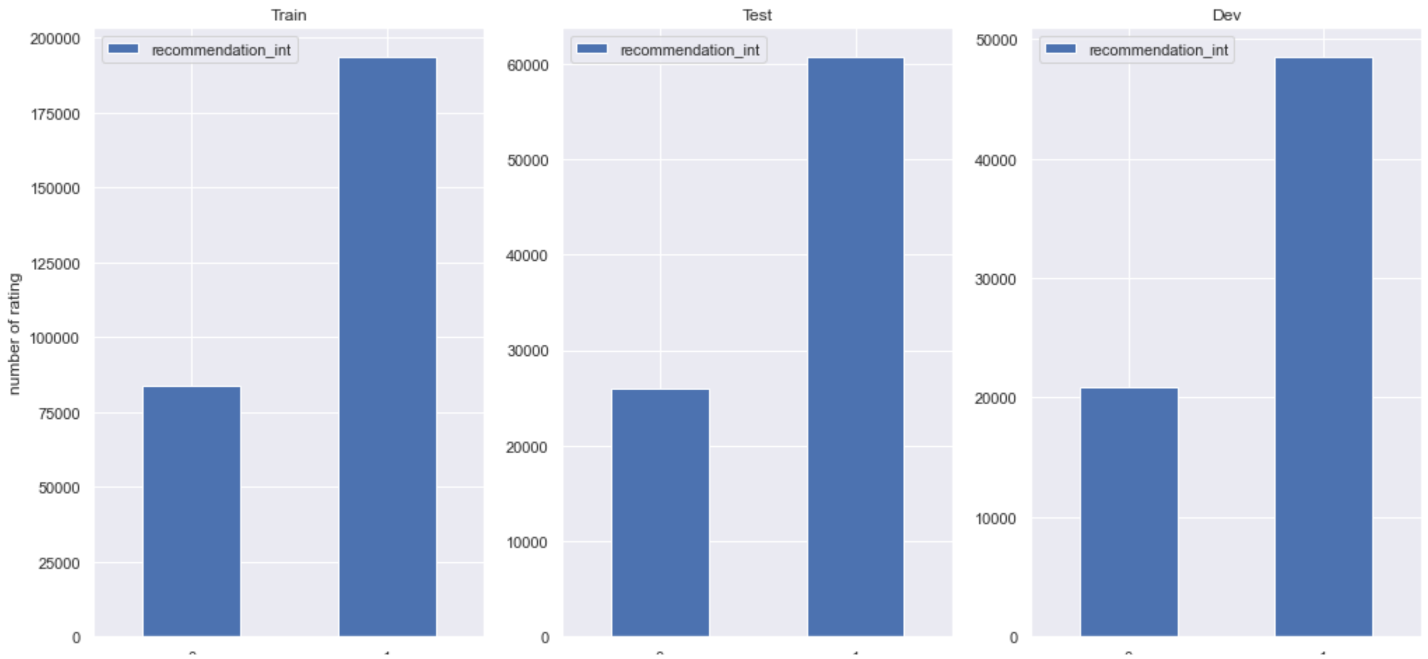
```
train_data, dev_data = train_test_split(data_frame, test_size=0.20)
```

In [119]:

```
fig, axes = plt.subplots(ncols=3)

plot1 = train_data.recommendation_int.value_counts().sort_index().plot(kind='bar', legen
d=True, rot=0, ax=axes[0])
plot1.set_title("Train")
plot1.set_xlabel("Rating")
plot1.set_ylabel("number of rating")
plot2 = test_data.recommendation_int.value_counts().sort_index().plot(kind='bar', legend
=True, rot=0, ax=axes[1])
plot2.set_title("Test")
plot2.set_xlabel("Rating")
plot3 = dev_data.recommendation_int.value_counts().sort_index().plot(kind='bar', legend=
True, rot=0,figsize=(17, 8))
plot3.set_title("Dev")
plot3.set_xlabel("Rating")
```

Out[119]:

```
Text(0.5, 0, 'Rating')
```



For text analysis the best classifiers are Naive Bayes, Support Vector Machine and Random Forest Classifier

But before we train our model we have to convert reviews to TF IDF form

The formula that is used to compute the tf-idf is

tf-idf(t, d) = tf(t, d) * idf(t) and idf is computed as idf(d, t) = log [(1 + n) / (1 + df(d, t))] + 1

In [120]:

```
tfidf = TfidfVectorizer(max_features=1000)

# transforming traing data to tfidf form arrays. we can use this to train any model
x_train = tfidf.fit_transform(train_data['review']).toarray()
y_train = np.asarray(train_data['recommendation_int'])
```

In [121]:

```
x_dev = tfidf.transform(dev_data['review']).toarray()
y_dev = np.asarray(dev_data['recommendation_int'])
```

Trying out Naive bayes Model as this is a text analysis we will use multinomial Naive Bayes.

In [122]:

```
alpha = [0.01, .1, 0.5, 1, 2, 3]
multinomial_nb = {}
mse_multinomial_nb = {}
accuracy_multinomial_nb = {}
y_dev_pred_multinomial_nb = {}
for a in alpha:
    multinomial_nb[a] = MultinomialNB(alpha=a)
    multinomial_nb[a].fit(x_train, y_train)

    y_dev_pred_multinomial_nb[a] = (multinomial_nb[a].predict(x_dev))

    # Calculate the Mean Squared Error and Accuracy
    mse_multinomial_nb[a] = mean_squared_error(y_dev, y_dev_pred_multinomial_nb[a])
    accuracy_multinomial_nb[a] = accuracy_score(y_dev, y_dev_pred_multinomial_nb[a])*100

    # Print the Mean Squared Error and Accuracy
    print(f'Mean Squared Error = {mse_multinomial_nb[a]} for alpha = {a}')
    print(f'Accuracy = {accuracy_multinomial_nb[a]} for alpha = {a}')
```

```
Mean Squared Error = 0.16649841361407558 for alpha = 0.01
Accuracy = 83.35015863859245 for alpha = 0.01
Mean Squared Error = 0.16654167868474185 for alpha = 0.1
Accuracy = 83.34583213152581 for alpha = 0.1
Mean Squared Error = 0.16668589558696278 for alpha = 0.5
Accuracy = 83.33141044130372 for alpha = 0.5
Mean Squared Error = 0.16687337755985002 for alpha = 1
Accuracy = 83.312662244015 for alpha = 1
Mean Squared Error = 0.16708970291318143 for alpha = 2
Accuracy = 83.29102970868186 for alpha = 2
Mean Squared Error = 0.16704643784251513 for alpha = 3
Accuracy = 83.29535621574848 for alpha = 3
```

Alpha at 0.1 gave us the best accuracy

In [123]:

```
mse = {}
accuracy = {}
model = {}
model["MNB"] = multinomial_nb
```

```
mse["MNB"] = mse_multinomial_nb
accuracy["MNB"] = accuracy_multinomial_nb
```

Support Vector Machines (SVM)

In [124]:

```
C = [0.01, 0.1, 1, 10, 100, 1000]
svm = {}
mse_svm = {}
accuracy_svm = {}
y_dev_pred_svm = {}
for c in C:
    svm[c] = LinearSVC(C=c, dual = False)
    svm[c].fit(x_train, y_train)

    y_dev_pred_svm[c] = (svm[c].predict(x_dev))

    mse_svm[c] = mean_squared_error(y_dev, y_dev_pred_svm[c])
    accuracy_svm[c] = accuracy_score(y_dev, y_dev_pred_svm[c])*100

    print(f'Mean Squared Error = {mse_svm[c]} for C = {c}')
    print(f'Accuracy = {accuracy_svm[c]} for C = {c}')
```

```
Mean Squared Error = 0.1461494087107009 for C = 0.01
Accuracy = 85.38505912892991 for C = 0.01
Mean Squared Error = 0.14441880588404962 for C = 0.1
Accuracy = 85.55811941159503 for C = 0.1
Mean Squared Error = 0.1444332275742717 for C = 1
Accuracy = 85.55667724257283 for C = 1
Mean Squared Error = 0.14447649264493798 for C = 10
Accuracy = 85.5523507355062 for C = 10
Mean Squared Error = 0.1444620709547159 for C = 100
Accuracy = 85.55379290452841 for C = 100
Mean Squared Error = 0.1444620709547159 for C = 1000
Accuracy = 85.55379290452841 for C = 1000
```

Combining 2 models to test the accuracy.

In [125]:

```
model['SVM'] = svm
mse['SVM'] = mse_svm
accuracy['SVM'] = accuracy_svm
```

In [126]:

```
def svmmnb(x):
    x_array = tfidf.transform(x).toarray()
    y_pred_dict = {}

    for a in alpha:
        y_pred = model['MNB'][a].predict(x_array)[0]
        if y_pred in y_pred_dict:
            y_pred_dict[y_pred] += (accuracy['MNB'][a])
        else:
            y_pred_dict[y_pred] = (accuracy['MNB'][a])

    for c in C:
        y_pred = model['SVM'][c].predict(x_array)[0]
        if y_pred in y_pred_dict:
            y_pred_dict[y_pred] += (accuracy['SVM'][c])
        else:
            y_pred_dict[y_pred] = (accuracy['SVM'][c])

    inverse = [(value, key) for key, value in y_pred_dict.items()]
    return max(inverse)[1]
```

In [127]:

```
y_dev_pred_ens = []

for index, row in dev_data.iterrows():
    y_dev_pred_ens.append(svmmnb([row['review']]))


mse_ens = mean_squared_error(y_dev, y_dev_pred_ens)
accuracy_ens = accuracy_score(y_dev, y_dev_pred_ens)*100

print(f'Mean Squared Error = {mse_ens}')
print(f'Accuracy = {accuracy_ens}')
```

```
Mean Squared Error = 0.1459186616671474
Accuracy = 85.40813383328526
```

When SVM and MNB were combined I got an accuracy of 85.4081%

Trying Random Forest Classifier

In [128]:

```
estimators = [10, 50]
rfc = {}
mse_rfc = {}
accuracy_rfc = {}
y_dev_pred_rfc = {}
for n in estimators:
    rfc[n] = RandomForestClassifier(max_depth=25, n_estimators=n)
    rfc[n].fit(x_train, y_train)

    y_dev_pred_rfc[n] = (rfc[n].predict(x_dev))

    mse_rfc[n] = mean_squared_error(y_dev, y_dev_pred_rfc[n])
    accuracy_rfc[n] = accuracy_score(y_dev, y_dev_pred_rfc[n])*100

    print(f'Mean Squared Error = {mse_rfc[n]} for n = {n}')
    print(f'Accuracy = {accuracy_rfc[n]} for n = {n}')
```

```
Mean Squared Error = 0.20250937409864436 for n = 10
Accuracy = 79.74906259013557 for n = 10
Mean Squared Error = 0.19750504759157772 for n = 50
Accuracy = 80.24949524084222 for n = 50
```

This gave me an accuracy of 80.2494% which is very less compared to the other 2 models. We got good accuracy for SVM.

In [129]:

```
x_test = tfidf.transform(test_data['review']).toarray()
y_test = np.asarray(test_data['recommendation_int'])
```

Using the test data to test our best model

In [130]:

```
c=1
y_test_pred = (svm[c].predict(x_test))

mse_test = mean_squared_error(y_test, y_test_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)*100

print(f'Mean Squared Error test = {mse_test}')
print(f'Accuracy test = {accuracy_test}')
```

```
Mean Squared Error test = 0.14402076723391982
Accuracy test = 85.59792327660803
```

GUI

In [131]:

```python
def rating():
    x = textfield_label.get()
    results.delete("all")
    value = svmmnb([x])
    print(value)
    if value == 0:
        text = "Negative"
    else:
        text = "Positive"
    results.create_text(200, 80, text="The comment is "+text, font='Arial 20', fill='white')
    return value

window = Tk()
window.geometry("500x500")
window.title("Game Recommender")
window.config(bg="black")
head = Label(window, text="Game Review Classifier", bg="black", foreground="white", font=("Arial", 25)).pack()
Label(window, text="Comment", bg="black", foreground="white", font=("Arial", 10)).place(x=100, y=80)
Label(window, text="______________________________", bg="black", foreground="white").place(x=173, y=90)
textfield_label = Entry(window, bg="black", foreground="white", border=0, insertbackground="white")
textfield_label.focus()
textfield_label.place(x=175, y=80, height=20, width=150)
output = Label(window, bg="black", foreground="white", font=("Arial", 25)).place(x=50, y=250)
search = Button(window, text="Rate", command=rating).place(x=200, y=150, width=100)
print(search)
results = tkinter.Canvas(bg='black', width='400', height='200', highlightthickness=0)
results.place(x=50, y=250)
window.mainloop()
```

```
None
0
1
```

In [131]:

```

```