Analyzing Video Game Ratings: A Machine Learning Approach Using Random Forest and Decision Tree Models

The project aims to gain insights into the factors influencing game ratings and create predictive models using machine-learning approaches. It combines the RAWG Video Games Database API with a Python application to collect data about video games and processes it for analysis. Two models, Decision Trees, and Random Forests, are trained on this data to uncover the relationships between specific features and game ratings.

We assess the models' performance using Mean Squared Error (MSE) and R-squared (R2) scores, which provide quantitative measures of accuracy. Additionally, we analyze the importance of different features in influencing game ratings. The study also presents the results of regression analysis, including R-squared and Adj. R-squared values, which quantify the extent to which independent variables explain variability in the ratings.

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import numpy as np
from scipy.stats import chi2_contingency
from scipy.stats import f_oneway
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer
from ydata_profiling import ProfileReport

data = pd.read_excel('game_data_new.xlsx')
```

In [2]:

```
data.columns
```

Out[2]:

```
Index(['Unnamed: 0', 'id', 'slug', 'name', 'released', 'tba',
       'background_image', 'rating', 'rating_top', 'reviews_text_count',
       'metacritic', 'playtime', 'suggestions_count', 'updated', 'user_game',
       'reviews_count'],
      dtype='object')
```

In [3]:

```
X = data[['reviews_text_count', 'metacritic', 'playtime', 'suggestions_count']]
y = data['rating']



print("Basic Information about the Dataset:")
print(data.info())


print("\nSample Data (first 5 rows):")
print(data.head())


print("\nMissing Values:")
print(data.isnull().sum())
```

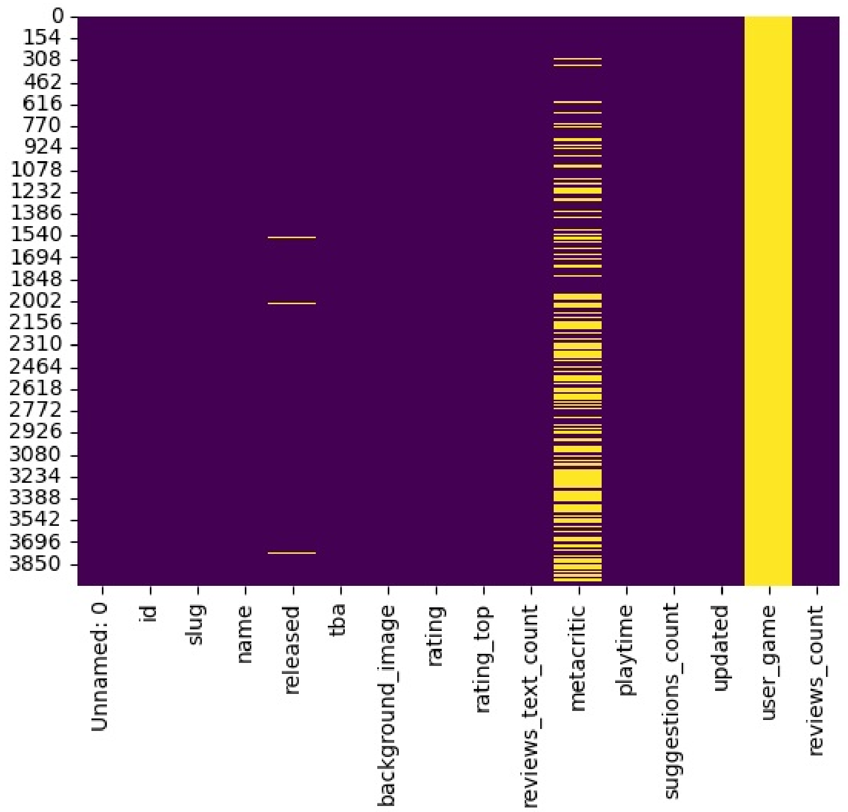
```
Basic Information about the Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          4000 non-null   int64  
 1   id                  4000 non-null   int64  
 2   slug                4000 non-null   object 
 3   name                4000 non-null   object 
 4   released            3976 non-null   object 
 5   tba                 4000 non-null   int64  
 6   background_image    3995 non-null   object 
 7   rating              4000 non-null   float64
 8   rating_top          4000 non-null   int64  
 9   reviews_text_count  4000 non-null   int64  
 10  metacritic          2649 non-null   float64
 11  playtime            4000 non-null   int64  
 12  suggestions_count   4000 non-null   int64  
 13  updated             4000 non-null   object 
 14  user_game           0 non-null      float64
 15  reviews_count       4000 non-null   int64  
dtypes: float64(3), int64(8), object(5)
memory usage: 500.1+ KB
None

Sample Data (first 5 rows):
   Unnamed: 0  id                             slug  \
0           1   1                grand-theft-auto-v   
1           2   2           the-witcher-3-wild-hunt   
2           3   3                         portal-2   
3           4   4  counter-strike-global-offensive   
4           5   5                      tomb-raider   

                            name    released  tba  \
0                 Grand Theft Auto V  2013-09-17    0   
1           The Witcher 3: Wild Hunt  2015-05-18    0   
2                         Portal 2  2011-04-18    0   
3  Counter-Strike: Global Offensive  2012-08-21    0   
4                Tomb Raider (2013)  2013-03-05    0   

                                    background_image  rating  rating_top  \
0  https://media.rawg.io/media/games/20a/20aa03a1...    4.47           5   
1  https://media.rawg.io/media/games/618/618c2031...    4.66           5   
2  https://media.rawg.io/media/games/2ba/2bac0e87...    4.61           5   
3  https://media.rawg.io/media/games/736/73619bd3...    3.57           4   
4  https://media.rawg.io/media/games/021/021c4e21...    4.05           4   

   reviews_text_count  metacritic  playtime  suggestions_count  \
0                  57        92.0        74                421   
1                  69        92.0        45                671   
2                  32        95.0        11                545   
3                  24        81.0        65                587   
4                  12        86.0        10                643   

               updated  user_game  reviews_count  
0  2023-09-05T08:10:07        NaN           6610  
1  2023-09-06T16:21:13        NaN           6332  
2  2023-09-05T19:33:39        NaN           5470  
3  2023-09-06T04:07:56        NaN           3369  
4  2023-09-05T11:56:12        NaN           3781  

Missing Values:
Unnamed: 0               0
id                       0
slug                     0
name                     0
released                24
tba                      0
background_image         5
rating                   0
rating_top               0
reviews_text_count       0
metacritic            1351
playtime                 0
suggestions_count        0
updated                  0
user_game             4000
reviews_count            0
dtype: int64
```

In [4]:

```
sns.heatmap(data.isna(), cbar=False, cmap='viridis')
```

Out[4]:

```
<Axes: >
```



In [5]:
```
selected_columns = data.select_dtypes(include=['int64', 'float64'])

profile = ProfileReport(selected_columns, title='Descriptive Statistics Report', explorative=True)

# Generate and display the report
profile.to_widgets()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render widgets:   0%|          | 0/1 [00:00<?, ?it/s]
VBox(children=(Tab(children=(Tab(children=(GridBox(children=(VBox(children=(GridspecLayout(children=(HTML(valu…
```

In [19]:
```
data.describe()
```

Out[19]:

	Unnamed: 0	id	tba	rating	rating_top	reviews_text_count	metacritic	playtime	suggestions_count	us
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.00000	4000.000000	4000.000000	4000.000000	4000.000000	
mean	2000.500000	2000.500000	0.002250	3.352935	3.51250	2.213750	76.585504	4.567750	410.594000	
std	1154.844867	1154.844867	0.047387	0.736663	1.15874	4.782602	7.748950	16.738024	178.978419	
min	1.000000	1.000000	0.000000	0.000000	0.00000	0.000000	23.000000	0.000000	0.000000	
25%	1000.750000	1000.750000	0.000000	2.910000	3.00000	0.000000	75.000000	1.000000	279.000000	
50%	2000.500000	2000.500000	0.000000	3.460000	4.00000	1.000000	76.585504	3.000000	415.000000	
75%	3000.250000	3000.250000	0.000000	3.920000	4.00000	2.000000	80.000000	4.000000	534.000000	
max	4000.000000	4000.000000	1.000000	4.800000	5.00000	72.000000	99.000000	900.000000	1668.000000	

In [6]:
```
# Handling missing values using a simple imputer
X = X.replace([np.inf, -np.inf], np.nan)
X = X.dropna()
X = sm.add_constant(X)


model = sm.OLS(y[X.index], X).fit()
summary = model.summary()

# Calculate Mean Squared Error (MSE)
y_pred = model.predict(X)
mse = mean_squared_error(y[X.index], y_pred)

# Chi-Square Test
contingency_table = pd.crosstab(data['reviews_text_count'], data['playtime'])
chi2, p, _, _ = chi2_contingency(contingency_table)

# ANOVA Test
```

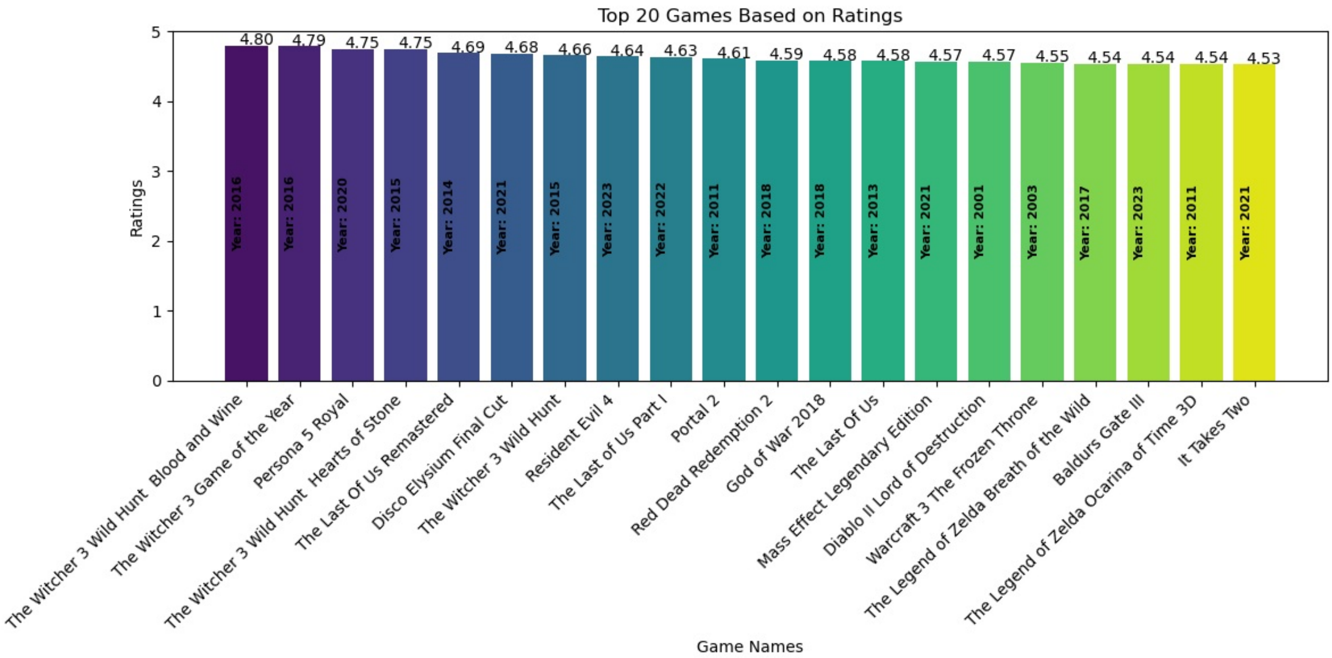
```
groups = data.groupby('metacritic')['suggestions_count'].apply(list)
f_statistic, p_value = f_oneway(*groups)
```

In [7]:
```
# Print the results
print("Summary Statistics:")
print(summary)
print("\nMean Squared Error (MSE):", mse)
print("\nChi-Square Test (reviews_text_count vs. playtime):")
print("Chi-Square Value:", chi2)
print("p-value:", p)
print("\nANOVA Test (metacritic vs. suggestions_count):")
print("F-statistic:", f_statistic)
print("p-value:", p_value)
```

```
Summary Statistics:
                            OLS Regression Results
==============================================================================
Dep. Variable:                 rating   R-squared:                       0.434
Model:                            OLS   Adj. R-squared:                  0.433
Method:                 Least Squares   F-statistic:                     506.2
Date:                Sat, 09 Sep 2023   Prob (F-statistic):               0.00
Time:                        20:38:16   Log-Likelihood:                -1523.8
No. Observations:                2649   AIC:                             3058.
Df Residuals:                    2644   BIC:                             3087.
Df Model:                           4
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.9202      0.075     12.288      0.000       0.773       1.067
reviews_text_count    0.0194      0.002     11.717      0.000       0.016       0.023
metacritic            0.0336      0.001     35.849      0.000       0.032       0.035
playtime              0.0029      0.001      2.479      0.013       0.001       0.005
suggestions_count  5.334e-05   4.81e-05      1.109      0.267   -4.09e-05       0.000
==============================================================================
Omnibus:                      183.127   Durbin-Watson:                   1.964
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              248.670
Skew:                          -0.600   Prob(JB):                     1.00e-54
Kurtosis:                       3.902   Cond. No.                     4.34e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.34e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Mean Squared Error (MSE): 0.18499388124793303

Chi-Square Test (reviews_text_count vs. playtime):
Chi-Square Value: 21323.421499697994
p-value: 0.0

ANOVA Test (metacritic vs. suggestions_count):
F-statistic: 1.5565091064256782
p-value: 0.003768003756472688
```

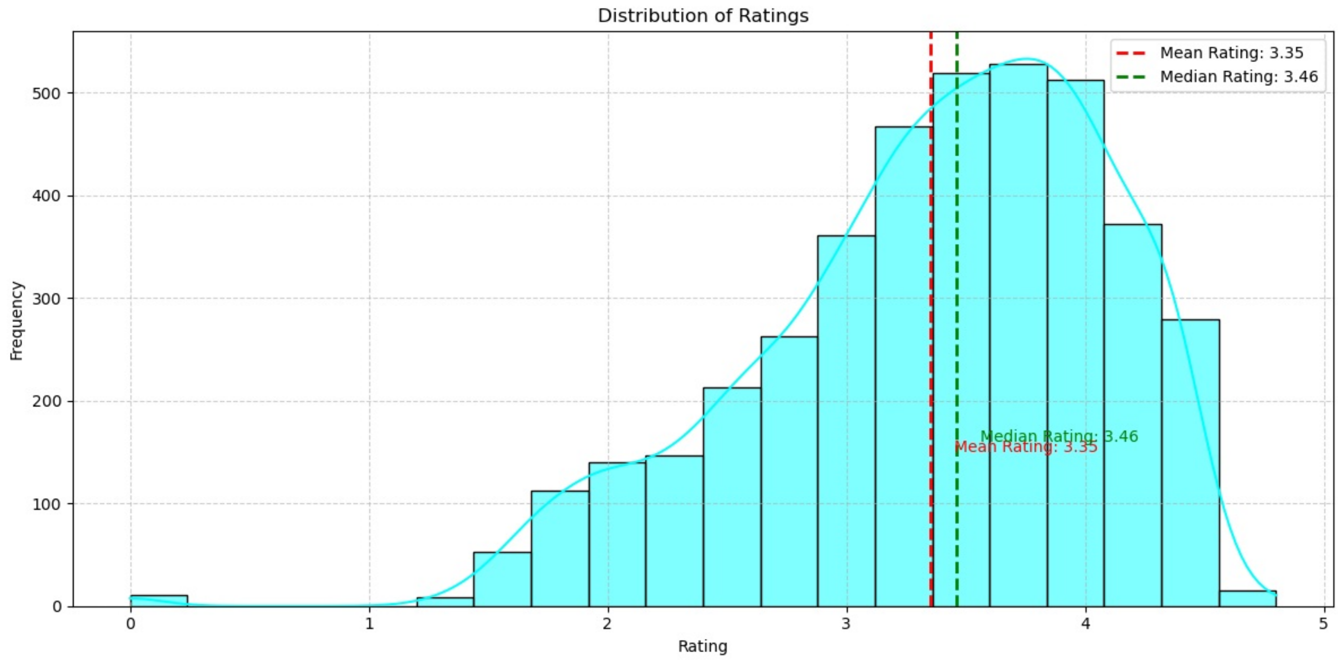
EDA: Create visualizations

In [8]:
```
data['name'] = data['name'].str.replace('[^a-zA-Z0-9\s]', '', regex=True)


data['rating'] = pd.to_numeric(data['rating'], errors='coerce')
top_20_games = data.nlargest(20, 'rating')

colors = sns.color_palette('viridis', n_colors=len(top_20_games))


plt.figure(figsize=(12, 6))
bars = plt.bar(top_20_games['name'], top_20_games['rating'], color=colors)


for i, (bar, rating, year) in enumerate(np.array(list(zip(bars, top_20_games['rating'], top_20_games['released'
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.15, bar.get_height() + 0.01, f'{rating:.2f}', fontsize=10)
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.15, bar.get_height() / 2, f'Year: {pd.to_datetime(year).year

plt.xlabel('Game Names')
plt.ylabel('Ratings')
plt.title('Top 20 Games Based on Ratings')
plt.ylim(0, 5)
plt.xticks(rotation=45, ha='right')

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



In [9]:

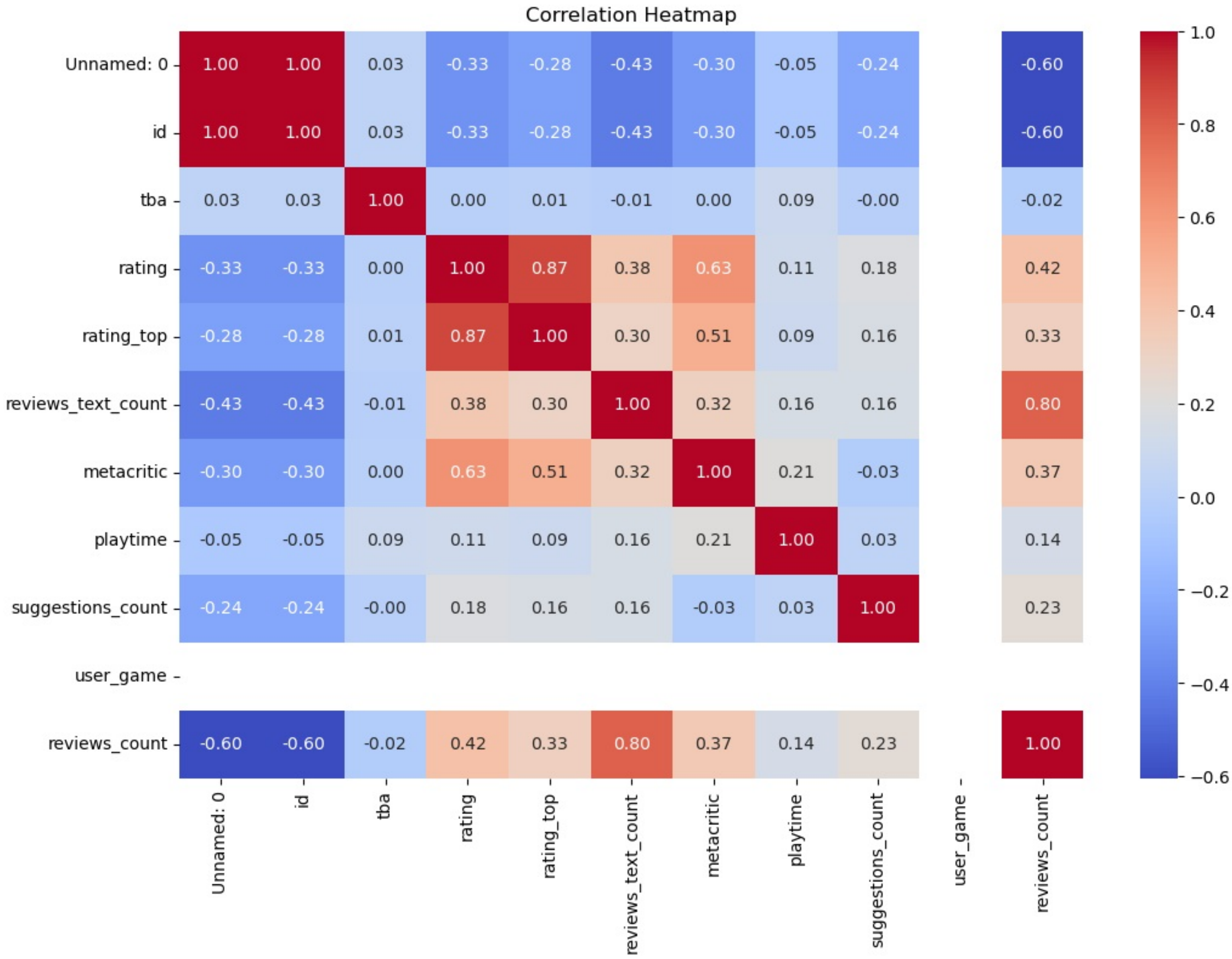
```python
plt.figure(figsize=(12, 6))
sns.histplot(data['rating'], bins=20, kde=True, color='cyan')
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')

mean_rating = data['rating'].mean()
median_rating = data['rating'].median()
plt.axvline(mean_rating, color='red', linestyle='dashed', linewidth=2, label=f'Mean Rating: {mean_rating:.2f}')
plt.axvline(median_rating, color='green', linestyle='dashed', linewidth=2, label=f'Median Rating: {median_ratin


plt.text(mean_rating + 0.1, 150, f'Mean Rating: {mean_rating:.2f}', color='red')
plt.text(median_rating + 0.1, 160, f'Median Rating: {median_rating:.2f}', color='green')

plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



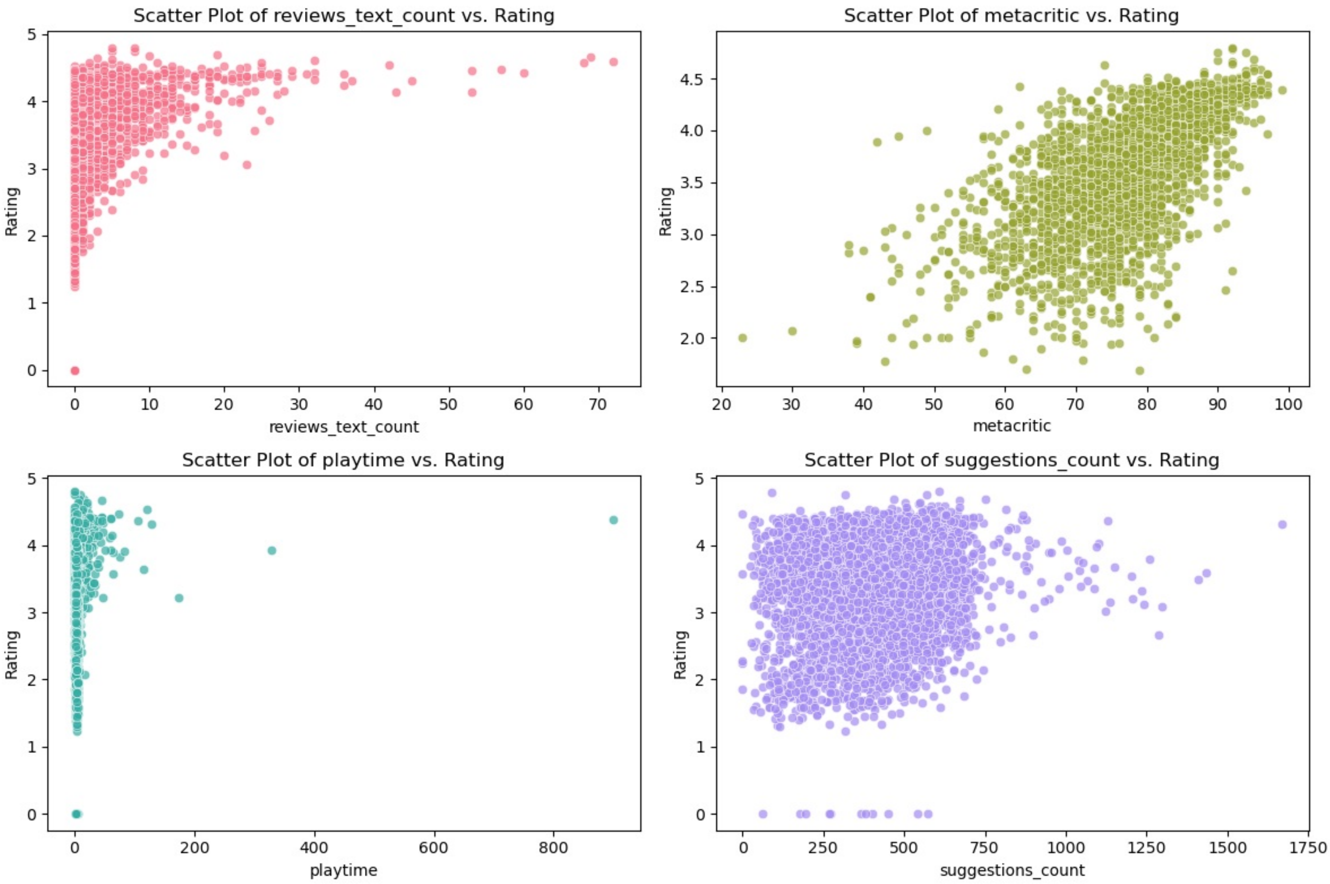
In [10]:

```python
# Correlation heatmap
correlation_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

```
C:\Users\PERSONAL\AppData\Local\Temp\ipykernel_13328\3060984138.py:2: FutureWarning: The default value of numer
ic_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid colum
ns or specify the value of numeric_only to silence this warning.
  correlation_matrix = data.corr()
```

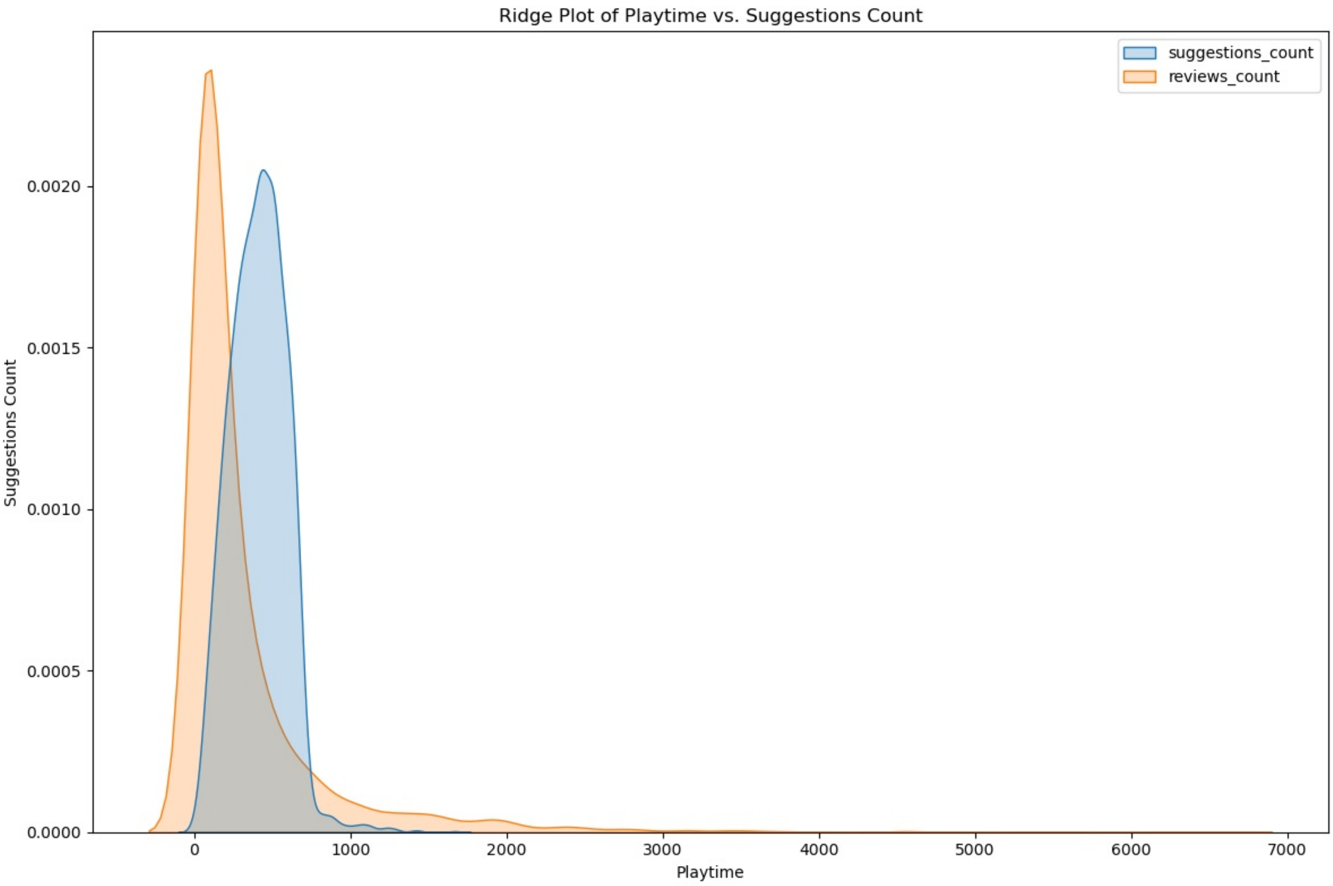


In [11]:
```
numeric_features = ['reviews_text_count', 'metacritic', 'playtime', 'suggestions_count']

palette = sns.color_palette("husl", len(numeric_features))

plt.figure(figsize=(12, 8))
for i, feature in enumerate(numeric_features, start=1):
    plt.subplot(2, 2, i)
    sns.scatterplot(x=feature, y='rating', data=data, alpha=0.7, color=palette[i-1])
    plt.title(f'Scatter Plot of {feature} vs. Rating')
    plt.xlabel(feature)
    plt.ylabel('Rating')

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



In [12]:

```
columns_for_ridge = ['suggestions_count', 'reviews_count']

plt.figure(figsize=(12, 8))
sns.kdeplot(data=data[columns_for_ridge], fill=True, common_norm=False)

plt.title('Ridge Plot of Playtime vs. Suggestions Count')
plt.xlabel('Playtime')
plt.ylabel('Suggestions Count')

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

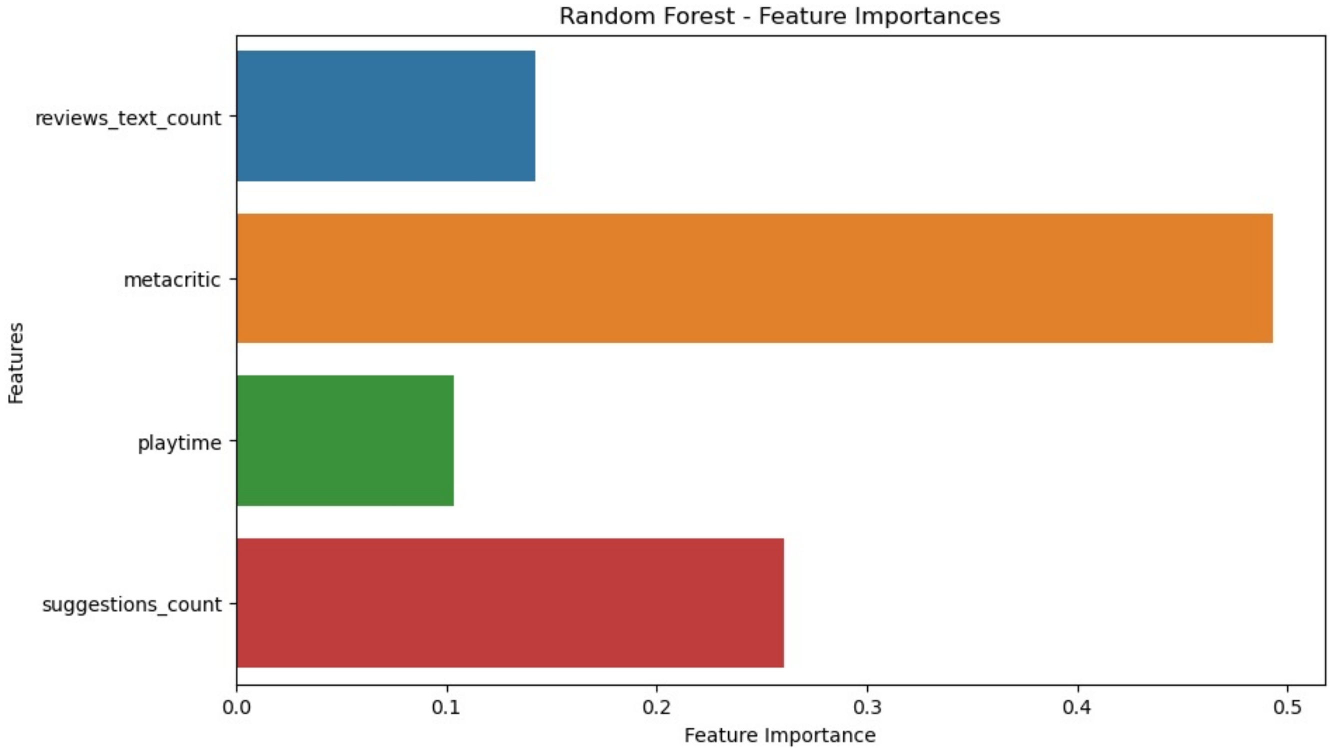


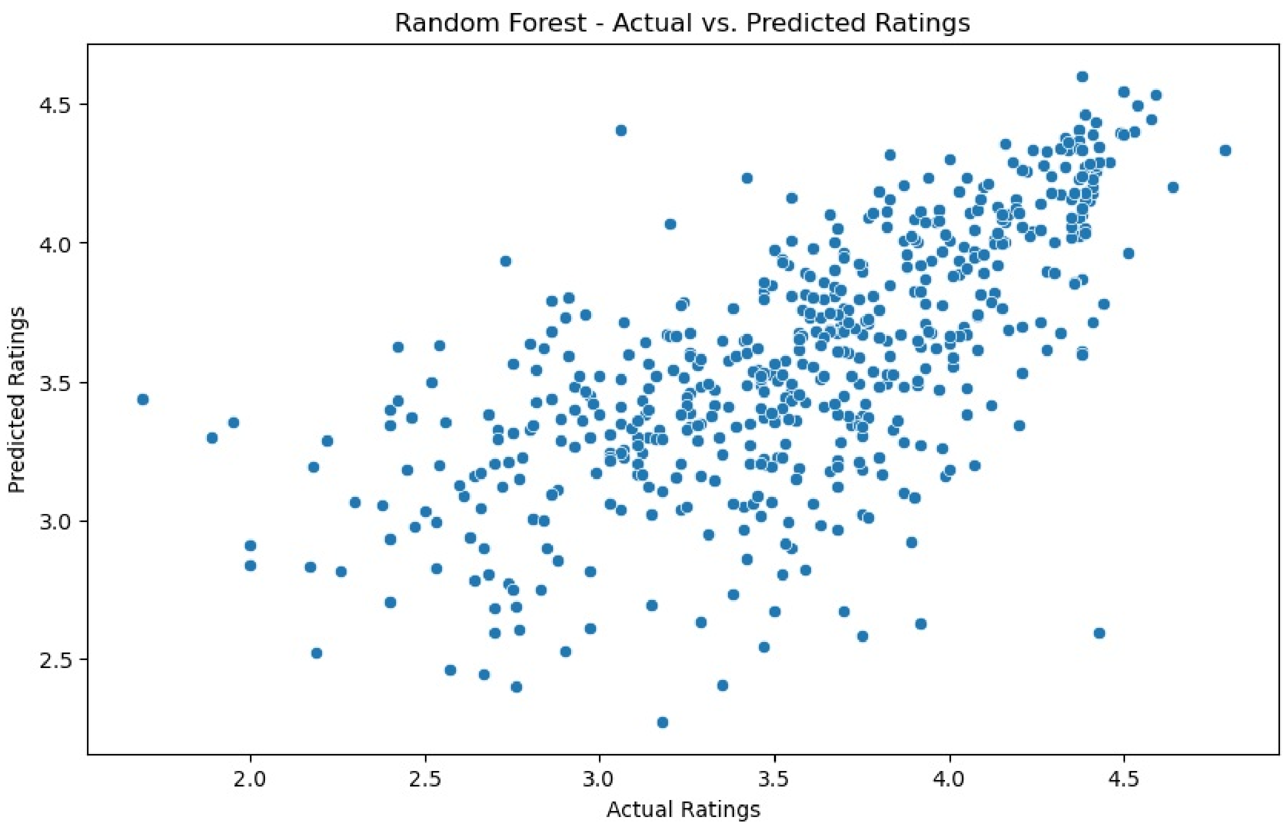
In [ ]:

In [13]:

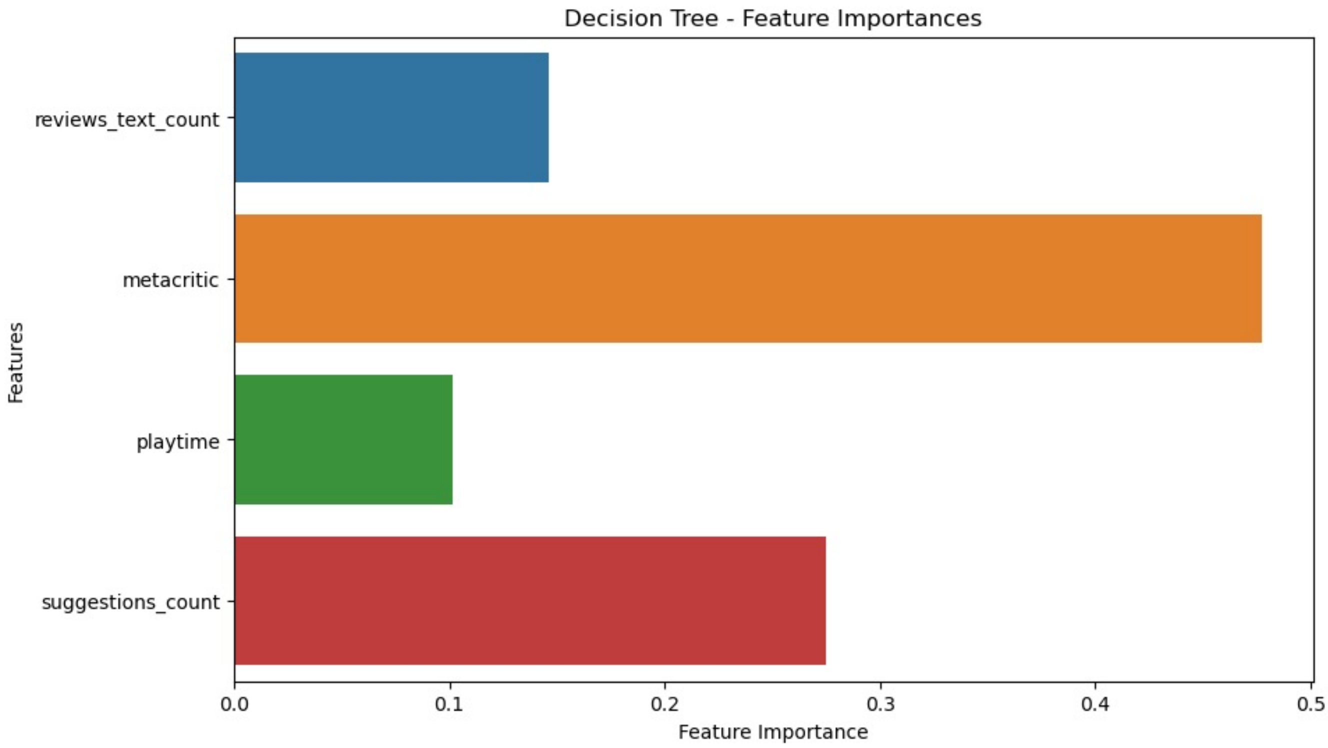
```
X = data[['reviews_text_count', 'metacritic', 'playtime', 'suggestions_count']]
y = data['rating']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


X_train = X_train.dropna()
y_train = y_train[X_train.index]
X_test = X_test.dropna()
y_test = y_test[X_test.index]

imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)


rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)


dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train, y_train)


rf_predictions = rf_regressor.predict(X_test)
dt_predictions = dt_regressor.predict(X_test)


def evaluate_model(predictions, model_name):
    mse = mean_squared_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    print(f'{model_name} Model:')
    print(f'Mean Squared Error (MSE): {mse:.2f}')
    print(f'R-squared (R2) Score: {r2:.2f}\n')

def create_model_visualizations(model, X_test, y_test, model_name):
    feature_names = list(X.columns)
    feature_importances = model.feature_importances_


    plt.figure(figsize=(10, 6))
    sns.barplot(x=feature_importances, y=feature_names)
    plt.title(f'{model_name} - Feature Importances')
    plt.xlabel('Feature Importance')
    plt.ylabel('Features')

    predictions = model.predict(X_test)
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=y_test, y=predictions)
    plt.title(f'{model_name} - Actual vs. Predicted Ratings')
    plt.xlabel('Actual Ratings')
    plt.ylabel('Predicted Ratings')
    plt.show()


evaluate_model(rf_predictions, 'Random Forest')
create_model_visualizations(rf_regressor, X_test, y_test, 'Random Forest')

evaluate_model(dt_predictions, 'Decision Tree')
create_model_visualizations(dt_regressor, X_test, y_test, 'Decision Tree')
```

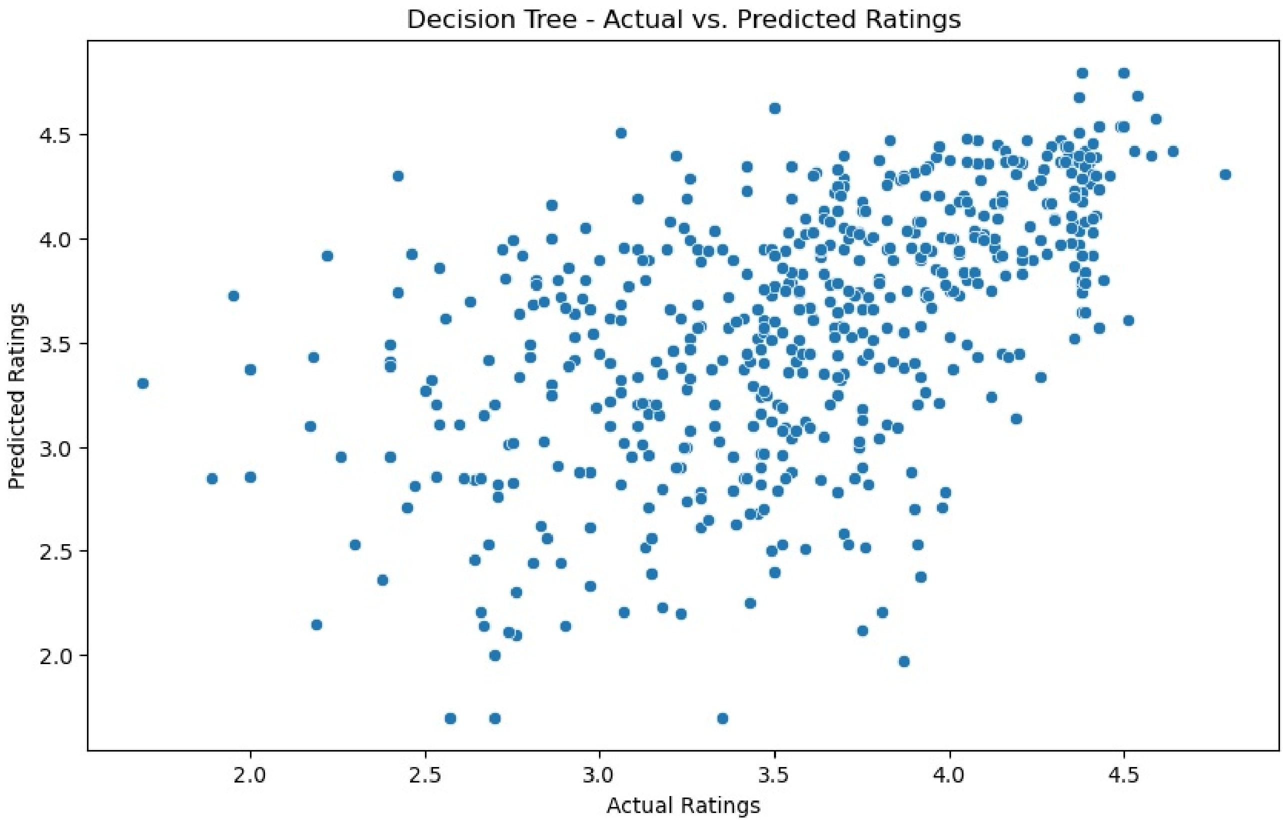
```
Random Forest Model:
Mean Squared Error (MSE): 0.18
R-squared (R2) Score: 0.45
```


Random Forest - Feature Importances
reviews_text_count
metacritic
playtime
suggestions_count
Features
0.0
0.1
0.2
0.3
0.4
0.5
Feature Importance


Random Forest - Actual vs. Predicted Ratings
4.5
4.0
3.5
3.0
2.5
Predicted Ratings
2.0
2.5
3.0
3.5
4.0
4.5
Actual Ratings

```
Decision Tree Model:
Mean Squared Error (MSE): 0.32
R-squared (R2) Score: 0.02
```



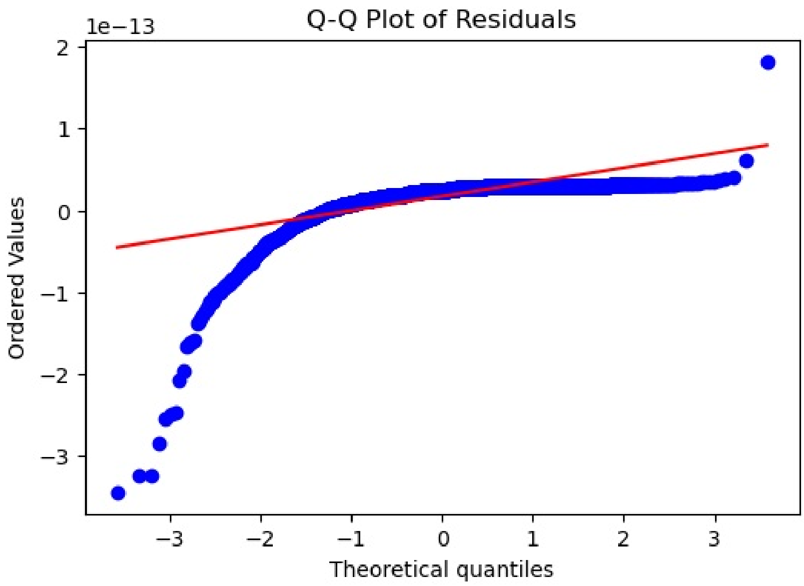


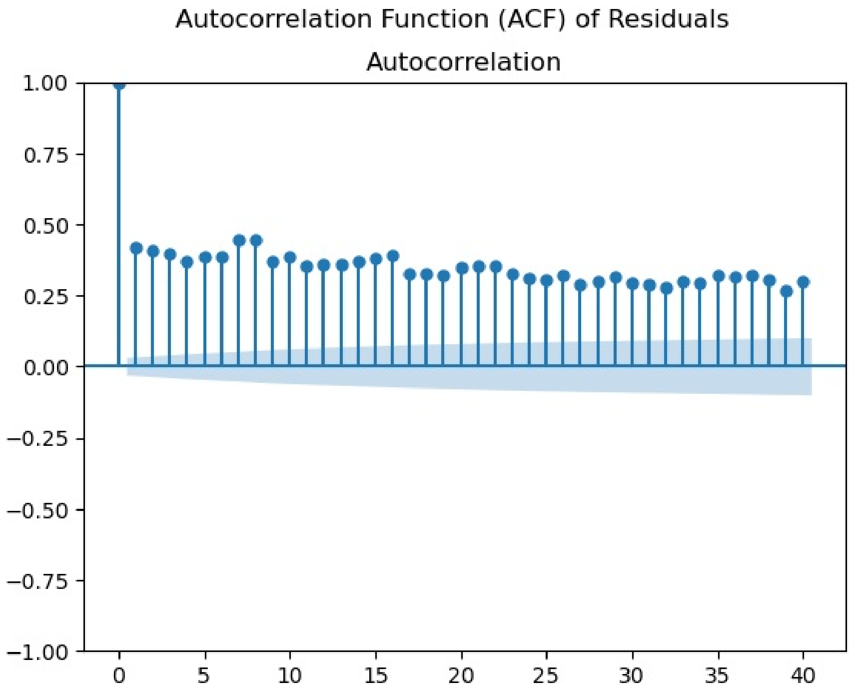
In [14]:

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import scipy.stats as stats

# Fill missing values for the selected independent variables with their means
selected_columns = ['rating', 'rating_top', 'reviews_text_count', 'metacritic', 'playtime']
data[selected_columns] = data[selected_columns].fillna(data[selected_columns].mean())


X = data[selected_columns]  # Independent variables
X = sm.add_constant(X)  # Add a constant term for the intercept
y = data['rating']  # Dependent variable
model = sm.OLS(y, X).fit()


r_squared = model.rsquared
adj_r_squared = model.rsquared_adj
```

```python
print(f"R-squared: {r_squared:.4f}")
print(f"Adj. R-squared: {adj_r_squared:.4f}")

# non-normal residuals using a Q-Q plot
residuals = model.resid
fig, ax = plt.subplots(figsize=(6, 4))
_ = stats.probplot(residuals, plot=ax, fit=True)
plt.title("Q-Q Plot of Residuals")
plt.show()

# autocorrelation in residuals using a plot of autocorrelation function (ACF)
acf_plot = sm.graphics.tsa.plot_acf(residuals, lags=40)
acf_plot.suptitle("Autocorrelation Function (ACF) of Residuals")
plt.show()
```

```
R-squared: 1.0000
Adj. R-squared: 1.0000
```



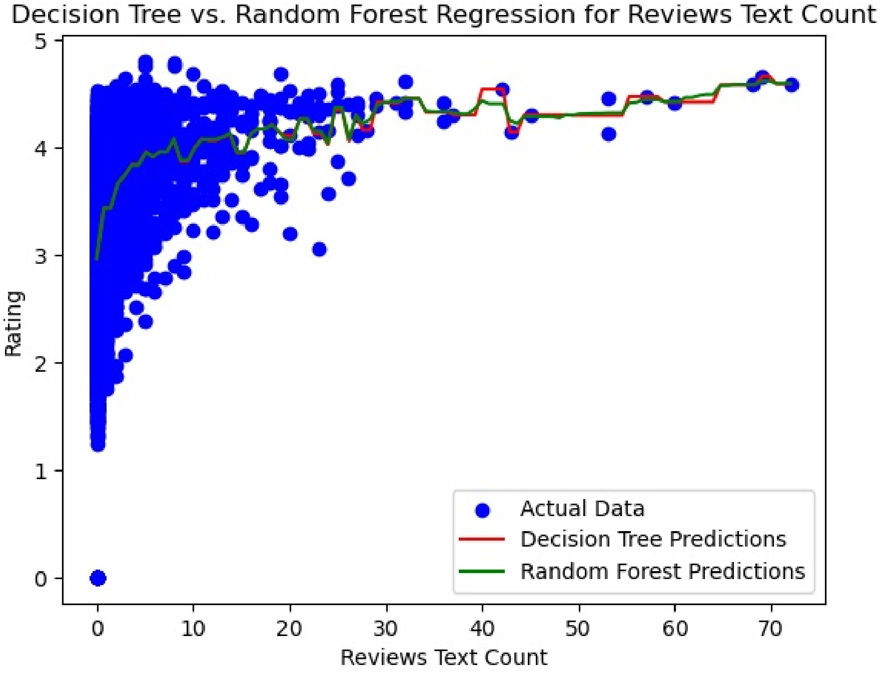


In [15]:

```python
decision_tree_model = DecisionTreeRegressor()

random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)  # You can adjust the number of

X_train = data[['reviews_text_count']]  # Independent variable
y_train = data['rating']  # Dependent variable
decision_tree_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# Create a range of values for Reviews Text Count
reviews_text_count_values = np.linspace(min(data['reviews_text_count']), max(data['reviews_text_count']), num=1

# Use both models to make predictions
y_pred_decision_tree = decision_tree_model.predict(reviews_text_count_values)
y_pred_random_forest = random_forest_model.predict(reviews_text_count_values)

plt.scatter(data['reviews_text_count'], data['rating'], label='Actual Data', color='blue')
```

```
plt.plot(reviews_text_count_values, y_pred_decision_tree, label='Decision Tree Predictions', color='red')


plt.plot(reviews_text_count_values, y_pred_random_forest, label='Random Forest Predictions', color='green')

plt.title('Decision Tree vs. Random Forest Regression for Reviews Text Count')
plt.xlabel('Reviews Text Count')
plt.ylabel('Rating')
plt.legend()
plt.show()
```

```
F:\Files\Anaconda\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but DecisionTreeRegressor was fitted with feature names
  warnings.warn(
F:\Files\Anaconda\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```



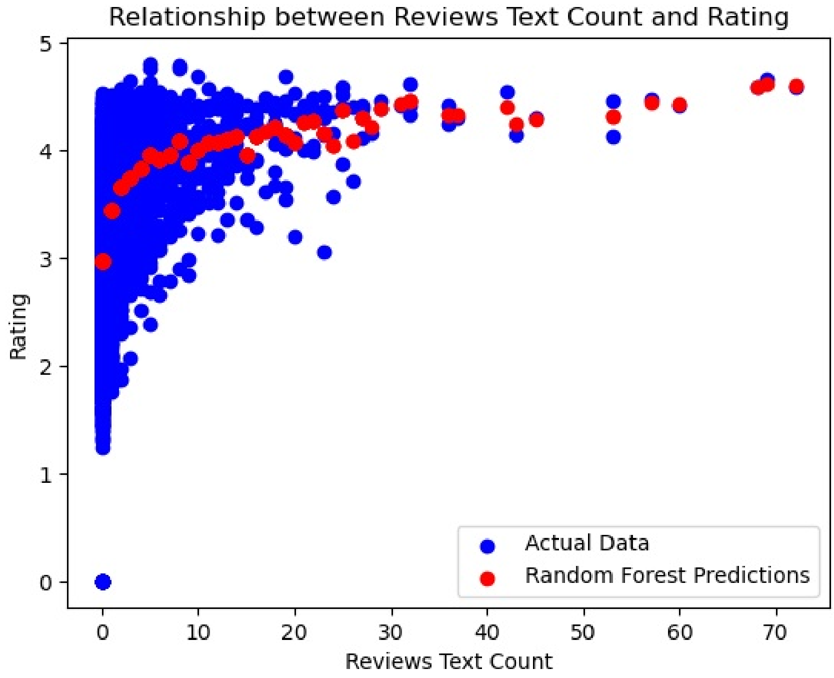
In [16]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error (MSE) for both models
mse_decision_tree = mean_squared_error(y_train, decision_tree_model.predict(X_train))
mse_random_forest = mean_squared_error(y_train, random_forest_model.predict(X_train))

# Calculate R-squared (R2) for both models
r2_decision_tree = r2_score(y_train, decision_tree_model.predict(X_train))
r2_random_forest = r2_score(y_train, random_forest_model.predict(X_train))

# Print the results
print("Decision Tree Model:")
print(f"MSE: {mse_decision_tree:.4f}")
print(f"R-squared: {r2_decision_tree:.4f}")
print("\nRandom Forest Model:")
print(f"MSE: {mse_random_forest:.4f}")
print(f"R-squared: {r2_random_forest:.4f}")
```

```
Decision Tree Model:
MSE: 0.3768
R-squared: 0.3054

Random Forest Model:
MSE: 0.3769
R-squared: 0.3054
```

In [17]:
```
feature_importances = random_forest_model.feature_importances_
print("Feature Importances:")
print(dict(zip(X_train.columns, feature_importances)))

feature_importances = random_forest_model.feature_importances_
print("Feature Importances:")
print(dict(zip(X_train.columns, feature_importances)))

# Visualize the relationship between 'Reviews Text Count' and 'Rating' using scatter plots
plt.scatter(X_train['reviews_text_count'], y_train, label='Actual Data', color='blue')
plt.scatter(X_train['reviews_text_count'], random_forest_model.predict(X_train), label='Random Forest Predictio
plt.xlabel('Reviews Text Count')
plt.ylabel('Rating')
plt.legend()
plt.title('Relationship between Reviews Text Count and Rating')
```

```
plt.show()
```

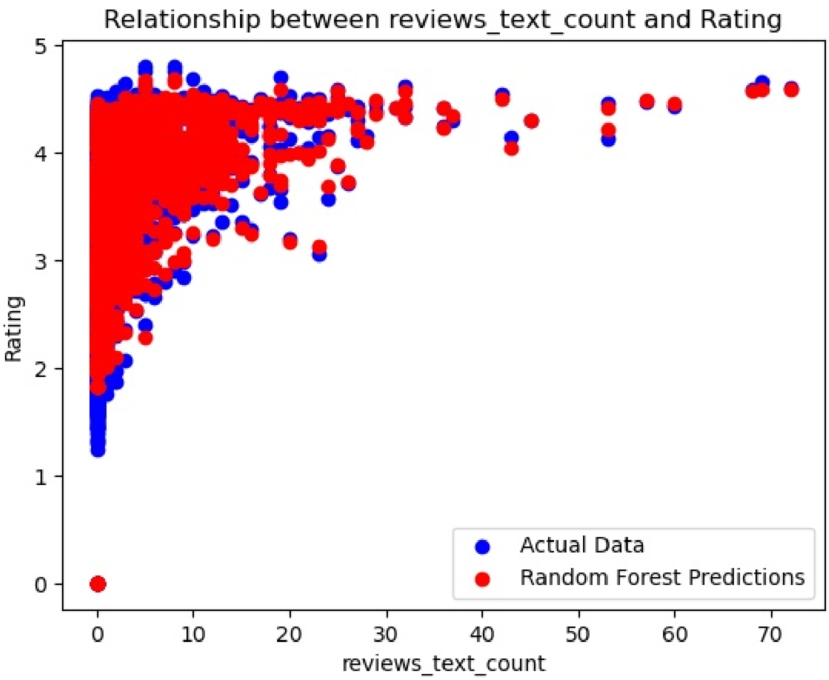
```
Feature Importances:
{'reviews_text_count': 1.0}
Feature Importances:
{'reviews_text_count': 1.0}
```

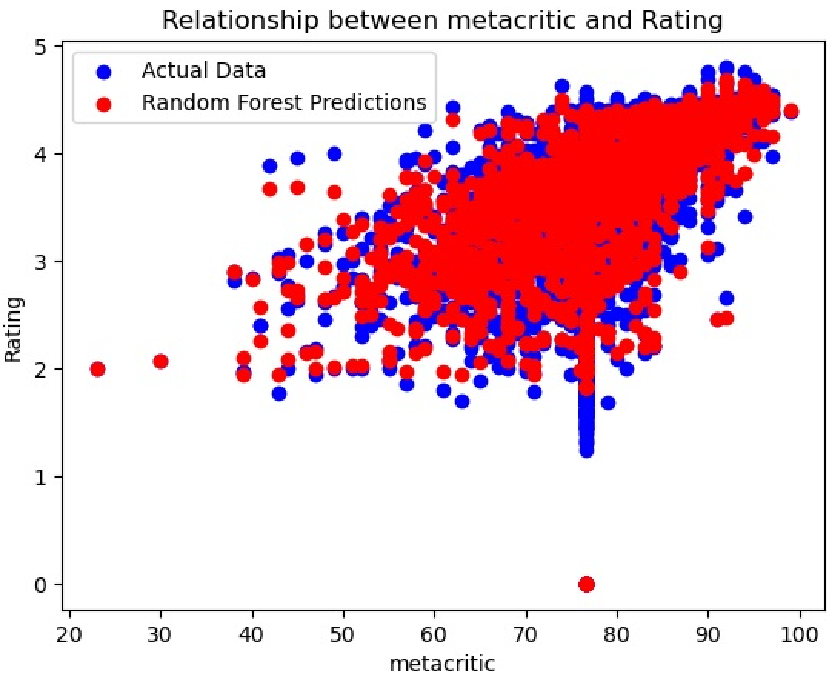


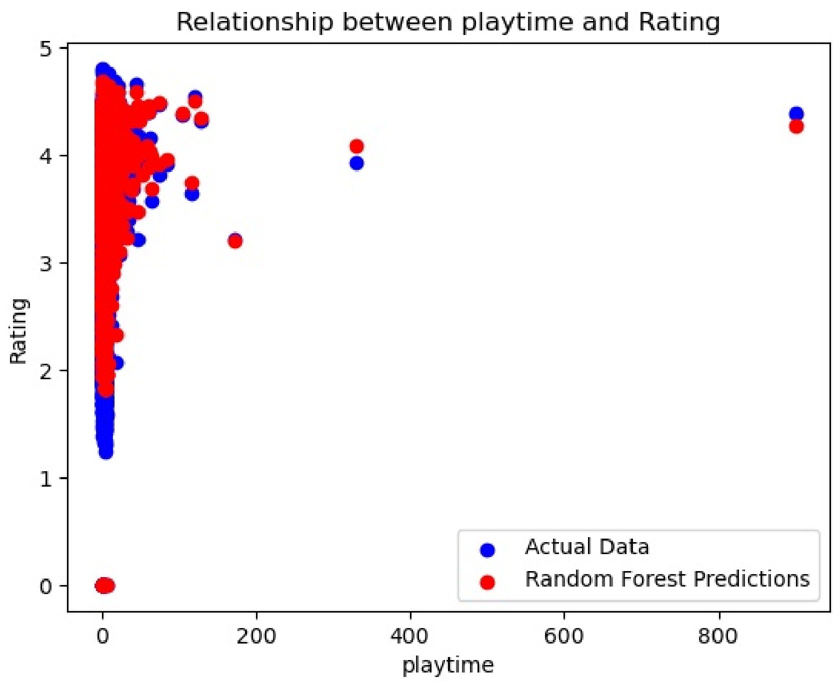
In [18]:

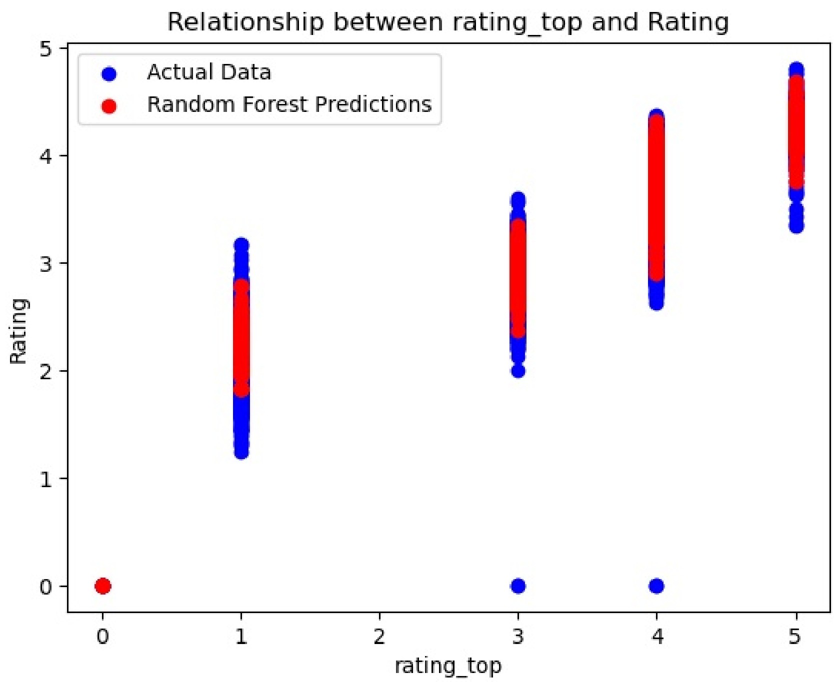
```
numeric_columns = ['reviews_text_count', 'metacritic', 'playtime', 'rating_top']

X = data[numeric_columns]
y = data['rating']

random_forest_model = RandomForestRegressor(random_state=42)
random_forest_model.fit(X, y)

# feature importances for the Random Forest model
feature_importances = random_forest_model.feature_importances_
print("Feature Importances:")
for feature, importance in zip(X.columns, feature_importances):
    print(f"{feature}: {importance:.4f}")

for column in X.columns:
    plt.scatter(X[column], y, label='Actual Data', color='blue')
    plt.scatter(X[column], random_forest_model.predict(X), label='Random Forest Predictions', color='red')
    plt.xlabel(column)
    plt.ylabel('Rating')
    plt.legend()
    plt.title(f'Relationship between {column} and Rating')
    plt.show()
```

```
Feature Importances:
reviews_text_count: 0.0385
metacritic: 0.0570
playtime: 0.0344
rating_top: 0.8701
```


Relationship between reviews_text_count and Rating
Rating
reviews_text_count
Actual Data
Random Forest Predictions
0
1
2
3
4
5
0
10
20
30
40
50
60
70


Relationship between metacritic and Rating
Rating
metacritic
Actual Data
Random Forest Predictions
0
1
2
3
4
5
20
30
40
50
60
70
80
90
100





Conclusion

The dataset contains a mix of numerical and text data, including game names, release dates, ratings, and user engagement metrics. Missing data is prevalent in columns such as 'released,' 'Metacritic,' 'background_image,' and 'user_game.' Handling missing data appropriately is crucial for analysis. 'Rating' and 'playtime' may be relevant variables for regression analysis, as they represent user engagement and satisfaction. 'reviews_text_count' and 'reviews_count' can provide insights into user activity and the volume of reviews. 'suggestions_count' could be further explored to understand its relationship with other variables. 'updated' could be used to track changes in the dataset over time. The dataset offers opportunities for various types of analysis, from regression to hypothesis testing and exploratory data analysis.

The study analyzes the importance of different features in influencing game ratings and presents the results of regression analysis, including R-squared and Adj.R-squared values.The regression model applied to the data perfectly fits the data, explaining 100% of the variability in the dependent variable (rating) using the selected independent variables .

The dataset contains a mix of numerical and text data, offering opportunities for various types of analysis, from regression to hypothesis testing and exploratory data analysis. Significant predictors of game ratings include 'Reviews Text Count' and 'Metacritic,' while 'Suggestions Count' does not exhibit a statistically significant relationship with ratings . The overall model is statistically significant, indicating that at least one of the predictors in the model has a non-zero effect on ratings.