In [1]:

```python
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from scipy.stats import skew

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, roc_curve
import scikitplot as skplt

import optuna
```

In [2]:

```python
#Read movie classification dataset
df = pd.read_csv("Movie_classification.csv")
```

In [3]:

```python
df.head(5)
```

Out[3]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
 2   Multiplex coverage   506 non-null    float64
 3   Budget               506 non-null    float64
 4   Movie_length         506 non-null    float64
 5   Lead_ Actor_Rating   506 non-null    float64
 6   Lead_Actress_rating  506 non-null    float64
 7   Director_rating      506 non-null    float64
 8   Producer_rating      506 non-null    float64
 9   Critic_rating        506 non-null    float64
 10  Trailer_views        506 non-null    int64  
 11  3D_available         506 non-null    object 
 12  Time_taken           494 non-null    float64
 13  Twitter_hastags      506 non-null    float64
 14  Genre                506 non-null    object 
 15  Avg_age_actors       506 non-null    int64  
 16  Num_multiplex        506 non-null    int64  
 17  Collection           506 non-null    int64  
 18  Start_Tech_Oscar     506 non-null    int64  
dtypes: float64(12), int64(5), object(2)
memory usage: 75.2+ KB
```

In [5]:

```
#display missing values per column
missing_values_count = df.isnull().sum()

print(missing_values_count)
```

```
Marketing expense        0
Production expense       0
Multiplex coverage       0
Budget                   0
Movie_length             0
Lead_ Actor_Rating       0
Lead_Actress_rating      0
Director_rating          0
Producer_rating          0
Critic_rating            0
Trailer_views            0
3D_available             0
Time_taken              12
Twitter_hastags          0
Genre                    0
Avg_age_actors           0
Num_multiplex            0
Collection               0
Start_Tech_Oscar         0
dtype: int64
```

In [6]:

```
#remove missing values column
df1 = df.copy()
df1 = df1.dropna()
df1.reset_index(inplace=True,drop=True)

missing_values_count1 = df1.isnull().sum()

print(missing_values_count1)
```

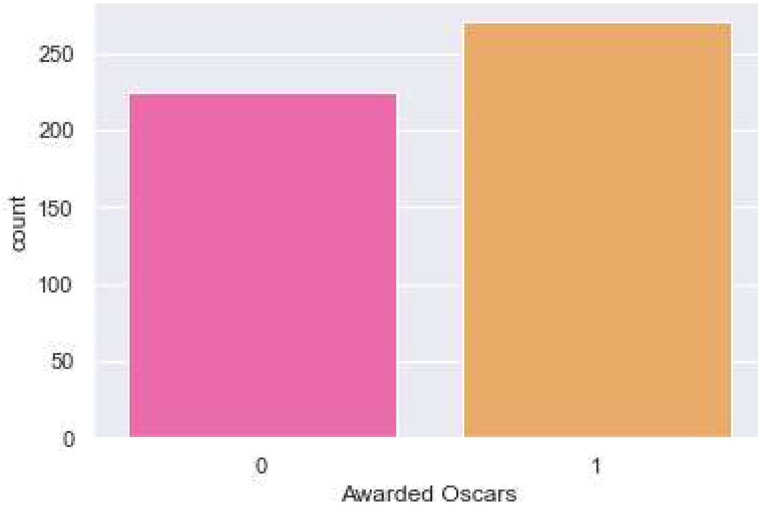
```
Marketing expense      0
Production expense     0
Multiplex coverage     0
Budget                 0
Movie_length           0
Lead_ Actor_Rating     0
Lead_Actress_rating    0
Director_rating        0
Producer_rating        0
Critic_rating          0
Trailer_views          0
3D_available           0
Time_taken             0
Twitter_hastags        0
Genre                  0
Avg_age_actors         0
Num_multiplex          0
Collection             0
Start_Tech_Oscar       0
dtype: int64
```

In [7]:

```
#set target value
target=df1["Start_Tech_Oscar"]
```

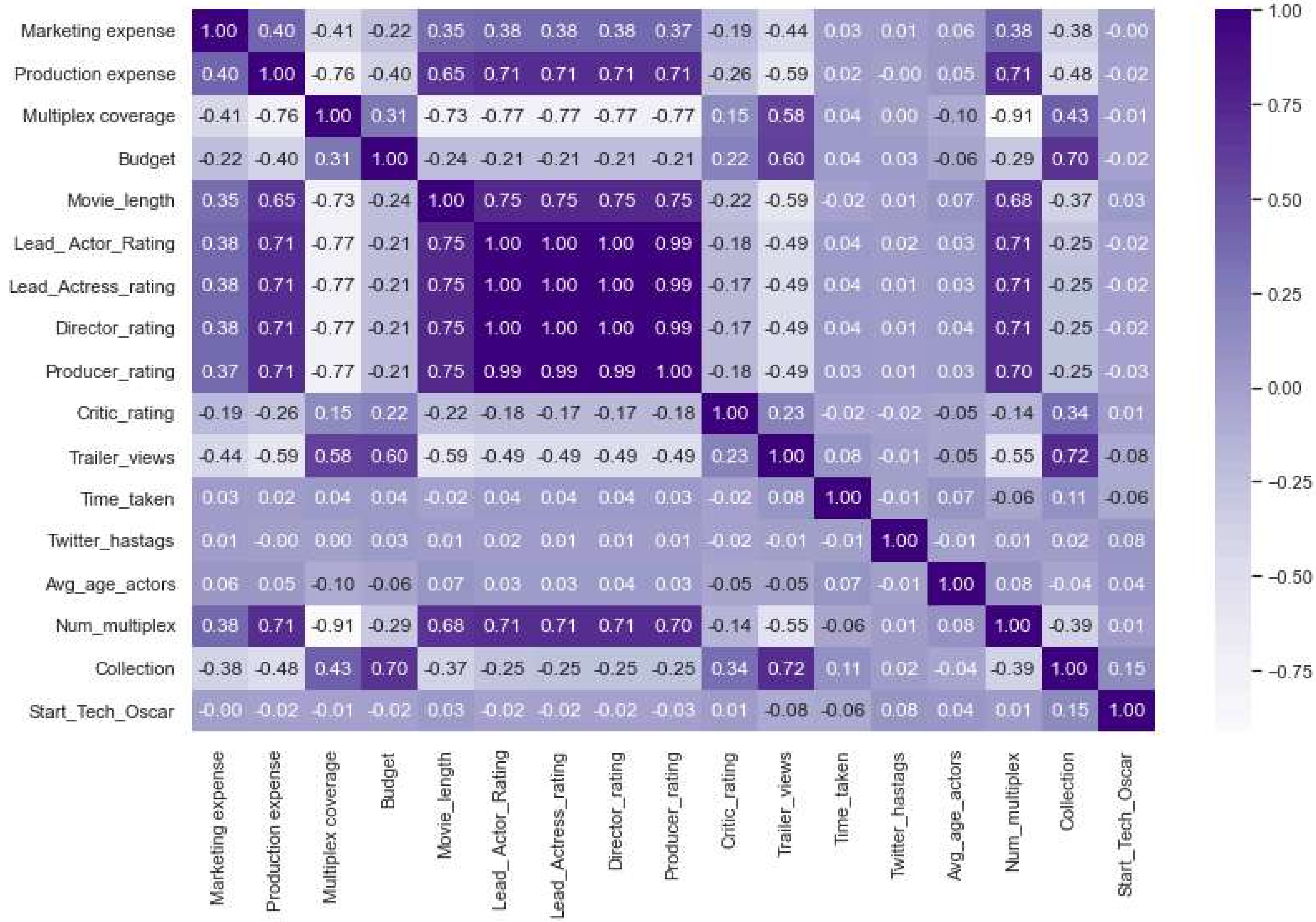
In [8]:

```
#count oscars that were awarded or not
sns.countplot(x = target, palette= "spring")
plt.xlabel("Awarded Oscars");
```



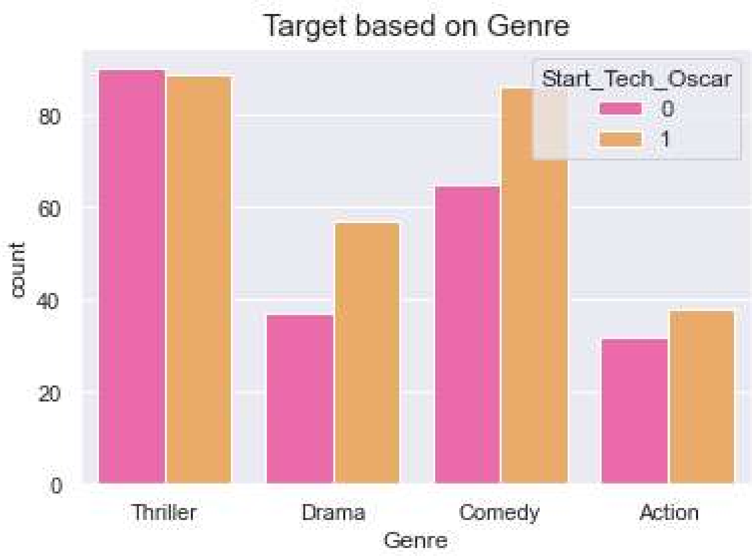
In [9]:

```python
#visualize correlation matrix
corr_matrix = df1.corr()
fig, ax = plt.subplots(figsize=(13, 8))
heatmap = sns.heatmap(corr_matrix, annot=True, cmap="Purples", fmt=".2f", ax=ax)
plt.show()
```



In [10]:

```
# Create a count plot of Genre with hue of Start_Tech_Oscar
sns.countplot(x="Genre", data=df1, hue="Start_Tech_Oscar", palette="spring")
plt.title('Target based on Genre', fontsize=15)
plt.show()
```

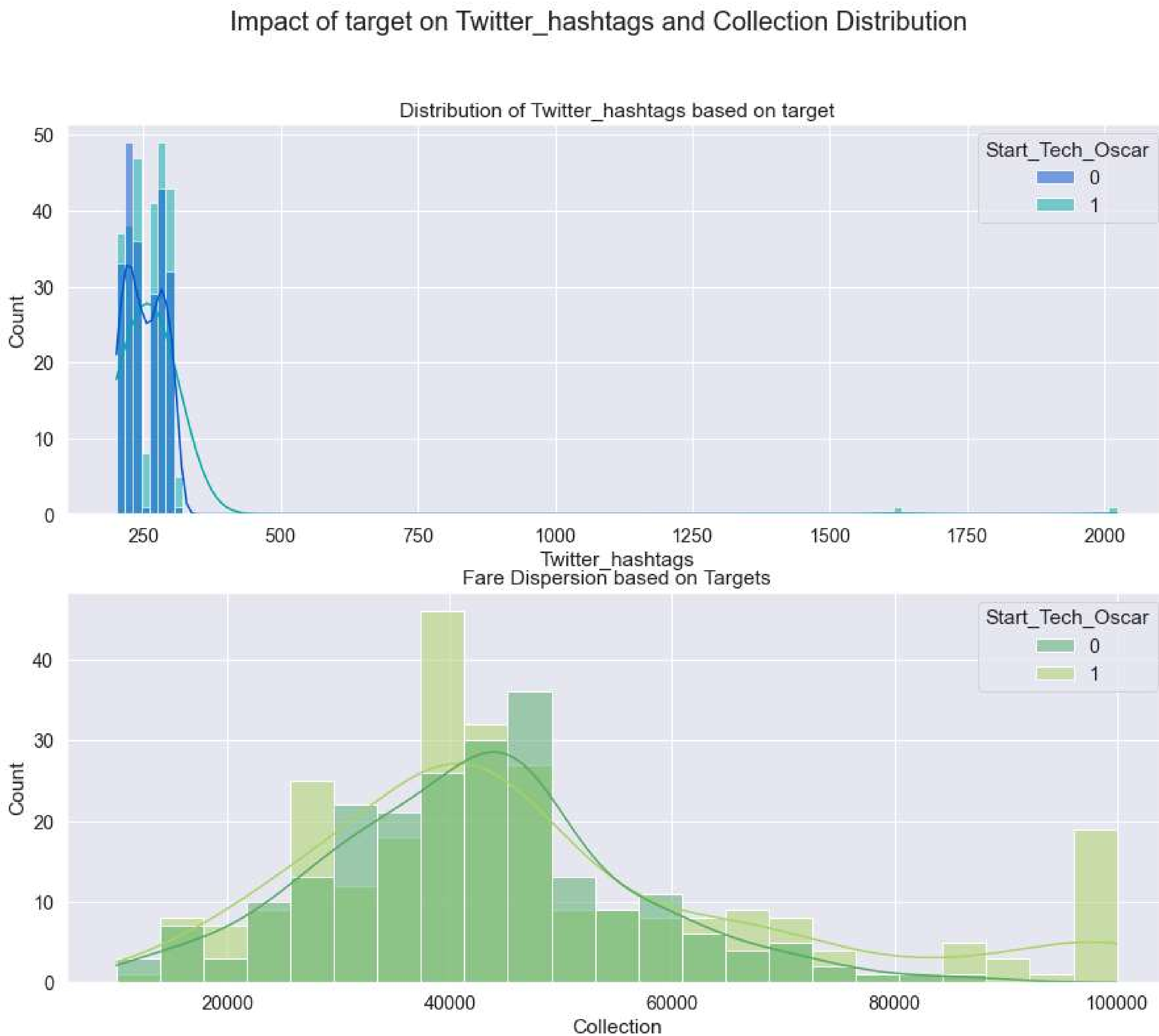


In [11]:

```
#Create a count plot of 3d available with hue of Start_Tech_Oscar
sns.countplot(x = "3D_available", data=df1, hue= "Start_Tech_Oscar", palette= "winter")
plt.title("Target based on Ticket Class", fontsize = 15);
plt.show()
```

In [12]:

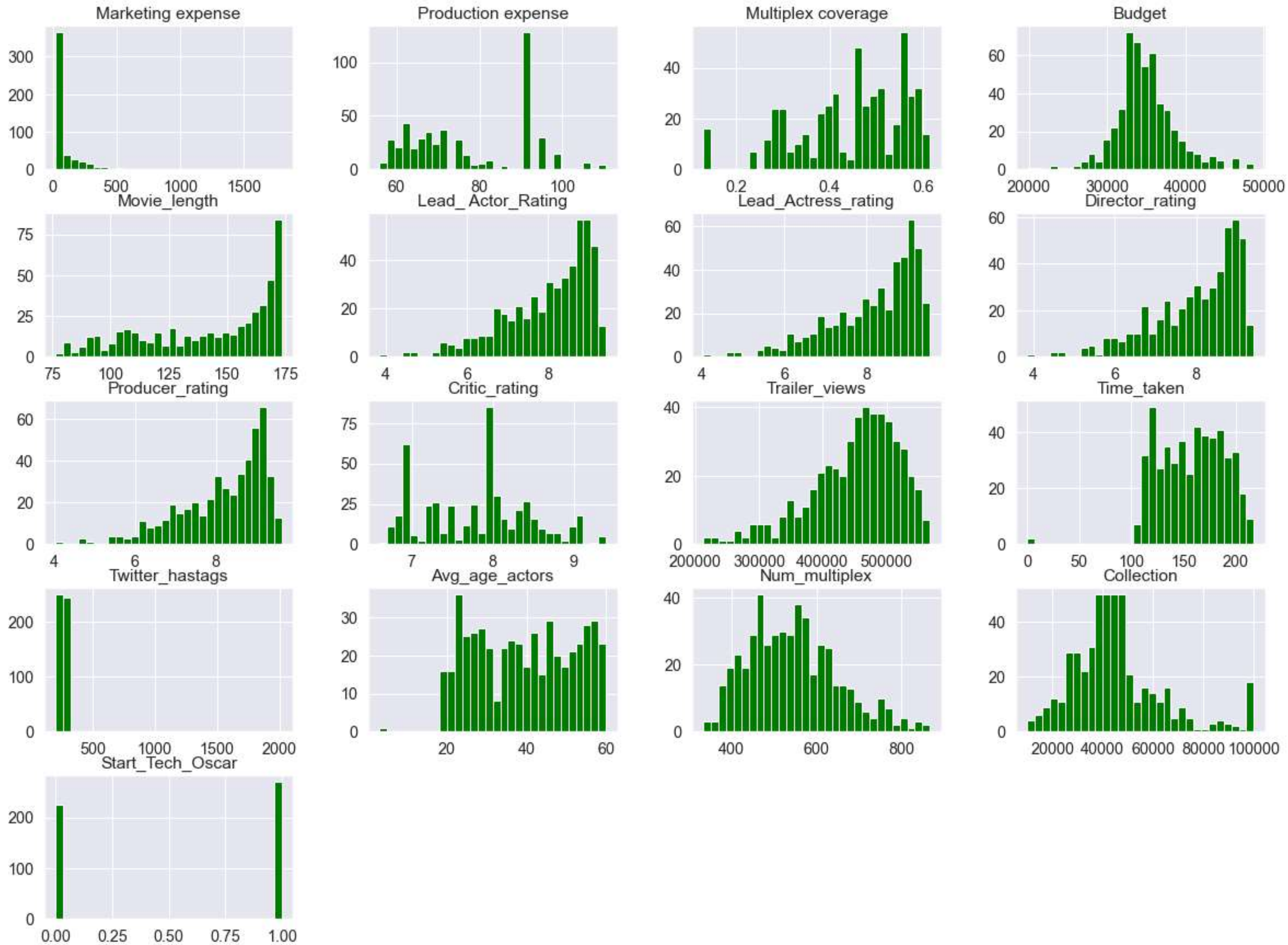
```
#Visualize impact of Twitter hashtags and Collection on target
sns.set_context("notebook", font_scale=1.3)

fig, ax = plt.subplots(nrows=2, figsize=(15, 12))

fig.suptitle("Impact of target on Twitter_hashtags and Collection Distribution", fontsize=20)

sns.histplot(x="Twitter_hastags", data=df1, hue="Start_Tech_Oscar", kde=True, ax=ax[0], palette="winter")
ax[0].set(xlabel='Twitter_hashtags', title="Distribution of Twitter_hashtags based on target")

sns.histplot(x="Collection", data=df1, hue="Start_Tech_Oscar", kde=True, ax=ax[1], palette="summer")
ax[1].set(xlabel="Collection", title="Fare Dispersion based on Targets")

plt.show()
```



Impact of target on Twitter_hashtags and Collection Distribution
Distribution of Twitter_hashtags based on target
Start_Tech_Oscar
0
1
Count
Twitter_hashtags
0
10
20
30
40
50
250
500
750
1000
1250
1500
1750
2000
Fare Dispersion based on Targets
Start_Tech_Oscar
0
1
Count
Collection
0
10
20
30
40
20000
40000
60000
80000
100000

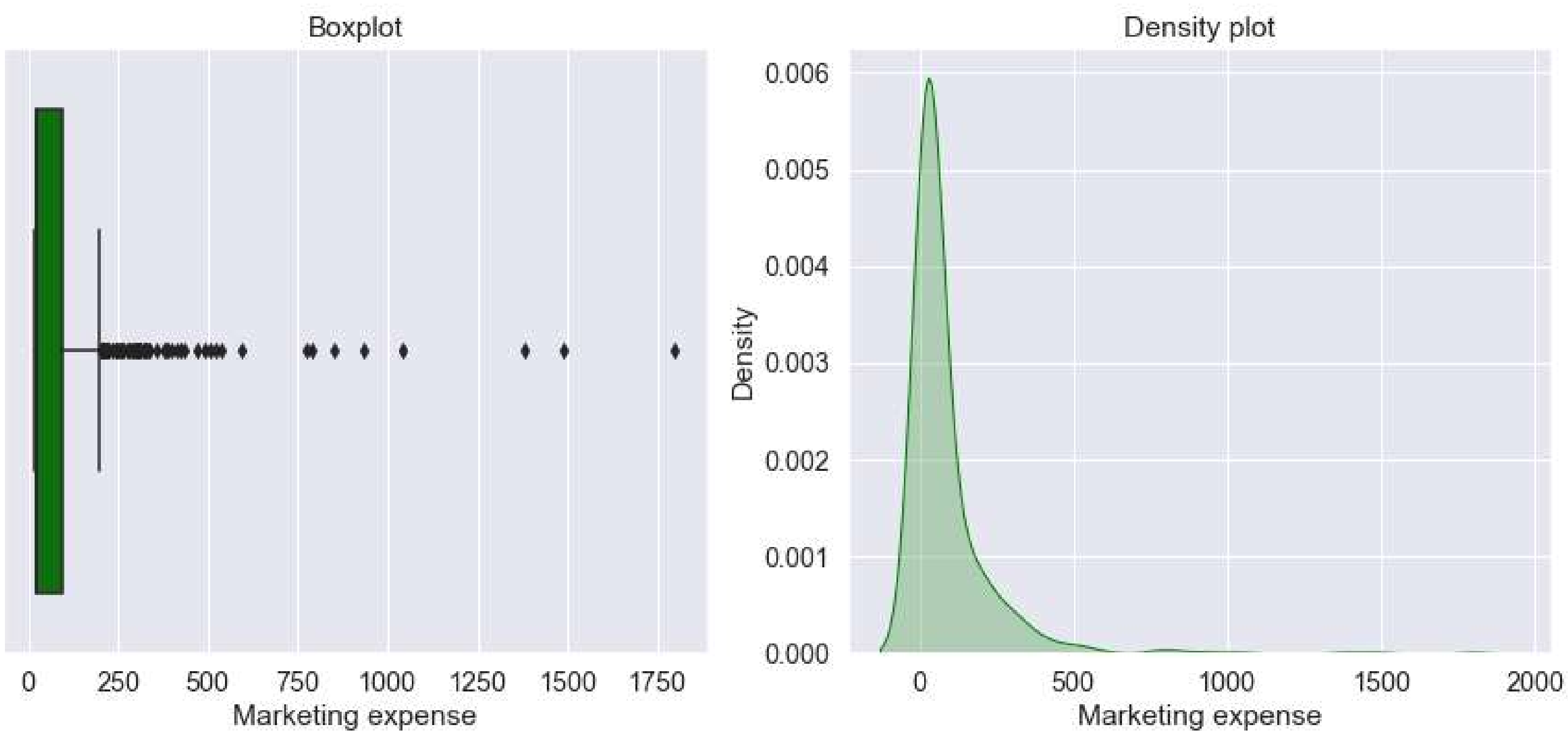
In [13]:

```python
#visualize histgrams for each attribute
df1.hist(bins = 30, figsize=(20, 15), color = "green");
```



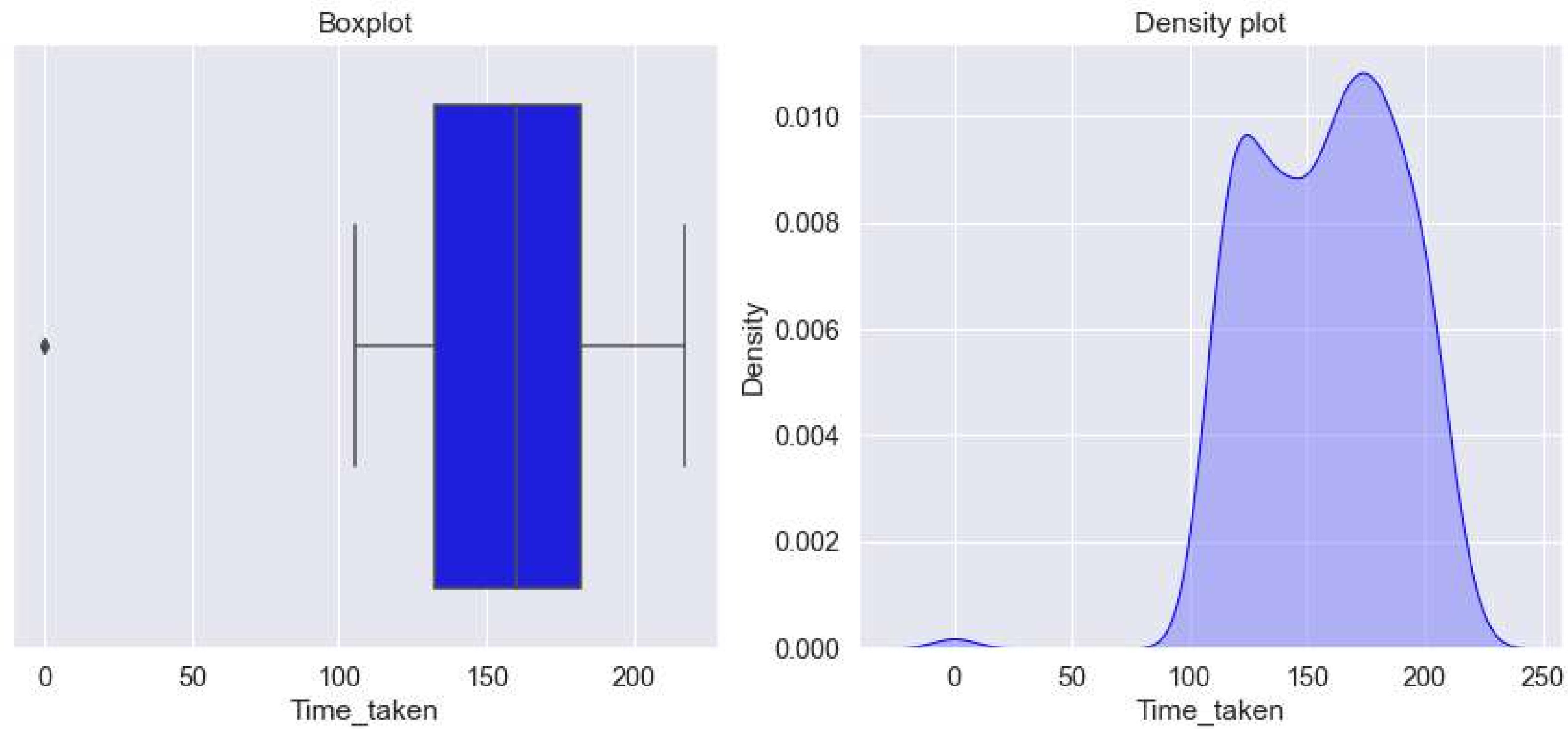
In [15]:

```
fig, axs = plt.subplots(ncols=2, figsize=(15, 6))

# plot the boxplot in the first subplot
sns.boxplot(x=df1["Marketing expense"], ax=axs[0], color='green')
axs[0].set_title("Boxplot")

# plot the density plot in the second subplot
sns.kdeplot(data=df1["Marketing expense"], shade=True, color="green", ax=axs[1])
axs[1].set_title("Density plot")

# display the figure
plt.show()
```



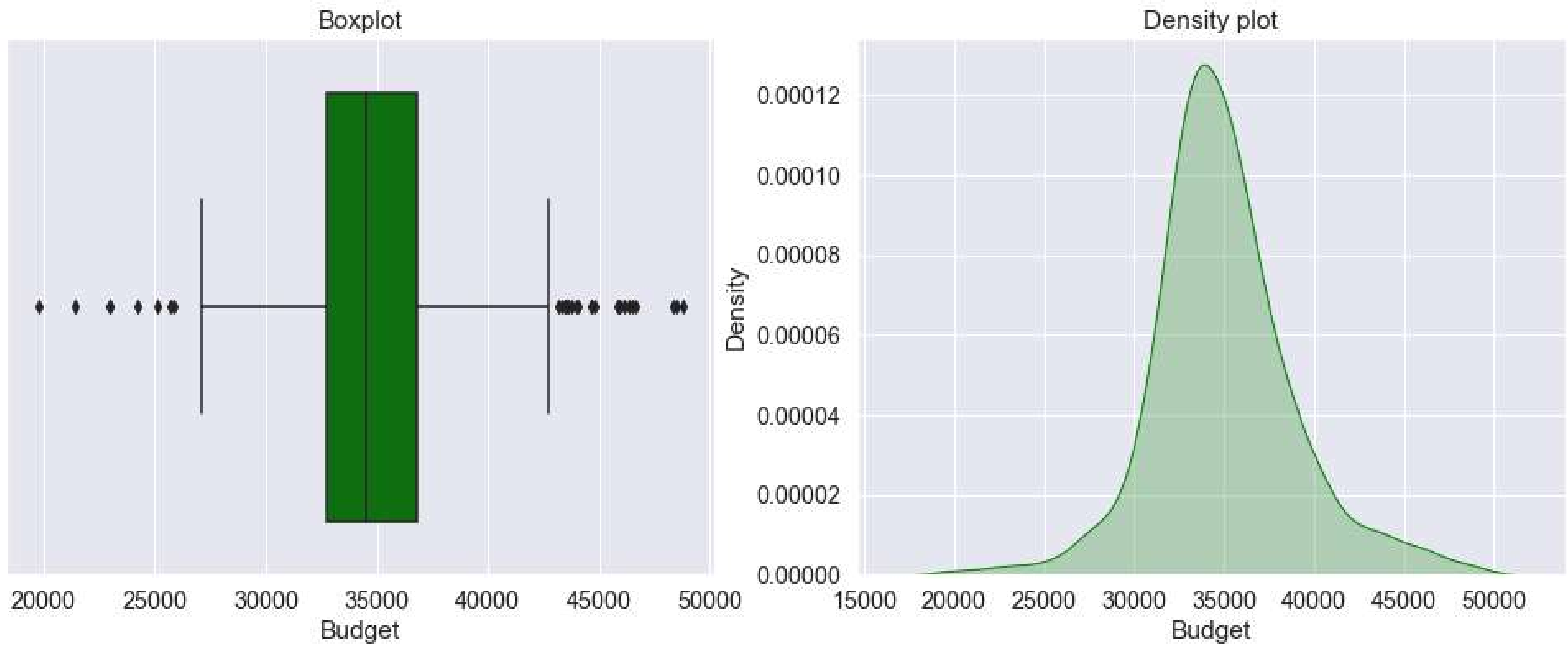
In [16]:

```python
fig, axs = plt.subplots(ncols=2, figsize=(15, 6))

# plot the boxplot in the first subplot
sns.boxplot(x=df1["Time_taken"], ax=axs[0], color='blue')
axs[0].set_title("Boxplot")

# plot the density plot in the second subplot
sns.kdeplot(data=df1["Time_taken"], shade=True, color="blue", ax=axs[1])
axs[1].set_title("Density plot")

# display the figure
plt.show()
```



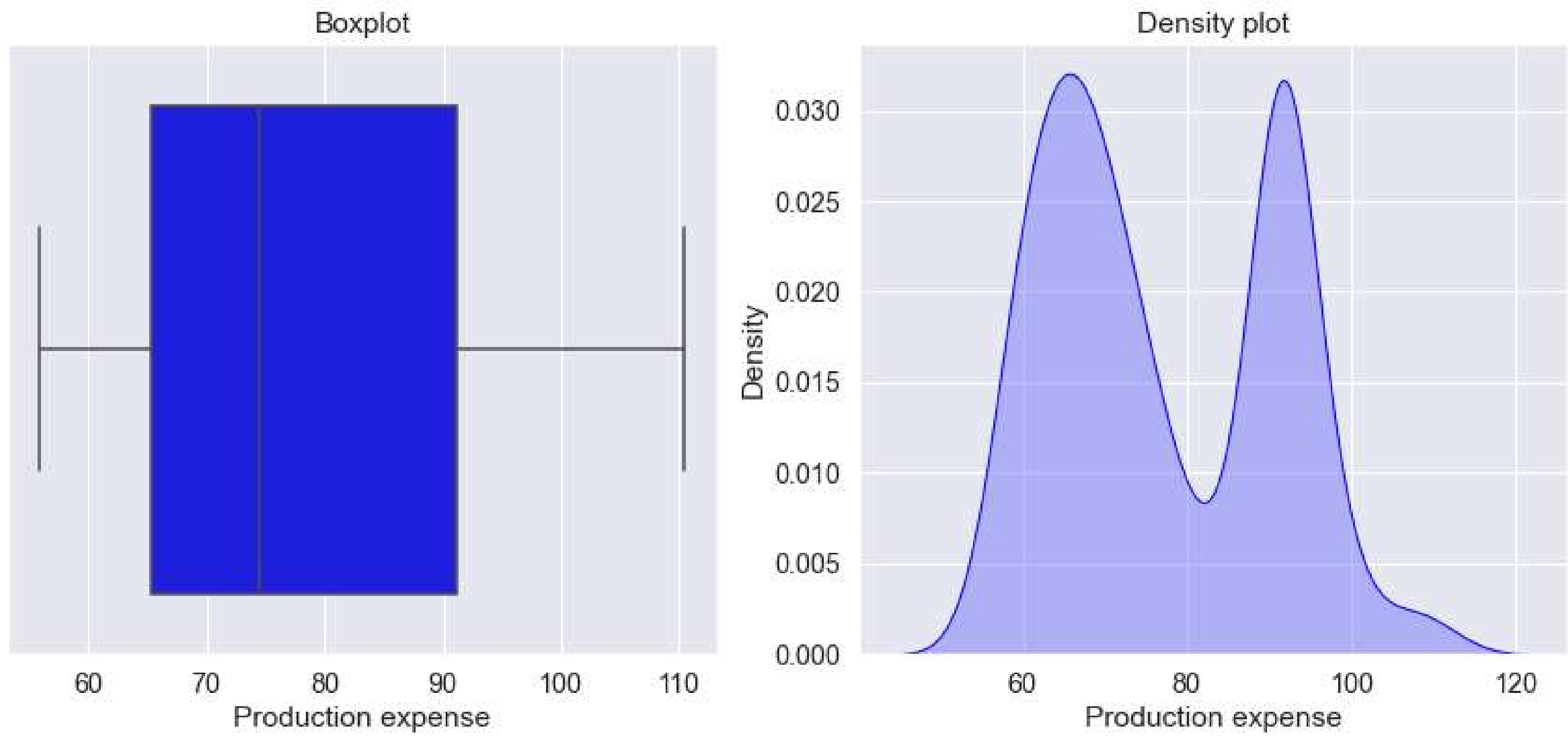
In [17]:

```
fig, axs = plt.subplots(ncols=2, figsize=(17, 6))

# plot the boxplot in the first subplot
sns.boxplot(x=df1["Budget"], ax=axs[0], color='green')
axs[0].set_title("Boxplot")

# plot the density plot in the second subplot
sns.kdeplot(data=df1["Budget"], shade=True, color="green", ax=axs[1])
axs[1].set_title("Density plot")

# display the figure
plt.show()
```



In [18]:

```
fig, axs = plt.subplots(ncols=2, figsize=(15, 6))

# plot the boxplot in the first subplot
sns.boxplot(x=df1["Production expense"], ax=axs[0], color='blue')
axs[0].set_title("Boxplot")

# plot the density plot in the second subplot
sns.kdeplot(data=df1["Production expense"], shade=True, color="blue", ax=axs[1])
axs[1].set_title("Density plot")

# display the figure
plt.show()
```



In [19]:

```
 #calculate expense and rating sums
df1['Expense'] = df1['Marketing expense'] + df1['Production expense']
df1['Rating'] = df1['Lead_ Actor_Rating'] + df1['Lead_Actress_rating'] + df1['Director_rating'] + df1['Pr
```

In [20]:

```
df2=df1.copy()
```

In [21]:

```
#drop columns
df2 = df2.drop(["Marketing expense","Production expense","Lead_ Actor_Rating","Lead_Actress_rating","Dire
```

In [22]:

```
 #drop rows based on conditions
df2.drop(df2[df2["Time_taken"] < 100].index, inplace = True)
df2.drop(df2[df2["Twitter_hastags"] > 500].index, inplace = True)
df2.reset_index(drop=True,inplace = True)
```

In [23]:

```
df2.head()
```

Out[23]:

	Multiplex coverage	Budget	Movie_length	Critic_rating	Trailer_views	3D_available	Time_taken	Twitter_hastags	Genre	Avg_age_ac
0	0.462	36524.125	138.7	7.94	527367	YES	109.60	223.840	Thriller	
1	0.531	35668.655	152.4	7.44	494055	NO	146.64	243.456	Drama	
2	0.542	38873.890	119.3	8.26	516279	YES	185.36	225.344	Drama	
3	0.542	39701.585	127.7	8.26	531448	NO	176.48	225.792	Drama	
4	0.542	35718.650	132.2	7.26	498425	YES	143.48	284.592	Comedy	

In [24]:
```python
#check skewness of the data
skew_df = pd.DataFrame(df2.select_dtypes(np.number).columns, columns=['Feature'])
skew_df['Skew'] = skew_df['Feature'].apply(lambda feature: skew(df2[feature]))
skew_df['Absolute Skew'] = skew_df['Skew'].apply(abs)
skew_df['Skewed'] = skew_df['Absolute Skew'].apply(lambda x: True if x >= 0.5 else False)
skew_df
```

Out[24]:

	Feature	Skew	Absolute Skew	Skewed
0	Multiplex coverage	-0.736768	0.736768	True
1	Budget	0.386061	0.386061	False
2	Movie_length	-0.581770	0.581770	True
3	Critic_rating	0.137840	0.137840	False
4	Trailer_views	-0.879177	0.879177	True
5	Time_taken	-0.012897	0.012897	False
6	Twitter_hastags	-0.020841	0.020841	False
7	Avg_age_actors	-0.009889	0.009889	False
8	Num_multiplex	0.541919	0.541919	True
9	Collection	1.115652	1.115652	True
10	Start_Tech_Oscar	-0.180320	0.180320	False
11	Expense	4.963830	4.963830	True
12	Rating	-1.006243	1.006243	True

In [25]:
```python
#apply log transformation to reduce skewness
for column in skew_df.query("Skewed == True")['Feature'].values:
    df2[column] = np.log1p(df2[column])
```

In [26]:

```
df2.head()
```

Out[26]:

	Multiplex coverage	Budget	Movie_length	Critic_rating	Trailer_views	3D_available	Time_taken	Twitter_hastags	Genre	Avg_age_ac
0	0.379805	36524.125	4.939497	7.94	13.175654	YES	109.60	223.840	Thriller	
1	0.425921	35668.655	5.033049	7.44	13.110404	NO	146.64	243.456	Drama	
2	0.433080	38873.890	4.789989	8.26	13.154405	YES	185.36	225.344	Drama	
3	0.433080	39701.585	4.857484	8.26	13.183363	NO	176.48	225.792	Drama	
4	0.433080	35718.650	4.891852	7.26	13.119210	YES	143.48	284.592	Comedy	

In [27]:

```
#one-hot encode categorical variables
df3=df2.copy()
df3 = pd.get_dummies(df3)
df3.head()
```

Out[27]:

	Multiplex coverage	Budget	Movie_length	Critic_rating	Trailer_views	Time_taken	Twitter_hastags	Avg_age_actors	Num_multiplex	C
0	0.379805	36524.125	4.939497	7.94	13.175654	109.60	223.840	23	6.204558	1
1	0.425921	35668.655	5.033049	7.44	13.110404	146.64	243.456	42	6.137727	1
2	0.433080	38873.890	4.789989	8.26	13.154405	185.36	225.344	45	6.159095	1
3	0.433080	39701.585	4.857484	8.26	13.183363	176.48	225.792	55	5.981414	1
4	0.433080	35718.650	4.891852	7.26	13.119210	143.48	284.592	53	6.133398	1

In [28]:

```
#scale the data
sc = StandardScaler()
df3[df2.select_dtypes(np.number).columns] = sc.fit_transform(df3[df2.select_dtypes(np.number).columns])
```

In [29]:
```python
#drop the target column from the dataset
df3.drop(["Start_Tech_Oscar"],axis =1, inplace = True)
df3.head()
```

Out[29]:

	Multiplex coverage	Budget	Movie_length	Critic_rating	Trailer_views	Time_taken	Twitter_hastags	Avg_age_actors	Num_multiplex	Co
0	0.173643	0.402450	0.000267	0.180195	0.998781	-1.629537	-0.920102	-1.307596	-0.410469	0
1	0.726879	0.184281	0.431080	-0.580711	0.616224	-0.383424	-0.334914	0.209189	-0.759252	0
2	0.812766	1.001707	-0.688232	0.667174	0.874197	0.919208	-0.875234	0.448682	-0.647733	1
3	0.812766	1.212793	-0.377410	0.667174	1.043976	0.620464	-0.861870	1.246990	-1.575033	1
4	0.812766	0.197032	-0.219144	-0.854637	0.667855	-0.489734	0.892264	1.087328	-0.781845	0

In [30]:
```python
#reset the target
target = df2["Start_Tech_Oscar"].astype(int)
target
```

Out[30]:
```
0      1
1      0
2      1
3      1
4      0
      ..
485    0
486    0
487    0
488    0
489    0
Name: Start_Tech_Oscar, Length: 490, dtype: int32
```

In [31]:
```python
#splitting and training the data
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size= 0.2, stratify= target, random
```

In [32]:

```
#choose models
models = {
    "logistic regression": LogisticRegression(),
    "xgboost" : XGBClassifier(),
    "naive bayes": GaussianNB(),
    "random forest" : RandomForestClassifier(),
}
```

In [33]:

```
#train models using defined algorithms
for name, model in models.items():
    model.fit(df3, target)
    print(f'{name} trained')
```

```
logistic regression trained
xgboost trained
naive bayes trained
random forest trained
```

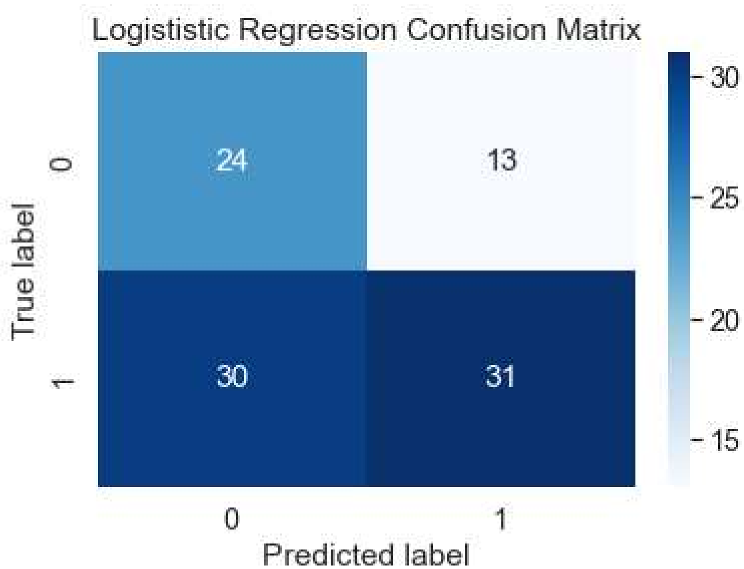
In [34]:

```
#cross-validate models and calculate AUC scores
from sklearn.model_selection import cross_val_score, KFold

results = {}

kf = KFold(n_splits=10)

for name, model in models.items():
    cv_results = cross_val_score(model, df3, target, scoring='roc_auc', cv=kf)
    results[name] = cv_results.mean()

# Print the mean cross-validation scores for each model
for name, mean_score in results.items():
    print(f'{name}: Mean AUC = {mean_score:.4f}')
```

```
logistic regression: Mean AUC = 0.6603
xgboost: Mean AUC = 0.6546
naive bayes: Mean AUC = 0.6052
random forest: Mean AUC = 0.6580
```

In [35]:

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

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size=0.2, random_state=42)

# Train the model (Logistic Regression)
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, cmap='Blues')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Logististic Regression Confusion Matrix')
plt.show()
```



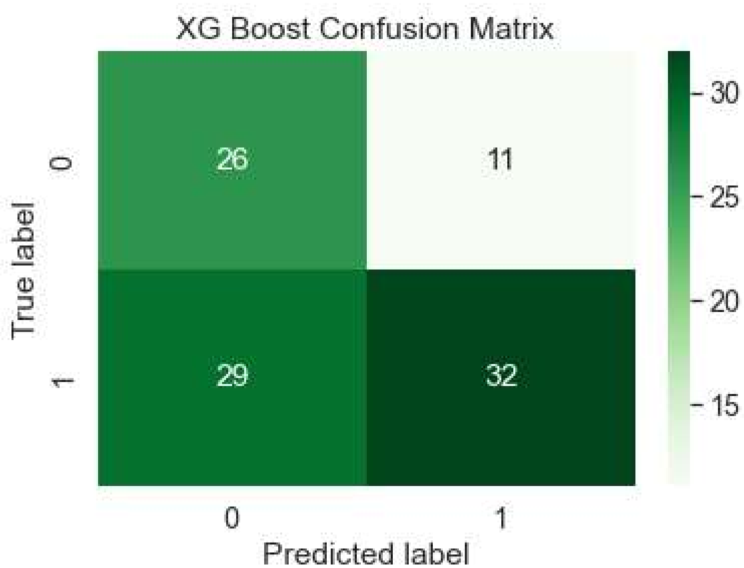
In [36]:

```python
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size=0.2, random_state=42)

# Train the model (Logistic Regression)
model = LogisticRegression()
model.fit(X_train, y_train)

# Make probability estimates for the positive class
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC AUC score
roc_auc = roc_auc_score(y_test, y_pred_proba)

print('ROC AUC score:', roc_auc)
```

```
ROC AUC score: 0.6619406291537439
```

In [37]:

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size=0.2, random_state=42)

# Train the model (XGBClassifier)
model = XGBClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, cmap='Greens')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('XG Boost Confusion Matrix')
plt.show()
```



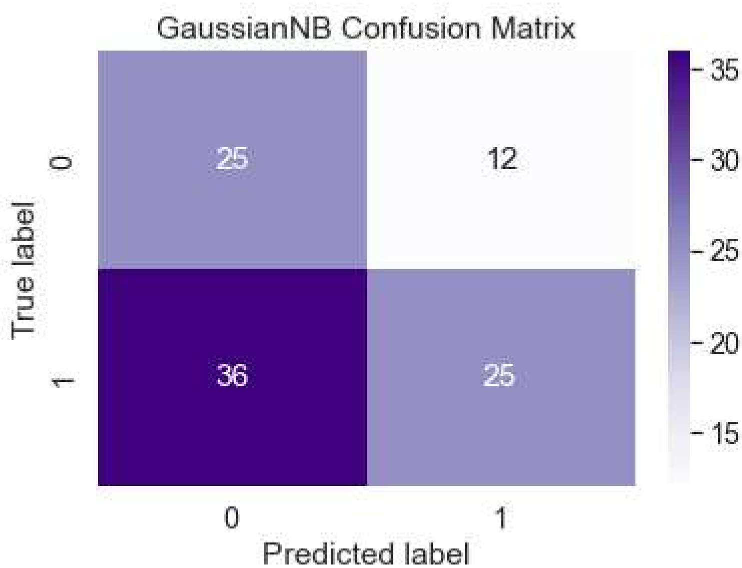
In [38]:

```python
from sklearn.metrics import roc_auc_score
from xgboost import XGBClassifier

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size=0.2, random_state=42)

# Train the model (XGBClassifier)
model = XGBClassifier()
model.fit(X_train, y_train)

# Make probability estimates for the positive class
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC AUC score
roc_auc = roc_auc_score(y_test, y_pred_proba)

print('ROC AUC score:', roc_auc)
```

```
ROC AUC score: 0.6375719982277359
```

In [39]:

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size=0.2, random_state=42)

# Train the model (GaussianNB)
model = GaussianNB()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, cmap='Purples')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('GaussianNB Confusion Matrix')
plt.show()
```



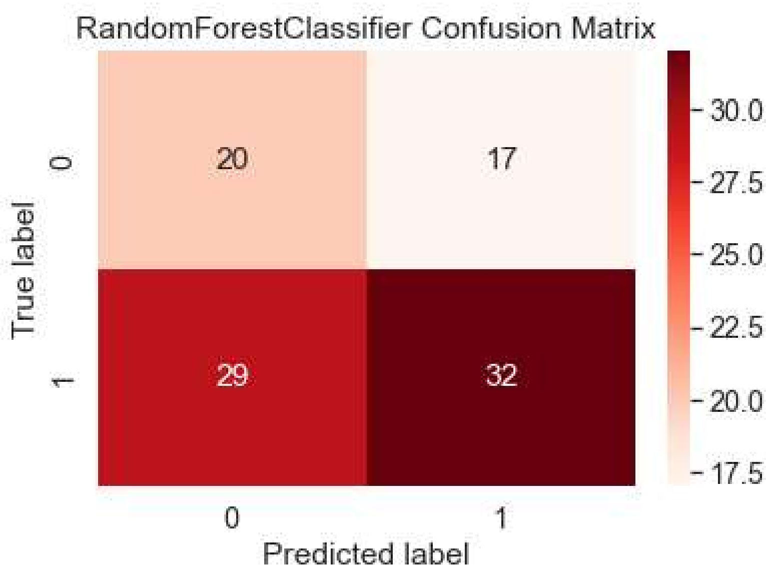
In [40]:

```
from sklearn.metrics import roc_auc_score

# Calculate ROC AUC score
y_pred_proba = model.predict_proba(X_test)[:,1]
roc_score = roc_auc_score(y_test, y_pred_proba)

print('ROC AUC Score:', roc_score)
```

```
ROC AUC Score: 0.5795303500221534
```

In [41]:

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df3, target, test_size=0.2, random_state=42)

# Train the model (RandomForestClassifier)
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, cmap='Reds')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('RandomForestClassifier Confusion Matrix')
plt.show()
```



In [44]:

```
from sklearn.metrics import roc_auc_score

# Calculate predicted probabilities for test set
y_pred_proba = model.predict_proba(X_test)

# Reshape predicted probabilities to be two-dimensional
y_pred_proba = y_pred_proba.reshape(-1, 2)

# Calculate ROC score for RandomForestClassifier
roc_score = roc_auc_score(y_test, y_pred_proba[:, 1])

print("ROC score:", roc_score)
```

```
ROC score: 0.601905183872397
```

In [ ]:

```

```