Figure out gaming genre trends from indi games 2000-2025

Compare pricing from those years and how its growing

Make a predictive modle for future endevors

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste
df
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
RuntimeError: module compiled against API version 0xe but this version of numpy is 0xd
```

Out[1]:

	gameid	title	developers	publishers	genres	year
0	1417940	捕鱼炸翻天	['上海愉游网络科技有限公司']	['上海姚际信息科技有限公司']	['Adventure', 'Casual', 'Free To Play', 'Indie...	2020-10-26
1	1358770	The falling tower	['Carotaa']	['Dream Night Studio']	['Indie', 'RPG', 'Strategy']	2020-07-15
2	652780	The Mutational	['DinjaStudios']	['DinjaStudios']	['Action', 'Adventure', 'Casual', 'Free To Pla...	2020-02-10
3	437160	The Lion's Song: Episode 1 - Silence	["Mi'pu'mi Games GmbH"]	["Mi'pu'mi Games GmbH"]	['Adventure', 'Indie']	2016-07-07
4	1619570	Death Roads: Tournament	['The Knights of Unity']	['The Knights of Unity', 'Surefire.Games']	['Indie', 'Racing', 'Strategy']	2023-11-15
...	...	...	...	...	...	...
9995	1103890	Opening Up	['SM']	['SM']	['Adventure']	2020-06-04
9996	2192240	The Ninja	['Anthony Antunovic']	['Anthony Antunovic']	['Action', 'Adventure', 'Casual', 'Indie']	2022-11-11
9997	2633180	Bloody Good Friends	['Five Owls Games']	['Five Owls Games']	['Casual', 'Indie']	2024-05-16
9998	334040	Down To One	['Gadget Games']	['My Way Games']	['Action']	2016-01-07
9999	1763110	PROJECT: Halloween	['Studio Madeleine Chai']	['Studio Madeleine Chai']	['Casual', 'Indie']	2021-10-21

10000 rows × 6 columns

In [2]:
```
df2 = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 St
df2
```

Out[2]:

	gameid	Name	Estimated owners	Required age	About the game	Reviews	Windows	Metacritic score	User score	Pos
0	Surface: Project Dawn	Sep 15, 2023	1	9.99	The next spine-tingling installment of the Sur...	NaN	True	0	0	
1	Dis Pontibus	Jan 7, 2019	0	4.99	Dis Pontibus is a single-player puzzle game se...	NaN	True	0	0	
2	Furry Sweeper	Feb 20, 2023	1	2.69	Furry Sweeper is a variant of minesweeper wher...	NaN	True	0	0	
3	Street Hoop	Dec 31, 2019	0	4.99	Street Hoop is DATA EAST's 3 on 3 street baske...	NaN	True	0	0	
4	Flag Sweeper	Mar 24, 2023	0	0.00	This is a two player version of minesweeper wh...	NaN	True	0	0	
...	...	...	...	...	...	...	...	...	...	
9995	The End o,,,o	May 11, 2017	0	0.00	A short game about the day things got weird. Y...	NaN	True	0	0	
9996	Adventure Time: Pirates of the Enchiridion	Jul 17, 2018	8	24.99	Summary Ahoy! The Land of Ooo is underwater an...	NaN	True	0	0	
9997	Action Alien: Tropical	Sep 1, 2018	0	3.99	Explore the sea with your boat and clear islan...	NaN	True	0	0	
9998	Retrowave Drive	Feb 3, 2021	0	1.99	Retrowave drive is an arcade	NaN	True	0	0	

	gameid	Name	Estimated owners	Required age	About the game	Reviews	Windows	Metacritic score	User score	Pos
					racing game in re...					
9999	NULL [Remastered]	Jun 27, 2023	1	9.99	You are trapped in a mansion with 8 other peop...	NaN	True	0	0	

10000 rows × 19 columns

In [3]:
```
df2.rename(columns={'gameid': 'title'}, inplace=True)
df2
```

Out[3]:

	title	Name	Estimated owners	Required age	About the game	Reviews	Windows	Metacritic score	User score	Pos
0	Surface: Project Dawn	Sep 15, 2023	1	9.99	The next spine-tingling installment of the Sur...	NaN	True	0	0	
1	Dis Pontibus	Jan 7, 2019	0	4.99	Dis Pontibus is a single-player puzzle game se...	NaN	True	0	0	
2	Furry Sweeper	Feb 20, 2023	1	2.69	Furry Sweeper is a variant of minesweeper wher...	NaN	True	0	0	
3	Street Hoop	Dec 31, 2019	0	4.99	Street Hoop is DATA EAST's 3 on 3 street baske...	NaN	True	0	0	
4	Flag Sweeper	Mar 24, 2023	0	0.00	This is a two player version of minesweeper wh...	NaN	True	0	0	
...	...	...	...	...	...	...	...	...	...	

	title	Name	Estimated owners	Required age	About the game	Reviews	Windows	Metacritic score	User score	Pos
9995	The End o,,,o	May 11, 2017	0	0.00	A short game about the day things got weird. Y...	NaN	True	0	0	
9996	Adventure Time: Pirates of the Enchiridion	Jul 17, 2018	8	24.99	Summary Ahoy! The Land of Ooo is underwater an...	NaN	True	0	0	
9997	Action Alien: Tropical	Sep 1, 2018	0	3.99	Explore the sea with your boat and clear islan...	NaN	True	0	0	
9998	Retrowave Drive	Feb 3, 2021	0	1.99	Retrowave drive is an arcade racing game in re...	NaN	True	0	0	
9999	NULL [Remastered]	Jun 27, 2023	1	9.99	You are trapped in a mansion with 8 other peop...	NaN	True	0	0	

10000 rows × 19 columns

In [4]:
```
joined_df = pd.merge(df, df2, on='title', how='outer')
joined_df
```

Out[4]:

	gameid	title	developers	publishers	genres	year	Name	Estimated owners	R
0	1417940.0	捕鱼炸翻天	['上海愉游网络科技有限公司']	['上海姚际信息科技有限公司']	['Adventure', 'Casual', 'Free To Play', 'Indie...	2020-10-26	Oct 26, 2020	1.0	

	gameid	title	developers	publishers	genres	year	Name	Estimated owners	R
1	1358770.0	The falling tower	['Carotaa']	['Dream Night Studio']	['Indie', 'RPG', 'Strategy']	2020-07-15	NaN	NaN	
2	652780.0	The Mutational	['DinjaStudios']	['DinjaStudios']	['Action', 'Adventure', 'Casual', 'Free To Pla...	2020-02-10	NaN	NaN	
3	437160.0	The Lion's Song: Episode 1 - Silence	["Mi'pu'mi Games GmbH"]	["Mi'pu'mi Games GmbH"]	['Adventure', 'Indie']	2016-07-07	NaN	NaN	
4	1619570.0	Death Roads: Tournament	['The Knights of Unity']	['The Knights of Unity', 'Surefire.Games']	['Indie', 'Racing', 'Strategy']	2023-11-15	NaN	NaN	
...	...	...	...	...	...	...	...	...	
18980	NaN	The End o,,,o	NaN	NaN	NaN	NaN	May 11, 2017	0.0	
18981	NaN	Adventure Time: Pirates of the Enchiridion	NaN	NaN	NaN	NaN	Jul 17, 2018	8.0	
18982	NaN	Action Alien: Tropical	NaN	NaN	NaN	NaN	Sep 1, 2018	0.0	
18983	NaN	Retrowave Drive	NaN	NaN	NaN	NaN	Feb 3, 2021	0.0	
18984	NaN	NULL [Remastered]	NaN	NaN	NaN	NaN	Jun 27, 2023	1.0	

18985 rows × 24 columns

In [5]:
```
df3 = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 St
df3
```

Out[5]:

	title	description	price	salePercentage	recentReviews	allReviews
0	Ori and the Will of the Wisps	Play the critically acclaimed masterpiece. Emb...	$9.89	-67%	Overwhelmingly Positive	Overwhelmingly Positive
1	Flashing Lights - Police, Firefighting, Emerge...	Play solo or in up to 10-player multiplayer co...	$8.49	-66%	Very Positive	Very Positive
2	Thronefall	A minimalist game about building and defending...	$5.24	-25%	Overwhelmingly Positive	Overwhelmingly Positive
3	DRAGON QUEST® XI S: Echoes of an Elusive Age™ ...	The Definitive Edition includes the critically...	$23.99	-40%	Very Positive	Very Positive
4	UNDYING	As Anling's zombie infection sets in, her days...	$13.99	-30%	Mostly Positive	Mostly Positive
...	...	...	...	...	...	...
81	Bendy and the Dark Revival	Bendy and the Dark Revival® is a first-person ...	$5.99	-80%	Very Positive	Very Positive
82	STAR WARS™ - The Force Unleashed™ Ultimate Sit...	A game that will show gamers the deepest, dark...	$6.99	-65%	Very Positive	Very Positive
83	Thymesia	Thymesia is a gruelling action-RPG with fast-p...	$14.99	-40%	Very Positive	Very Positive
84	Last Train Home	The Great War is over - the fight continues. C...	$26.39	-34%	Very Positive	Very Positive
85	Fallout 76	Bethesda Game Studios welcome you to Fallout 7...	$9.99	-75%	Mostly Positive	Mostly Positive

86 rows × 6 columns

In [6]:
```
df4 = pd.merge(joined_df, df3, on='title', how='outer')
df4
```

Out[6]:

	gameid	title	developers	publishers	genres	year	Name	Estimated owners	Re
0	1417940.0	捕鱼炸翻天	['上海愉游网络科技有限公司']	['上海姚际信息科技有限公司']	['Adventure', 'Casual', 'Free To Play', 'Indie...	2020-10-26	Oct 26, 2020	1.0	
1	1358770.0	The falling tower	['Carotaa']	['Dream Night Studio']	['Indie', 'RPG', 'Strategy']	2020-07-15	NaN	NaN	
2	652780.0	The Mutational	['DinjaStudios']	['DinjaStudios']	['Action', 'Adventure', 'Casual', 'Free To Pla...	2020-02-10	NaN	NaN	
3	437160.0	The Lion's Song: Episode 1 - Silence	["Mi'pu'mi Games GmbH"]	["Mi'pu'mi Games GmbH"]	['Adventure', 'Indie']	2016-07-07	NaN	NaN	
4	1619570.0	Death Roads: Tournament	['The Knights of Unity']	['The Knights of Unity', 'Surefire.Games']	['Indie', 'Racing', 'Strategy']	2023-11-15	NaN	NaN	
...	...	...	...	...	...	...	...	...	...
19050	NaN	STRANGER OF PARADISE FINAL FANTASY ORIGIN	NaN	NaN	NaN	NaN	NaN	NaN	
19051	NaN	Grounded	NaN	NaN	NaN	NaN	NaN	NaN	

	gameid	title	developers	publishers	genres	year	Name	Estimated owners	Re
19052	NaN	STAR WARS™ - The Force Unleashed™ Ultimate Sit...	NaN	NaN	NaN	NaN	NaN	NaN	
19053	NaN	Last Train Home	NaN	NaN	NaN	NaN	NaN	NaN	
19054	NaN	Fallout 76	NaN	NaN	NaN	NaN	NaN	NaN	

19055 rows × 29 columns

In [7]:
```
df4.columns.tolist()
```

Out[7]:
```
['gameid',
 'title',
 'developers',
 'publishers',
 'genres',
 'year',
 'Name',
 'Estimated owners',
 'Required age',
 'About the game',
 'Reviews',
 'Windows',
 'Metacritic score',
 'User score',
 'Positive',
 'Negative',
 'Achievements',
 'Recommendations',
 'Average playtime forever',
 'Average playtime two weeks',
 'Median playtime forever',
 'Median playtime two weeks',
 'Developers',
 'Genres',
 'description',
 'price',
 'salePercentage',
```

```
'recentReviews',
'allReviews']
```

In [8]:
```
df4.drop(columns=['Average playtime two weeks', 'Median playtime two weeks', 'recentRev
```

In [9]:
```
sampled_df4 = df4.sample(n=10000, random_state=42)
sampled_df4
```

Out[9]:

	gameid	title	developers	publishers	genres	year	Name	Estimated owners	Required age
9152	1657960.0	Anime puzzle	['wow wow Games']	['wow wow Games']	['Casual']	2021-06-24	NaN	NaN	NaN
6494	1762690.0	Epidemyc	['Ruben Dario Acosta']	['Ruben Dario Acosta']	['Indie', 'Strategy']	2021-10-08	NaN	NaN	NaN
4336	1120940.0	Forgotten Passages	['26PM']	['26PM']	['Adventure', 'Indie']	2020-01-16	NaN	NaN	NaN
15864	NaN	My Loving Wife	NaN	NaN	NaN	NaN	Dec 6, 2022	0.0	0.00
18474	NaN	Womb	NaN	NaN	NaN	NaN	Aug 30, 2023	0.0	5.99
...	...	...	...	...	...	...	...	...	...
1103	2092520.0	Hero Hours Contract 2: A Factory for Magical G...	["Steve O'Gorman"]	["Steve O'Gorman"]	['Indie', 'RPG', 'Strategy']	2022-09-07	NaN	NaN	NaN
11432	NaN	Doc Clock: The Toasted Sandwich of Time	NaN	NaN	NaN	NaN	Oct 15, 2010	0.0	4.99
11907	NaN	Caterpillar	NaN	NaN	NaN	NaN	Jun 2, 2022	0.0	4.99

	gameid	title	developers	publishers	genres	year	Name	Estimated owners	Required age
15053	NaN	Flyland Wars: 0 Ball Game [Trainer]	NaN	NaN	NaN	NaN	Oct 4, 2021	0.0	0.00
15399	NaN	Warlocks 2: God Slayers	NaN	NaN	NaN	NaN	Jul 18, 2019	0.0	19.99

10000 rows × 26 columns

In [10]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler


df = pd.read_csv(r'C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste


columns_to_drop = [
    'developers', 'genres', 'About the game', 'description',
    'price', 'salePercentage', 'allReviews'
]
df_cleaned = df.drop(columns=[col for col in columns_to_drop if col in df.columns])
if 'Price' in df_cleaned.columns:
    df_cleaned['Price'] = pd.to_numeric(df_cleaned['Price'], errors='coerce')

for col in ['year', 'Release Date']:
    if col in df_cleaned.columns:
        df_cleaned[col] = pd.to_datetime(df_cleaned[col], errors='coerce')


numeric_df = df_cleaned.select_dtypes(include=['float64', 'int64'])
scaler = StandardScaler()
scaled_values = scaler.fit_transform(numeric_df)
scaled_df = pd.DataFrame(scaled_values, columns=numeric_df.columns)


print(scaled_df.head())
scaled_df.to_csv(r'C:\Users\Tense\Documents\Flatiron\Data sets\games_scaled_output.csv'
scaled_df
```

```
     gameid  Estimated owners     Price  Metacritic score  User score  \
0  0.209261               NaN       NaN               NaN         NaN
1       NaN         -0.010516  0.207527          6.053743   -0.023137
2       NaN         -0.037736 -0.284235         -0.185636   -0.023137
3       NaN         -0.037736 -0.406561         -0.185636   -0.023137
4       NaN         -0.031985 -0.099825         -0.185636   -0.023137

   Positive  Negative  Achievements  Recommendations  \
0       NaN       NaN           NaN              NaN
1  0.469182  0.207920      0.271522         0.489813
```

```
2 -0.048458 -0.053788     -0.114456        -0.046369
3 -0.048458 -0.054534     -0.114456        -0.046369
4 -0.029777 -0.032165     -0.114456        -0.024707

   Average playtime forever  Median playtime forever
0                       NaN                      NaN
1                  1.085561                 1.230565
2                 -0.136247                -0.195329
3                 -0.136247                -0.195329
4                 -0.136247                -0.195329
```

Out[10]:

	gameid	Estimated owners	Price	Metacritic score	User score	Positive	Negative	Achievements	Recom
0	0.209261	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	-0.010516	0.207527	6.053743	-0.023137	0.469182	0.207920	0.271522	
2	NaN	-0.037736	-0.284235	-0.185636	-0.023137	-0.048458	-0.053788	-0.114456	
3	NaN	-0.037736	-0.406561	-0.185636	-0.023137	-0.048458	-0.054534	-0.114456	
4	NaN	-0.031985	-0.099825	-0.185636	-0.023137	-0.029777	-0.032165	-0.114456	
...	...	...	...	...	...	...	...	...	
9995	-0.622711	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9996	NaN	1.306004	1.436933	6.053743	-0.023137	0.597550	0.618004	0.058739	
9997	1.389141	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9998	1.673171	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9999	NaN	-0.037736	-0.099825	-0.185636	-0.023137	-0.048395	-0.054534	-0.025384	

10000 rows × 11 columns

In [11]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression

X = future_predictions[['Release Year']]
y = future_predictions[['Predicted Number of Games', 'Predicted Avg Price', 'Predicted
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=

model = LinearRegression()
model.fit(X_train, y_train)


y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-681d2e71db99> in <module>
```

```
      3 from sklearn.linear_model import LinearRegression
      4
----> 5 X = future_predictions[['Release Year']]
      6 y = future_predictions[['Predicted Number of Games', 'Predicted Avg Price', 'Predicted Total Owners']]
      7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

NameError: name 'future_predictions' is not defined
```

In [ ]:
```
df_scaled = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2(
df_scaled
```

Out[ ]:

	gameid	Estimated owners	Price	Metacritic score	User score	Positive	Negative	Achievements	Recom
0	0.209261	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	-0.010516	0.207527	6.053743	-0.023137	0.469182	0.207920	0.271522	
2	NaN	-0.037736	-0.284235	-0.185636	-0.023137	-0.048458	-0.053788	-0.114456	
3	NaN	-0.037736	-0.406561	-0.185636	-0.023137	-0.048458	-0.054534	-0.114456	
4	NaN	-0.031985	-0.099825	-0.185636	-0.023137	-0.029777	-0.032165	-0.114456	
...	...	...	...	...	...	...	...	...	
9995	-0.622711	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9996	NaN	1.306004	1.436933	6.053743	-0.023137	0.597550	0.618004	0.058739	
9997	1.389141	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9998	1.673171	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9999	NaN	-0.037736	-0.099825	-0.185636	-0.023137	-0.048395	-0.054534	-0.025384	

10000 rows × 11 columns

In [ ]:
```
df_filled = df.copy()
df_filled['price'] = df_filled['price'].replace('[\$,]', '', regex=True).astype(float)
df_filled['Primary Genre'] = df_filled['Genres'].dropna().apply(lambda x: x.split(',')[


median_price_by_genre = df_filled.groupby('Primary Genre')['price'].median()


def fill_price(row):
    if pd.isna(row['price']):
        genre = row['Primary Genre']
        if pd.notna(genre) and genre in median_price_by_genre:
            return median_price_by_genre[genre]
        else:
            return df_filled['price'].median()
    return row['price']

df_filled['price_filled'] = df_filled.apply(fill_price, axis=1)
```

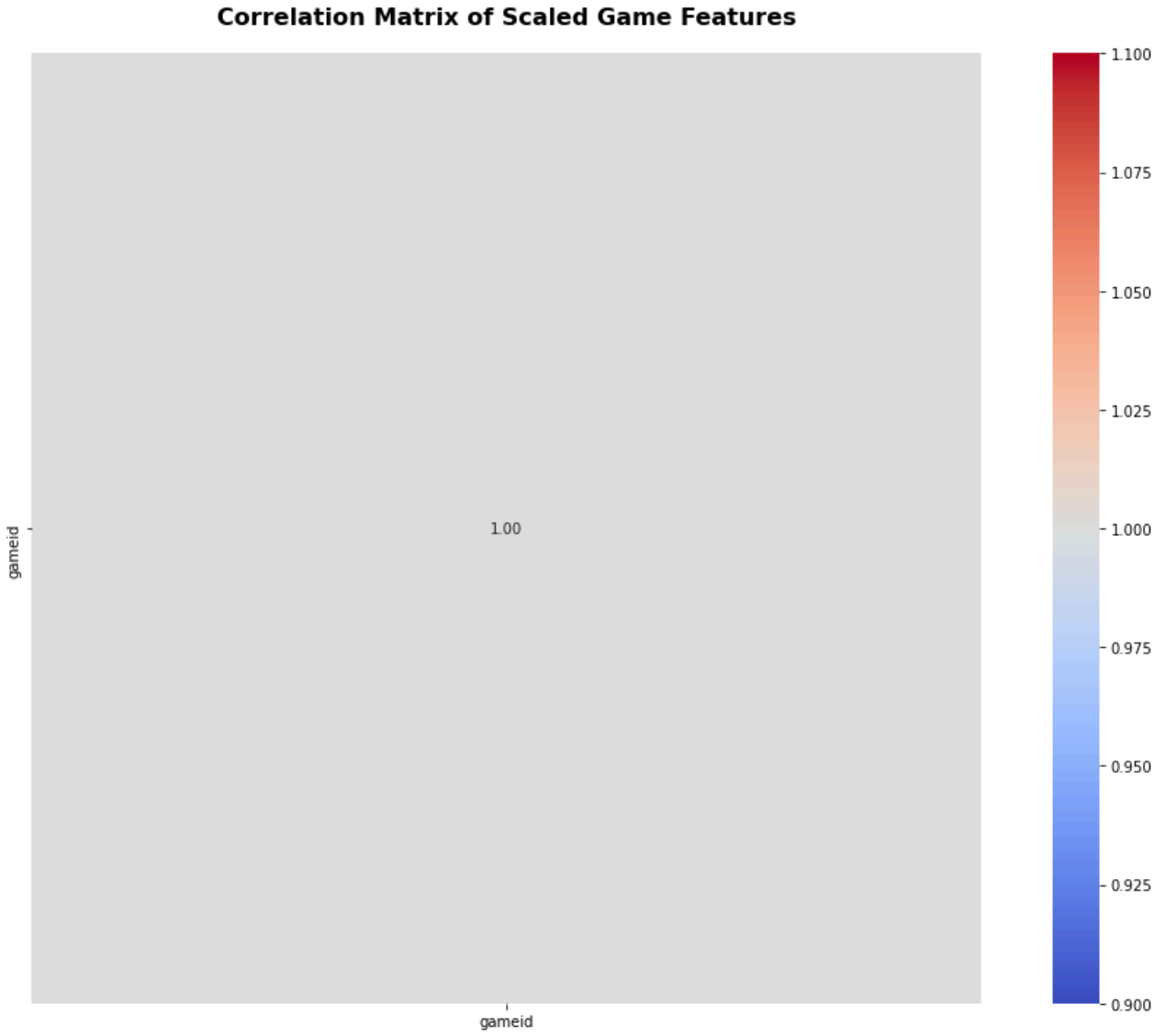
```
original_missing = df['price'].isna().sum()
filled_missing = df_filled['price_filled'].isna().sum()

print("Original missing prices:", original_missing)
print("Still missing after fill:", filled_missing)
```

```
Original missing prices: 9971
Still missing after fill: 2994
```

In [ ]:
```
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor


price_features = [
    'Recommendations', 'Achievements', 'Average playtime forever',
    'Median playtime forever', 'Estimated owners'
]

df_price_input = df_filled[price_features + ['price']].copy()

imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(df_price_input[price_features])

mask_train = df_price_input['price'].notna()
X_train_price = X_imputed[mask_train]
y_train_price = df_price_input.loc[mask_train, 'price']

model = RandomForestRegressor(random_state=42)
model.fit(X_train_price, y_train_price)

mask_missing = df_price_input['price'].isna()
X_missing_price = X_imputed[mask_missing]
predicted_prices = model.predict(X_missing_price)

df_filled.loc[mask_missing, 'price_predicted'] = predicted_prices
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

columns_to_drop = [
    'developers', 'genres', 'About the game', 'description',
    'price', 'salePercentage', 'allReviews'
]
df_cleaned = df.drop(columns=[col for col in columns_to_drop if col in df.columns])


if 'Price' in df_cleaned.columns:
    df_cleaned['Price'] = pd.to_numeric(df_cleaned['Price'], errors='coerce')


for col in ['year', 'Release Date']:
    if col in df_cleaned.columns:
        df_cleaned[col] = pd.to_datetime(df_cleaned[col], errors='coerce')


numeric_df = df_cleaned.select_dtypes(include=['float64', 'int64'])

scaler = StandardScaler()
scaled_values = scaler.fit_transform(numeric_df)
scaled_df = pd.DataFrame(scaled_values, columns=numeric_df.columns)
```

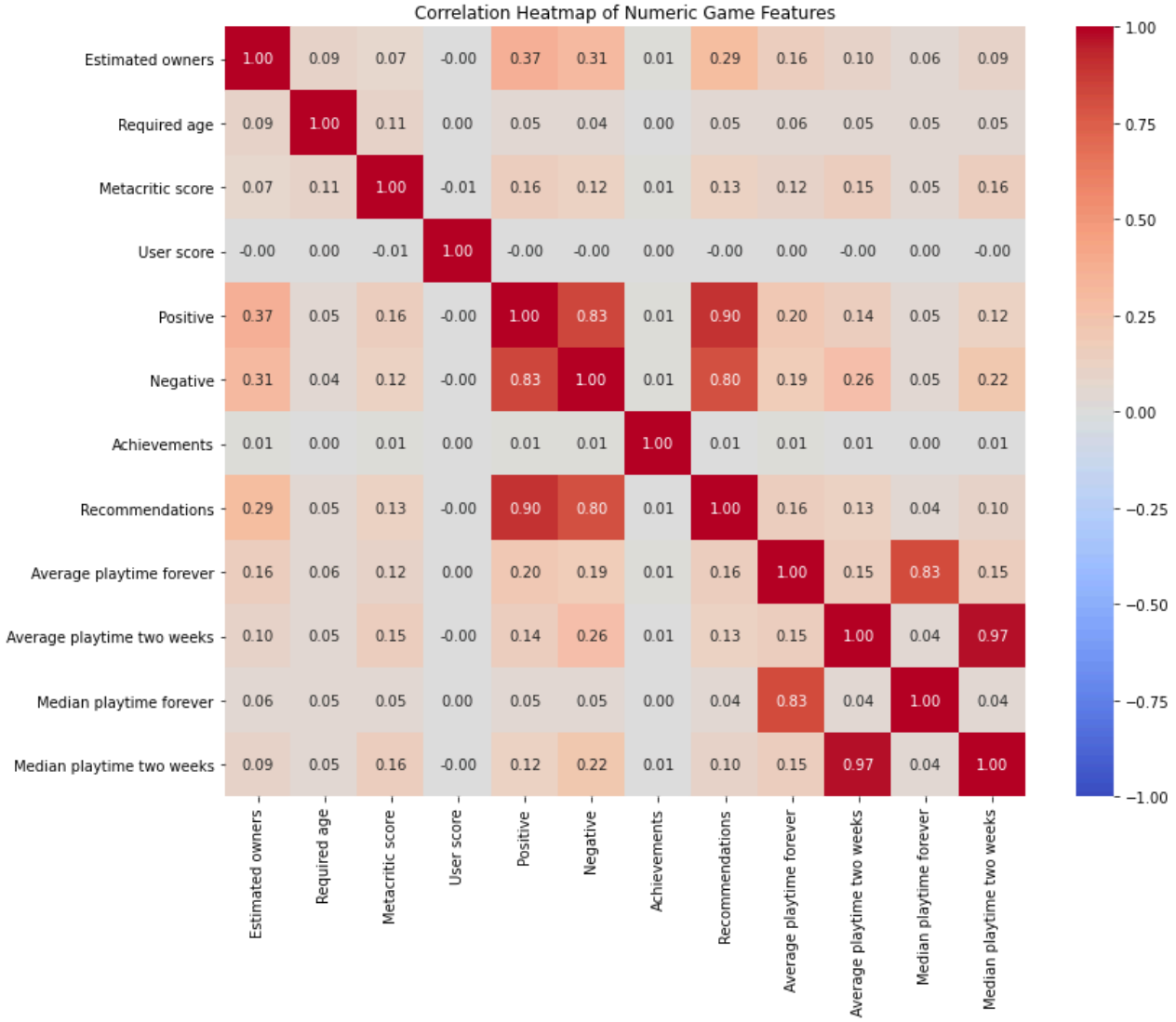
```
correlation_matrix = scaled_df.corr()

plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title("Correlation Matrix of Scaled Game Features", fontsize=16, weight='bold', pad

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



In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np


df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste
numeric_df = df.select_dtypes(include=['number'])

corr_matrix = numeric_df.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Correlation Heatmap of Numeric Game Features")
plt.tight_layout()
```

```
plt.show()

strong_corrs = (
    corr_matrix.where(~np.eye(corr_matrix.shape[0], dtype=bool))
    .stack()
    .reset_index()
)
strong_corrs.columns = ["Feature 1", "Feature 2", "Correlation"]
strong_corrs = strong_corrs[strong_corrs["Correlation"].abs() > 0.6]
```



In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score


indie_games = sampled_df4[sampled_df4['Genres'].str.contains('Indie', na=False)]
indie_games['year'] = pd.to_datetime(indie_games['Release Date'], errors='coerce').dt.y
indie_games = indie_games.dropna(subset=['year', 'Price', 'Estimated owners'])

X = indie_games[['year']]
y = indie_games[['Price', 'Estimated owners']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

rf_model = RandomForestRegressor(random_state=42, n_estimators=100)
```

```
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)


mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest Mean Squared Error: {mse_rf}")
print(f"Random Forest R^2 Score: {r2_rf}")

future_years = pd.DataFrame({'year': range(2026, 2031)})
future_predictions_rf = rf_model.predict(future_years)
future_years['Predicted Price (RF)'] = future_predictions_rf[:, 0]
future_years['Predicted Estimated Owners (RF)'] = future_predictions_rf[:, 1]

print(future_years)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\_libs\index.pyx in pand
as._libs.index.IndexEngine.get_loc()

c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\_libs\index.pyx in pand
as._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_
item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_
item()

KeyError: 'Release Date'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
<ipython-input-18-92cf6a904d1b> in <module>
      5
      6 indie_games = sampled_df4[sampled_df4['Genres'].str.contains('Indie', na=False)]
----> 7 indie_games['year'] = pd.to_datetime(indie_games['Release Date'], errors='coerc
e').dt.year
      8 indie_games = indie_games.dropna(subset=['year', 'Price', 'Estimated owners'])
      9

c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py in __geti
tem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexes\base.py in
get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
```

```
3632             except TypeError:
3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Release Date'
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("games cleaned scaled.csv")

features = ['Recommendations', 'Achievements', 'Average playtime forever', 'Median play
target = 'Estimated owners'

df_model = df[features + [target]].dropna()

X = df_model[features]
y = df_model[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R^2 Score:", r2)
```

```
Mean Squared Error: 1015620.760149249
R^2 Score: 0.4590666886042214
```

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(future_years['year'], future_years['Predicted Price'], label='Predicted Price'
plt.plot(future_years['year'], future_years['Predicted Estimated Owners'], label='Predi
plt.xlabel('Year')
plt.ylabel('Predicted Values')
plt.title('Future Predictions: Price and Estimated Owners')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-05de2f1df1ef> in <module>
      2
      3 plt.figure(figsize=(10, 6))
----> 4 plt.plot(future_years['year'], future_years['Predicted Price'], label='Predicted
Price', marker='o')
      5 plt.plot(future_years['year'], future_years['Predicted Estimated Owners'], label
='Predicted Estimated Owners', marker='o')
      6 plt.xlabel('Year')

NameError: name 'future_years' is not defined
<Figure size 720x432 with 0 Axes>
```

In [ ]:
```
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste

relevant_columns = ['Estimated owners', 'Price', 'Metacritic score', 'User score',
                    'Positive', 'Negative', 'Achievements', 'Recommendations',
                    'Average playtime forever', 'Median playtime forever']
clean_df = df[relevant_columns].dropna()


scaler = StandardScaler()
scaled_data = scaler.fit_transform(clean_df)
svdd = OneClassSVM(kernel='rbf', gamma='auto', nu=0.05)
svdd.fit(scaled_data)

predictions = svdd.predict(scaled_data)

clean_df['SVDD_Prediction'] = predictions

inliers = clean_df[clean_df['SVDD_Prediction'] == 1]
outliers = clean_df[clean_df['SVDD_Prediction'] == -1]


plt.figure(figsize=(10, 6))
plt.scatter(inliers['Price'], inliers['Estimated owners'], label='Normal Games', c='blu
plt.scatter(outliers['Price'], outliers['Estimated owners'], label='Anomalous Games', c
plt.xlabel('Price (scaled)')
plt.ylabel('Estimated Owners (scaled)')
plt.title('SVDD: Indie Games - High/Underperformers')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-21-53b98ea54ec7> in <module>
      9                     'Positive', 'Negative', 'Achievements', 'Recommendations',
     10                     'Average playtime forever', 'Median playtime forever']
---> 11 clean_df = df[relevant_columns].dropna()
     12
     13

c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py in __geti
tem__(self, key)
   3509             if is_iterator(key):
   3510                 key = list(key)
-> 3511             indexer = self.columns._get_indexer_strict(key, "columns")[1]
   3512
   3513         # take() does not accept boolean indexers

c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexes\base.py in
_get_indexer_strict(self, key, axis_name)
   5794             keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
   5795
-> 5796         self._raise_if_missing(keyarr, indexer, axis_name)
   5797
   5798         keyarr = self.take(indexer)
```

```
c:\Users\Tense\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexes\base.py in
_raise_if_missing(self, key, indexer, axis_name)
   5854                 if use_interval_msg:
   5855                     key = list(key)
-> 5856                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   5857
   5858             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique
())

KeyError: "None of [Index(['Estimated owners', 'Price', 'Metacritic score', 'User scor
e',\n       'Positive', 'Negative', 'Achievements', 'Recommendations',\n       'Average
playtime forever', 'Median playtime forever'],\n      dtype='object')] are in the [colum
ns]"
```

In [ ]:

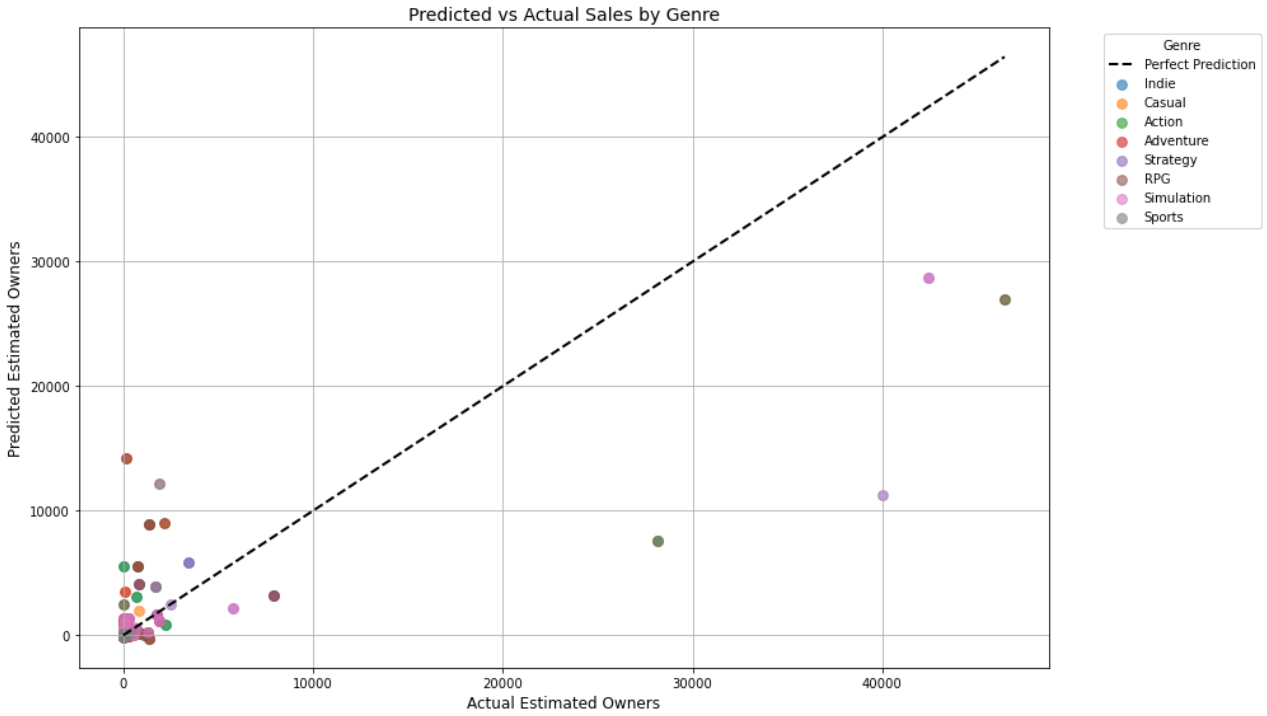
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste


df['Is_Indie'] = df['Genres'].fillna('').apply(lambda x: 'Indie' in x)
df['Release_Date'] = pd.to_datetime(df['Name'], errors='coerce')
df['Release_Year'] = df['Release_Date'].dt.year
df['Release_Month'] = df['Release_Date'].dt.month
df['Windows'] = df['Windows'].astype(int)
df['Estimated owners'] = pd.to_numeric(df['Estimated owners'], errors='coerce')


df = df.dropna(subset=['Estimated owners'])


interaction_cols = [
    'Positive', 'Negative', 'Recommendations',
    'Average playtime forever', 'Median playtime forever'
]
df_filtered = df[(df[interaction_cols].sum(axis=1)) > 0]

features = [
    'Required age', 'Windows', 'Metacritic score', 'User score',
    'Positive', 'Negative', 'Achievements', 'Recommendations',
    'Average playtime forever', 'Median playtime forever',
    'Release_Year', 'Release_Month', 'Is_Indie'
]
X = df_filtered[features].fillna(0)
y = df_filtered['Estimated owners']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

model = GradientBoostingRegressor(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
Mean Squared Error: 1691736.02
R² Score: 0.6301
```

In [ ]:
```
import matplotlib.pyplot as plt
genre_keywords = ['Indie', 'Casual', 'Action', 'Adventure', 'Strategy', 'RPG', 'Simulat


for genre in genre_keywords:
    df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)


genre_flags = df_filtered.loc[X_test.index, genre_keywords]


X_test_plot = X_test.copy()
X_test_plot['Predicted'] = y_pred
X_test_plot['Actual'] = y_test.values


X_test_plot = pd.concat([X_test_plot, genre_flags], axis=1)


plt.figure(figsize=(14, 8))

for genre in genre_keywords:
    genre_subset = X_test_plot[X_test_plot[genre] == True]
    plt.scatter(
        genre_subset['Actual'],
        genre_subset['Predicted'],
        alpha=0.6,
        s=60,
        label=genre
    )


max_val = max(X_test_plot['Actual'].max(), X_test_plot['Predicted'].max())
plt.plot([0, max_val], [0, max_val], 'k--', lw=2, label='Perfect Prediction')


plt.xlabel('Actual Estimated Owners', fontsize=12)
plt.ylabel('Predicted Estimated Owners', fontsize=12)
plt.title('Predicted vs Actual Sales by Genre', fontsize=14)
plt.legend(title='Genre', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
```

```
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
<ipython-input-23-76effce1da16>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered[genre] = df_filtered['Genres'].fillna('').apply(lambda x: genre in x)
```

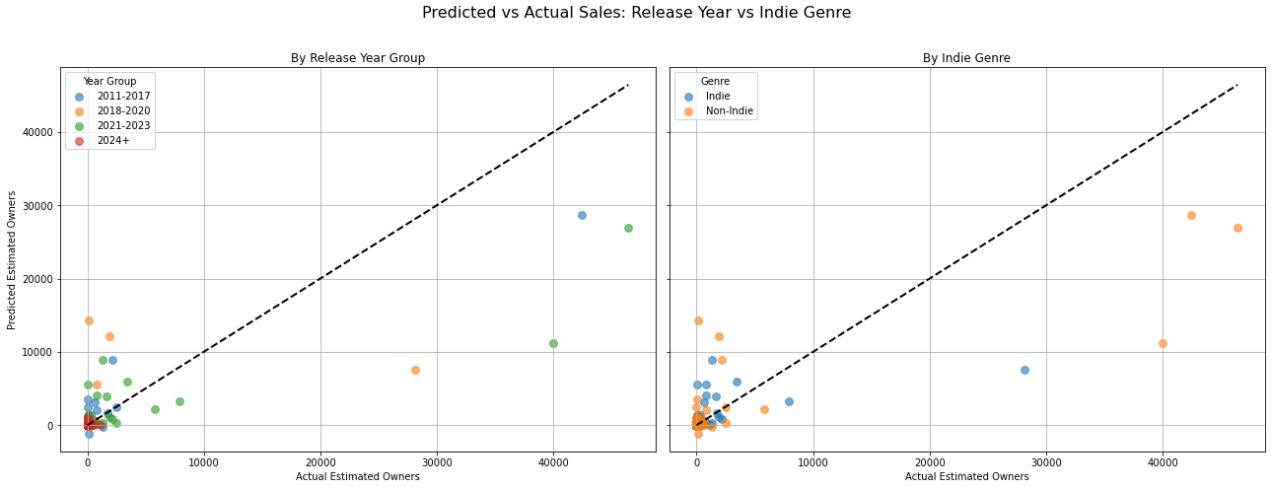


In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd


bins = [2010, 2017, 2020, 2023, 2025]
labels = ['2011-2017', '2018-2020', '2021-2023', '2024+']

df_filtered['Release_Year_Group'] = pd.cut(df_filtered['Release_Year'], bins=bins, labe


year_groups = df_filtered.loc[X_test.index, 'Release_Year_Group']
X_test_plot['Release_Year_Group'] = year_groups


fig, axes = plt.subplots(1, 2, figsize=(18, 7), sharey=True)


for group in labels:
    group_subset = X_test_plot[X_test_plot['Release_Year_Group'] == group]
    axes[0].scatter(
        group_subset['Actual'],
        group_subset['Predicted'],
        alpha=0.6,
        s=60,
        label=group
    )

max_val = max(X_test_plot['Actual'].max(), X_test_plot['Predicted'].max())
axes[0].plot([0, max_val], [0, max_val], 'k--', lw=2)
axes[0].set_title('By Release Year Group')
axes[0].set_xlabel('Actual Estimated Owners')
axes[0].set_ylabel('Predicted Estimated Owners')
axes[0].legend(title='Year Group')
axes[0].grid(True)
```

```
axes[1].scatter(
    X_test_plot[X_test_plot['Indie'] == True]['Actual'],
    X_test_plot[X_test_plot['Indie'] == True]['Predicted'],
    alpha=0.6,
    s=60,
    label='Indie'
)
axes[1].scatter(
    X_test_plot[X_test_plot['Indie'] == False]['Actual'],
    X_test_plot[X_test_plot['Indie'] == False]['Predicted'],
    alpha=0.6,
    s=60,
    label='Non-Indie'
)
axes[1].plot([0, max_val], [0, max_val], 'k--', lw=2)
axes[1].set_title('By Indie Genre')
axes[1].set_xlabel('Actual Estimated Owners')
axes[1].legend(title='Genre')
axes[1].grid(True)


plt.suptitle('Predicted vs Actual Sales: Release Year vs Indie Genre', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

```
<ipython-input-24-6d58651a126b>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered['Release_Year_Group'] = pd.cut(df_filtered['Release_Year'], bins=bins, lab
els=labels)
```

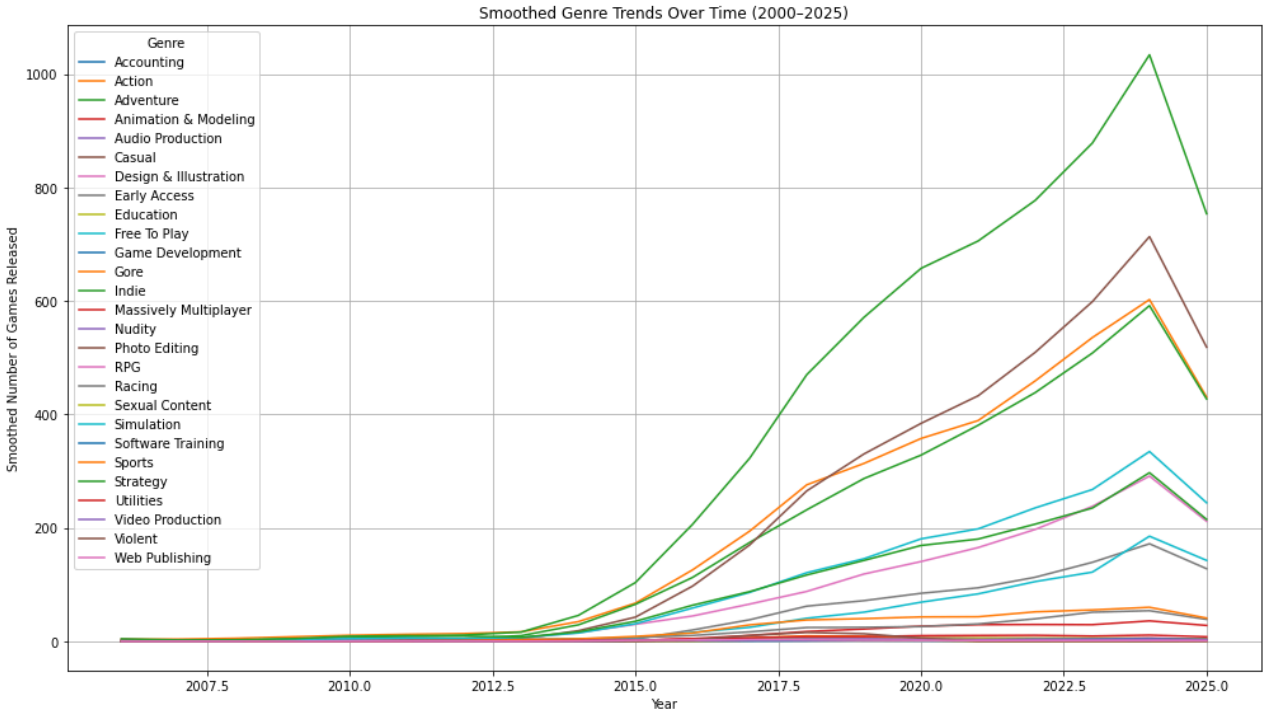


The predictions generally follow the perfect prediction line, but the spread increases slightly for older games, suggesting more variability or uncertainty in those predictions.

In [ ]:

```
import ast

df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste
df["genres"] = df["genres"].apply(ast.literal_eval)
df["release_year"] = pd.to_datetime(df["year"], errors='coerce').dt.year
df = df.dropna(subset=["release_year"])

df_exploded = df.explode("genres")
```

```
genre_trends = df_exploded.groupby(["release_year", "genres"]).size().reset_index(name=

genre_pivot = genre_trends.pivot(index='release_year', columns='genres', values='count'


genre_pivot_smooth = genre_pivot.rolling(window=3, min_periods=1).mean()


plt.figure(figsize=(14, 8))
for genre in genre_pivot_smooth.columns:
    plt.plot(genre_pivot_smooth.index, genre_pivot_smooth[genre], label=genre)

plt.title('Smoothed Genre Trends Over Time (2000-2025)')
plt.xlabel('Year')
plt.ylabel('Smoothed Number of Games Released')
plt.legend(title='Genre')
plt.grid(True)
plt.tight_layout()
plt.show()
```

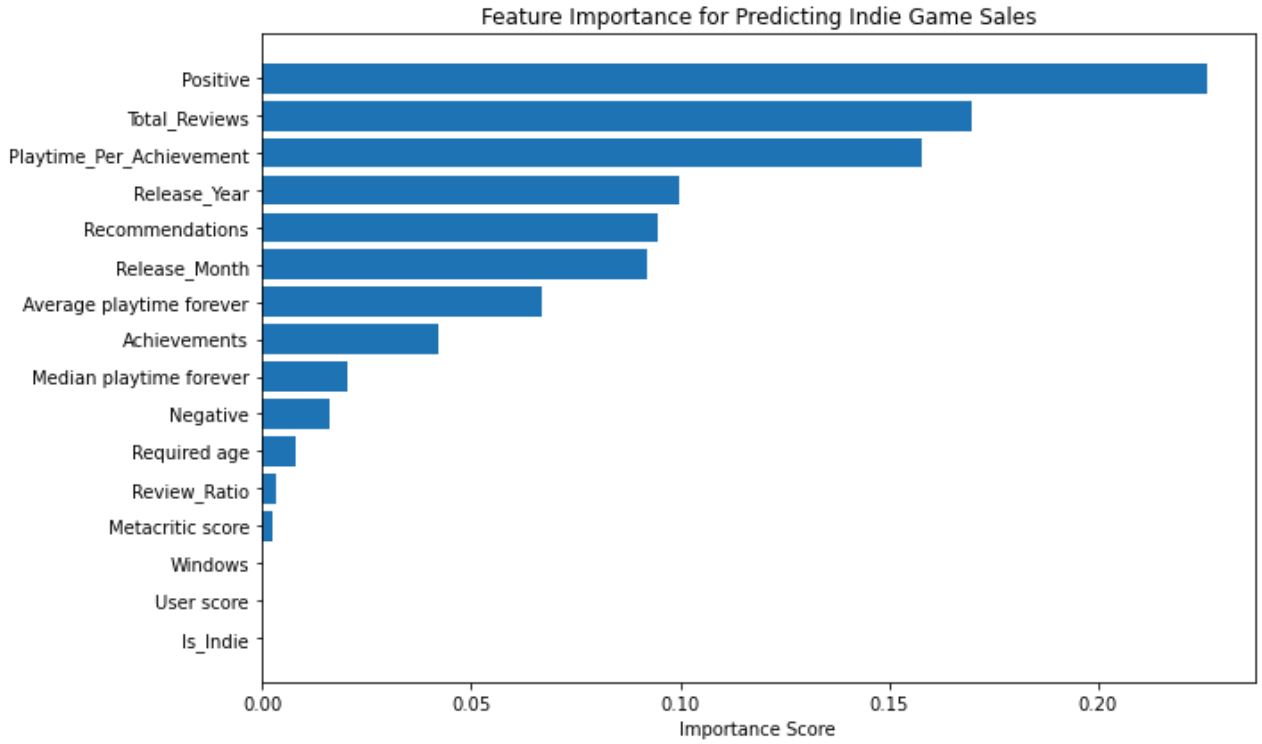


Indie and Casual genres show the most significant growth over time. Indie sees a steep and consistent rise from around 2010, peaking just before 2025. This reflects the surge in self-publishing tools. Casual also sees a steep incline, suggesting the expansion of accessible and mobile-friendly games.

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split


indie_data = df_filtered[df_filtered['Indie'] == True].copy()
```

```
indie_data['Total_Reviews'] = indie_data['Positive'] + indie_data['Negative']
indie_data['Review_Ratio'] = indie_data['Positive'] / (indie_data['Total_Reviews'] + 1)
indie_data['Playtime_Per_Achievement'] = indie_data['Average playtime forever'] / (indi


numeric_cols = indie_data.select_dtypes(include=[np.number]).columns
indie_data[numeric_cols] = indie_data[numeric_cols].replace([np.inf, -np.inf], 0)
indie_data[numeric_cols] = indie_data[numeric_cols].fillna(0)


features = [
    'Required age', 'Windows', 'Metacritic score', 'User score',
    'Positive', 'Negative', 'Achievements', 'Recommendations',
    'Average playtime forever', 'Median playtime forever',
    'Release_Year', 'Release_Month', 'Is_Indie'
]
enhanced_features = features + [
    'Total_Reviews',
    'Review_Ratio',
    'Playtime_Per_Achievement'
]


X_enhanced = indie_data[enhanced_features]
y_enhanced = indie_data['Estimated owners']


enhanced_model = GradientBoostingRegressor(random_state=42)
enhanced_model.fit(X_enhanced, y_enhanced)


importances = enhanced_model.feature_importances_
feature_names = X_enhanced.columns

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.gca().invert_yaxis()
plt.xlabel('Importance Score')
plt.title('Feature Importance for Predicting Indie Game Sales')
plt.tight_layout()
plt.show()
```

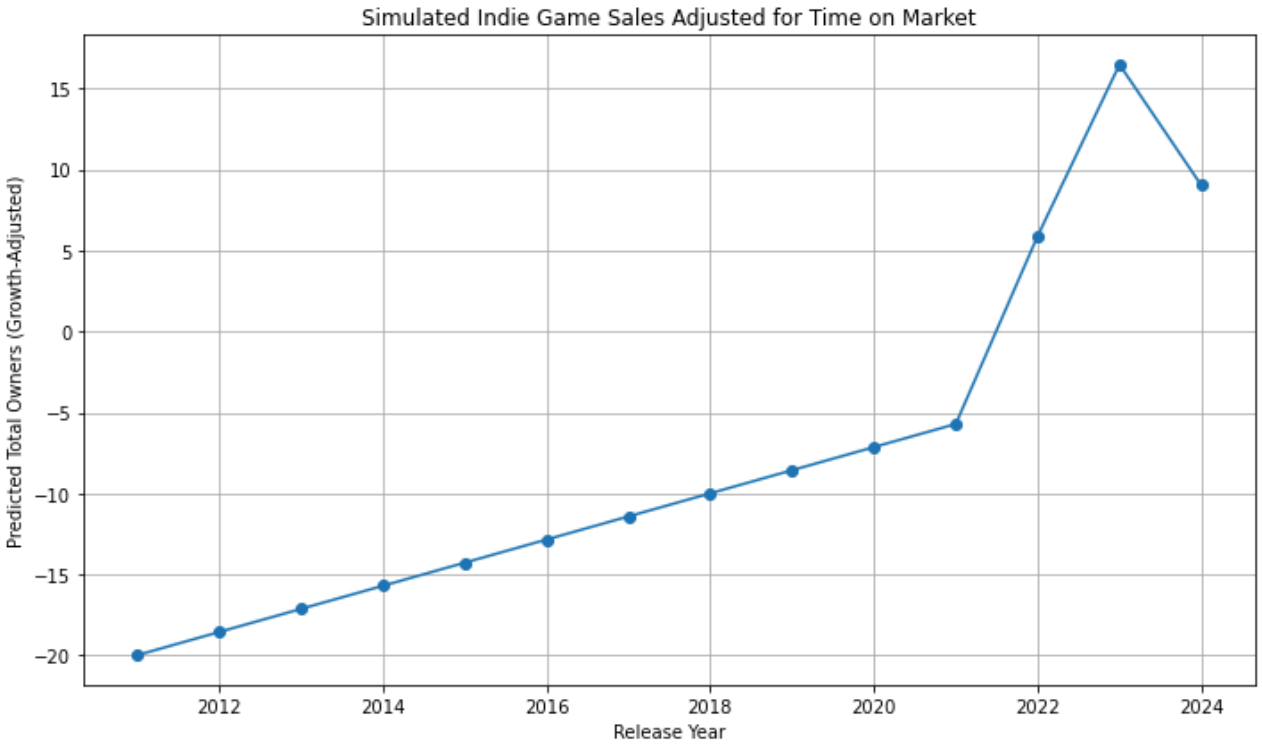


Investegating what aspect of the dataset will give me the best results

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingRegressor

indie_data = df_filtered[df_filtered['Indie'] == True].copy()
current_year = 2025
indie_data['Years_Since_Release'] = current_year - indie_data['Release_Year']
indie_data['Years_Since_Release'] = indie_data['Years_Since_Release'].clip(lower=1)


indie_data['Sales_Per_Year'] = indie_data['Estimated owners'] / indie_data['Years_Since


indie_data['Total_Reviews'] = indie_data['Positive'] + indie_data['Negative']
indie_data['Review_Ratio'] = indie_data['Positive'] / (indie_data['Total_Reviews'] + 1)
indie_data['Playtime_Per_Achievement'] = indie_data['Average playtime forever'] / (indi


numeric_cols = indie_data.select_dtypes(include=[np.number]).columns
indie_data[numeric_cols] = indie_data[numeric_cols].replace([np.inf, -np.inf], 0)
indie_data[numeric_cols] = indie_data[numeric_cols].fillna(0)


features = [
    'Required age', 'Windows', 'Metacritic score', 'User score',
    'Positive', 'Negative', 'Achievements', 'Recommendations',
    'Average playtime forever', 'Median playtime forever',
    'Release_Year', 'Release_Month', 'Is_Indie'
]
enhanced_features = features + ['Total_Reviews', 'Review_Ratio', 'Playtime_Per_Achievem
X_growth = indie_data[enhanced_features]
y_growth = indie_data['Sales_Per_Year']
```

```
growth_model = GradientBoostingRegressor(random_state=42)
growth_model.fit(X_growth, y_growth)


base_game = indie_data[enhanced_features].median()
years = list(range(2011, 2025))
simulated_growth = []

for year in years:
    game = base_game.copy()
    game['Release_Year'] = year
    game['Release_Month'] = 6
    game['Is_Indie'] = 1
    simulated_growth.append(game)

simulated_df = pd.DataFrame(simulated_growth)
predicted_growth_per_year = growth_model.predict(simulated_df)
predicted_total_owners = predicted_growth_per_year * np.clip((2025 - np.array(years)),


plt.figure(figsize=(10, 6))
plt.plot(years, predicted_total_owners, marker='o')
plt.xlabel('Release Year')
plt.ylabel('Predicted Total Owners (Growth-Adjusted)')
plt.title('Simulated Indie Game Sales Adjusted for Time on Market')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Time Matters, but Growth Is Key The model now predicts yearly growth in owners, not just raw totals.

This removes the bias where older games seem more successful just because they've been around longer.

Now, the focus is on how well a game performs year-over-year, regardless of age.

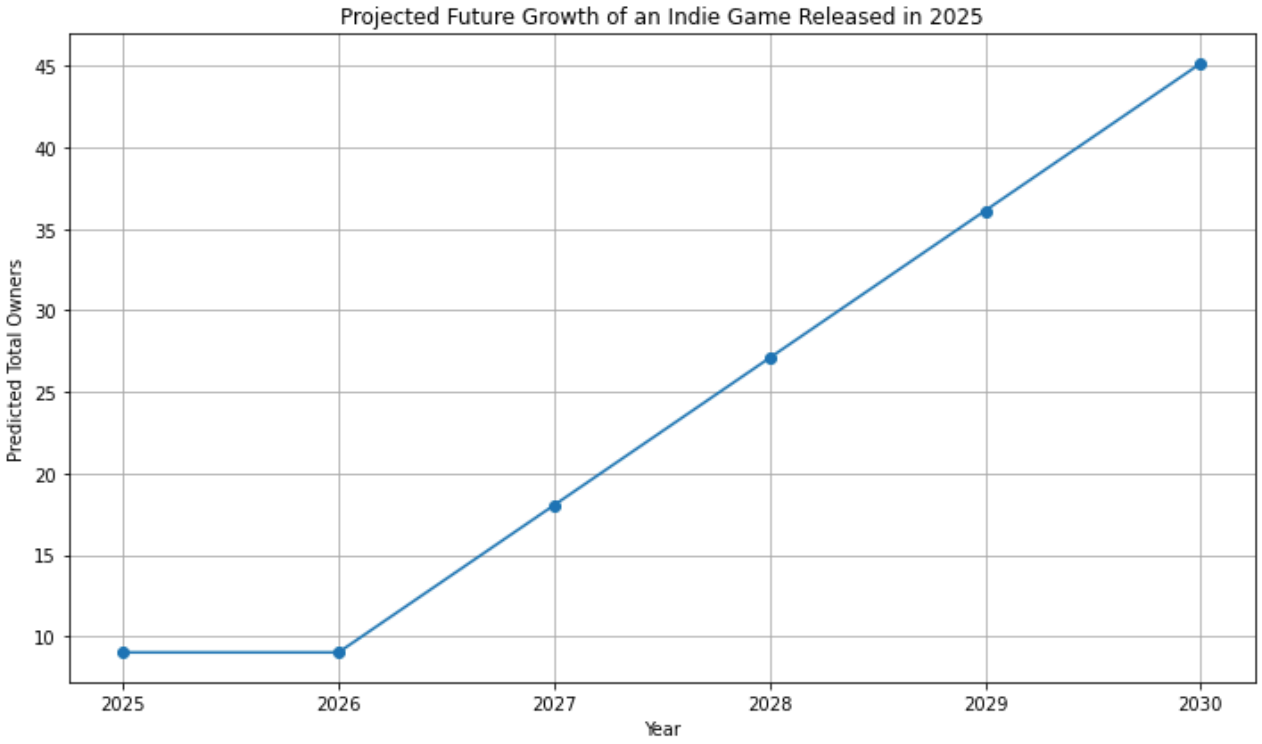
Older does not mean Better In your earlier raw sales model, older games always looked better.

In this model, that effect is mostly corrected. A new game can show just as much potential as an older one if its engagement signals (reviews, playtime, etc.) are strong.

Marketing Signals Still Dominate Even in a time-adjusted model, positive reviews, playtime, and recommendations are the strongest predictors of success.

This suggests that what drives growth isn't just when a game is released — it's how players respond to it once it launches.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


base_game = indie_data[enhanced_features].median()
future_years = list(range(2025, 2031))
simulated_future = []


for current_year in future_years:
    years_since_release = current_year - 2025
    years_since_release = max(1, years_since_release)


    game = base_game.copy()
    game['Release_Year'] = 2025
    game['Release_Month'] = 6
    game['Is_Indie'] = 1


    growth_per_year = growth_model.predict(pd.DataFrame([game]))[0]
    total_growth = growth_per_year * years_since_release


    simulated_future.append({
        'Year': current_year,
        'Predicted_Total_Owners': total_growth
    })

future_df = pd.DataFrame(simulated_future)

plt.figure(figsize=(10, 6))
plt.plot(future_df['Year'], future_df['Predicted_Total_Owners'], marker='o')
plt.xlabel('Year')
plt.ylabel('Predicted Total Owners')
plt.title('Projected Future Growth of an Indie Game Released in 2025')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Steady, Linear Growth The line rises consistently year after year, indicating the model expects this game to attract new players at a stable rate.

This mirrors the behavior of many successful Indie titles that grow through word-of-mouth, reviews, updates, and long-term community support.

The lack of big jumps suggests the model isn't accounting for viral success or sudden declines — it assumes organic, gradual growth.

This makes it a conservative forecast, which is helpful for planning but may underestimate breakout hits.

Model Confidence in Base Profile Because the base game is built from median values (average engagement, reviews, etc.), the model treats it as a solid, mid-tier Indie release.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


base_game = indie_data[enhanced_features].median()

future_years = list(range(2025, 2031))
simulated_sustained = []


for i, year in enumerate(future_years):
    years_since_release = year - 2025
    years_since_release = max(1, years_since_release)

    game = base_game.copy()
    game['Release_Year'] = 2025
    game['Release_Month'] = 6
```

```
    game['Is_Indie'] = 1


    if year >= 2026:
        game['Recommendations'] += 100
        game['Recommendations'] += 10 * (years_since_release - 1)

    growth_per_year = growth_model.predict(pd.DataFrame([game]))[0]
    total_owners = growth_per_year * years_since_release

    simulated_sustained.append({
        'Year': year,
        'Predicted_Total_Owners': total_owners
    })


sustained_df = pd.DataFrame(simulated_sustained)

plt.figure(figsize=(10, 6))
plt.plot(sustained_df['Year'], sustained_df['Predicted_Total_Owners'], marker='o', labe
plt.xlabel('Year')
plt.ylabel('Predicted Total Owners')
plt.title('Projected Growth with Sustained Marketing Campaign')
plt.grid(True)
plt.tight_layout()
plt.legend()
plt.show()
```

Growth trend

After 2026, the curve begins to steepen, showing that continued attention leads to increasing returns.

Spike Creates a New Baseline The one-time marketing push in 2026 dramatically improves the trajectory.

Sustained Momentum Beats One-Offs The ongoing yearly increase in recommendations means each new year builds on the last.

By 2030, the predicted total owners are much higher than if marketing had only spiked once or not at all.

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report


games_df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 20
games_df["Recommended_flag"] = (games_df["Recommendations"] > 0).astype(int)

features = [
    "Estimated owners", "Required age", "Metacritic score", "User score",
    "Positive", "Negative", "Achievements", "Average playtime forever",
    "Average playtime two weeks", "Median playtime forever", "Median playtime two weeks
    "Windows"
]


model_df = games_df[features + ["Recommended_flag"]].dropna()
X = model_df[features]
y = model_df["Recommended_flag"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4


param_grid = {
    'n_estimators': [100, 150],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0]
}

gbc = GradientBoostingClassifier(random_state=42)
grid_search = GridSearchCV(gbc, param_grid, cv=3, scoring='f1', n_jobs=-1)
grid_search.fit(X_train, y_train)


best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Best Parameters:", grid_search.best_params_)
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

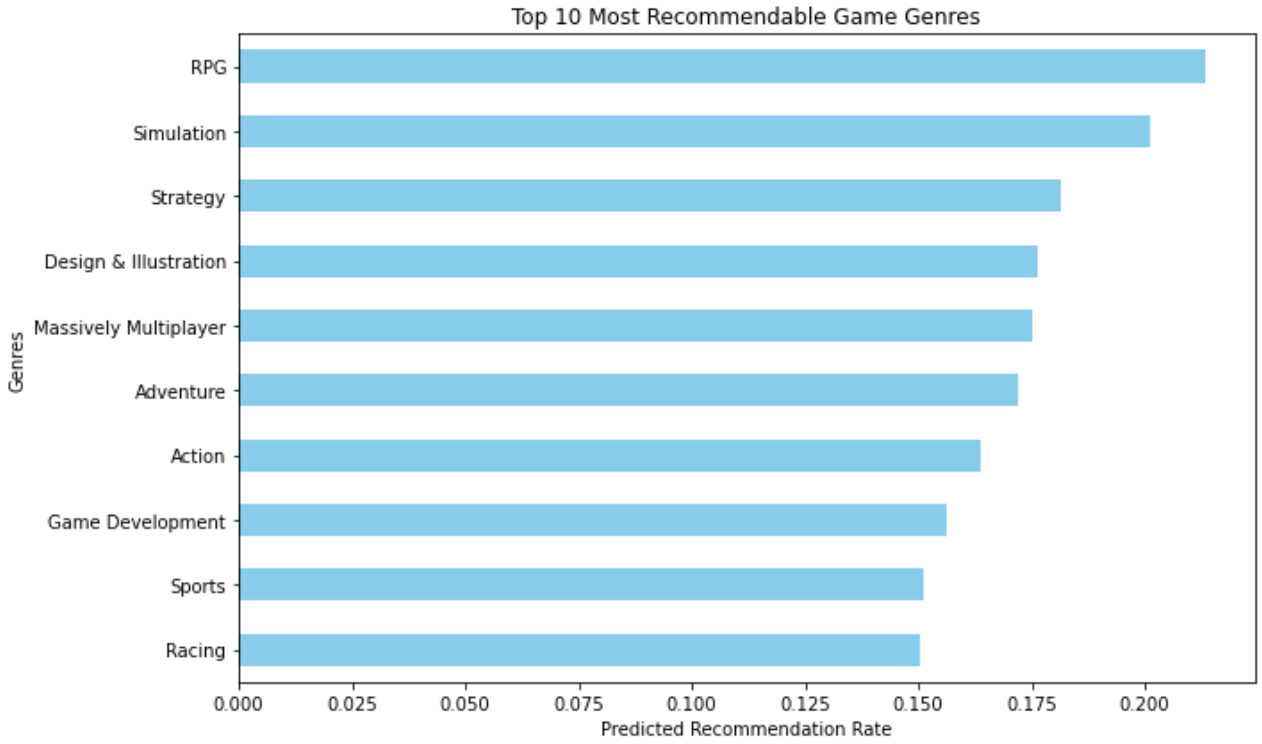
```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'subsampl
e': 0.8}
Accuracy: 0.9780
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99      1721
           1       0.94      0.90      0.92       279
```

```
              accuracy                           0.98      2000
             macro avg       0.96      0.95      0.95      2000
          weighted avg       0.98      0.98      0.98      2000
```

In [ ]:
```
df["predicted_recommendation"] = best_model.predict(X)


df_exploded = df[["Genres", "predicted_recommendation"]].dropna()
df_exploded["Genres"] = df_exploded["Genres"].str.split(",")
df_exploded = df_exploded.explode("Genres")


genre_stats = df_exploded.groupby("Genres")["predicted_recommendation"].agg(["mean", "c
genre_stats = genre_stats.rename(columns={"mean": "recommendation_rate", "count": "game


popular_genres = genre_stats[genre_stats["game_count"] >= 20]
popular_genres_sorted = popular_genres.sort_values("recommendation_rate", ascending=Fal

print(popular_genres_sorted.head(10))
```

```
                      recommendation_rate  game_count
Genres
RPG                                0.213510        1658
Simulation                         0.201168        1884
Strategy                           0.181325        1842
Design & Illustration              0.176471          51
Massively Multiplayer              0.175097         257
Adventure                          0.171845        3701
Action                             0.163762        3902
Game Development                   0.156250          32
Sports                             0.151242         443
Racing                             0.150142         353
```

In [ ]:
```
import matplotlib.pyplot as plt


df["predicted_recommendation"] = best_model.predict(X)
df_exploded = df[["Genres", "predicted_recommendation"]].dropna()
df_exploded["Genres"] = df_exploded["Genres"].str.split(",")
df_exploded = df_exploded.explode("Genres")


genre_stats = df_exploded.groupby("Genres")["predicted_recommendation"].agg(["mean", "c
genre_stats = genre_stats.rename(columns={"mean": "recommendation_rate", "count": "game


popular_genres = genre_stats[genre_stats["game_count"] >= 20]
popular_genres_sorted = popular_genres.sort_values("recommendation_rate", ascending=Fal


plt.figure(figsize=(10, 6))
popular_genres_sorted["recommendation_rate"].head(10).plot(kind="barh", color="skyblue"
plt.gca().invert_yaxis()
plt.xlabel("Predicted Recommendation Rate")
plt.title("Top 10 Most Recommendable Game Genres")
plt.tight_layout()
plt.show()
```



If an indie game were to be made, RPG, Simulation, and strategy are the top choices for a continously growing market.

In [ ]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\tense\anaconda3\envs\learn-e
nv\lib\site-packages (0.7.0)
Requirement already satisfied: joblib>=0.11 in c:\users\tense\anaconda3\envs\learn-env\l
ib\site-packages (from imbalanced-learn) (0.17.0)
Requirement already satisfied: scikit-learn>=0.23 in c:\users\tense\anaconda3\envs\learn
-env\lib\site-packages (from imbalanced-learn) (0.23.2)
Requirement already satisfied: scipy>=0.19.1 in c:\users\tense\anaconda3\envs\learn-env
\lib\site-packages (from imbalanced-learn) (1.5.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\tense\anaconda3\envs\learn-env
\lib\site-packages (from imbalanced-learn) (1.18.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\tense\anaconda3\envs\lea
rn-env\lib\site-packages (from scikit-learn>=0.23->imbalanced-learn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

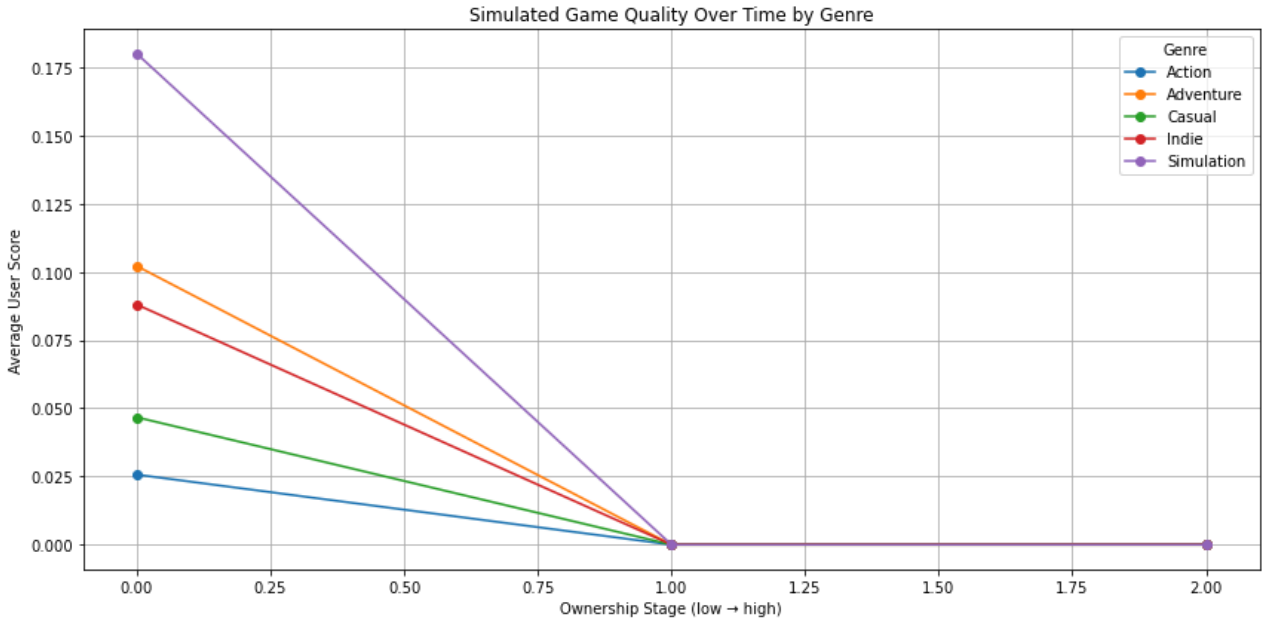
In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import SMOTE


df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste
df["High_quality_flag"] = (df["User score"] >= 30).astype(int)

features = [
    "Estimated owners", "Required age", "Positive", "Negative", "Achievements",
    "Average playtime forever", "Average playtime two weeks",
    "Median playtime forever", "Median playtime two weeks", "Windows"
]
df = df[features + ["High_quality_flag"]].dropna()

X = df[features]
```

```
y = df["High_quality_flag"]

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

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

model = GradientBoostingClassifier(random_state=42)
model.fit(X_resampled, y_resampled)

y_pred_resampled = model.predict(X_resampled)
print(classification_report(y_resampled, y_pred_resampled))
```

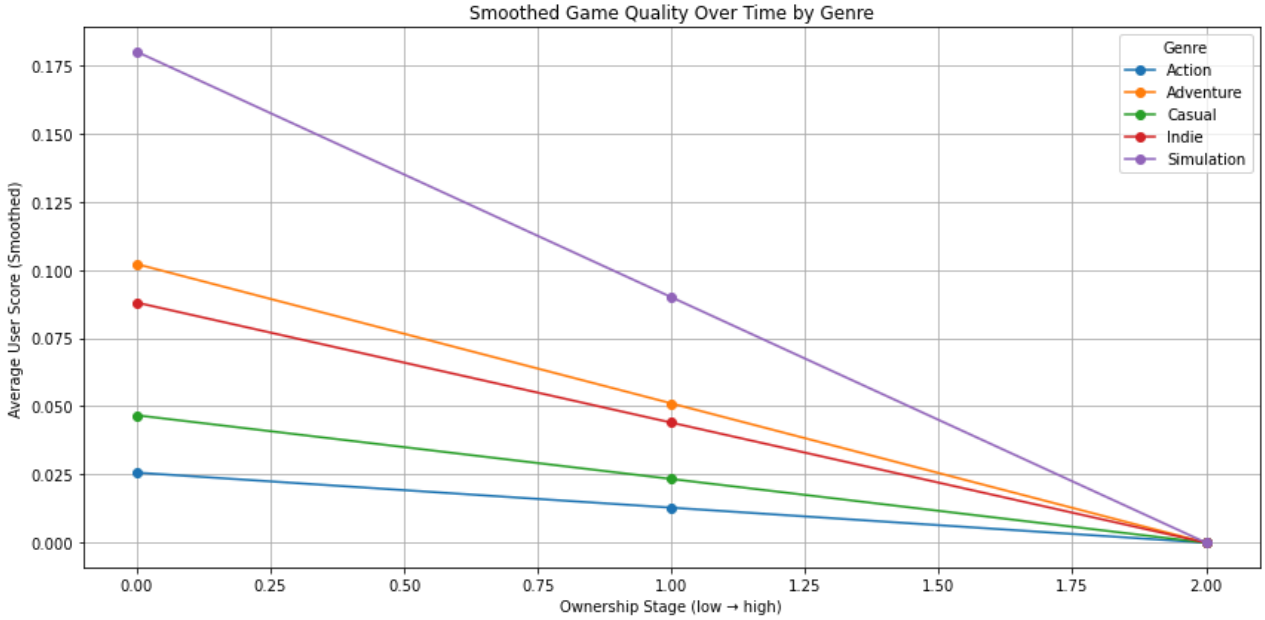
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      7994
           1       0.98      1.00      0.99      7994

    accuracy                           0.99     15988
   macro avg       0.99      0.99      0.99     15988
weighted avg       0.99      0.99      0.99     15988
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")


df = df[["estimated_owners", "user_score", "genres"]].dropna()


df["genres"] = df["genres"].str.split(",")
df = df.explode("genres")
df["genres"] = df["genres"].str.strip()


top_genres = df["genres"].value_counts().head(5).index
df = df[df["genres"].isin(top_genres)]
df["ownership_stage"] = pd.qcut(df["estimated_owners"], q=10, labels=False, duplicates=


grouped = df.groupby(["genres", "ownership_stage"])["user_score"].mean().reset_index()
pivot_df = grouped.pivot(index="ownership_stage", columns="genres", values="user_score"
plt.figure(figsize=(12, 6))
for genre in pivot_df.columns:
    plt.plot(pivot_df.index, pivot_df[genre], marker="o", label=genre)

plt.title("Simulated Game Quality Over Time by Genre")
plt.xlabel("Ownership Stage (low → high)")
plt.ylabel("Average User Score")
plt.legend(title="Genre")
plt.grid(True)
plt.tight_layout()
plt.show()
```



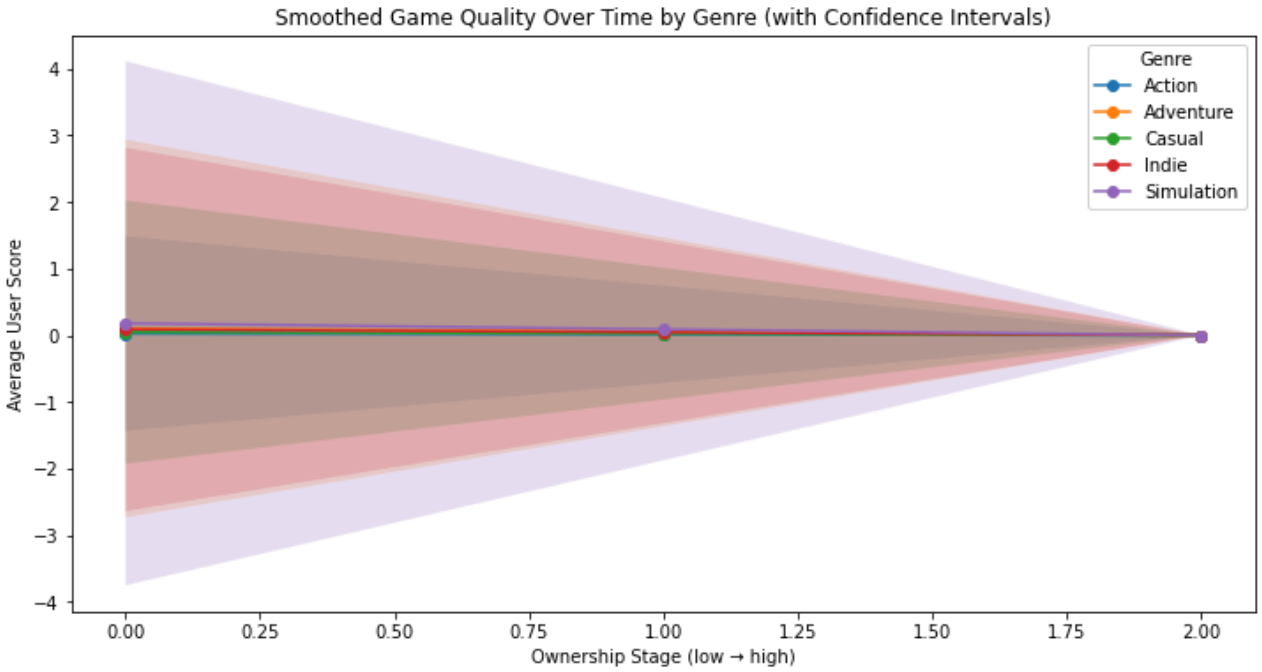
In [ ]:
```
plt.figure(figsize=(12, 6))

for genre in pivot_df.columns:
    smoothed = pivot_df[genre].rolling(window=2, min_periods=1).mean()  # Rolling avera
    plt.plot(smoothed.index, smoothed, marker="o", label=genre)

plt.title("Smoothed Game Quality Over Time by Genre")
plt.xlabel("Ownership Stage (low → high)")
plt.ylabel("Average User Score (Smoothed)")
plt.legend(title="Genre")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [ ]:
```
grouped_stats = df.groupby(["genres", "ownership_stage"])["user_score"].agg(['mean', 's

mean_pivot = grouped_stats.pivot(index="ownership_stage", columns="genres", values="mea
std_pivot = grouped_stats.pivot(index="ownership_stage", columns="genres", values="std"
```

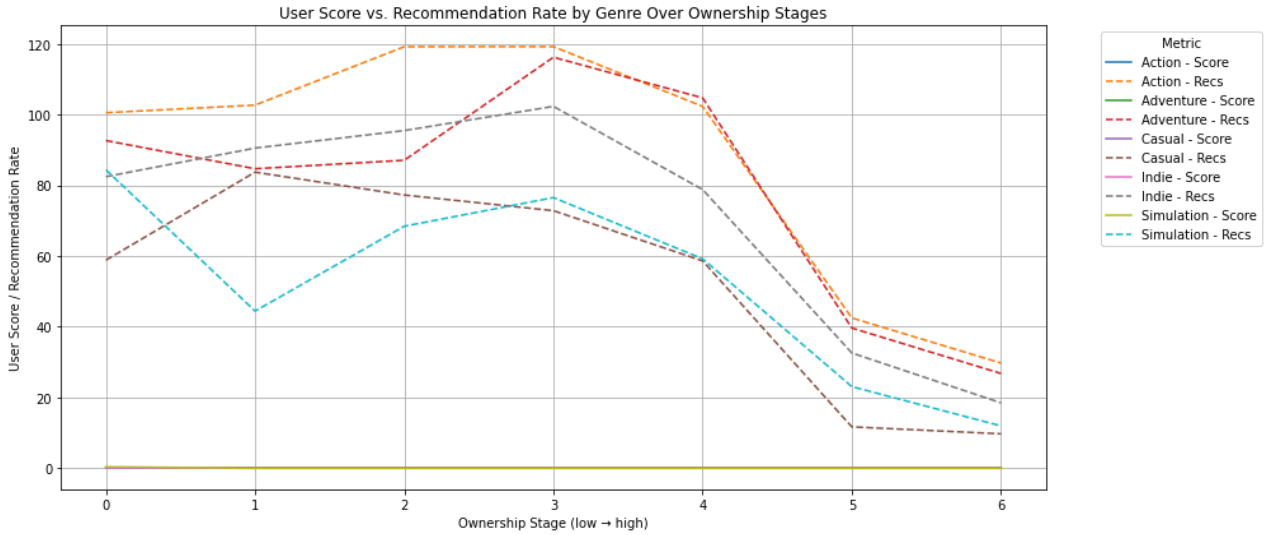
```
plt.figure(figsize=(12, 6))

for genre in mean_pivot.columns:
    smoothed_mean = mean_pivot[genre].rolling(window=2, min_periods=1).mean()
    smoothed_std = std_pivot[genre].rolling(window=2, min_periods=1).mean()


    plt.plot(smoothed_mean.index, smoothed_mean, label=genre, marker='o')
    plt.fill_between(
        smoothed_mean.index,
        smoothed_mean - smoothed_std,
        smoothed_mean + smoothed_std,
        alpha=0.2
    )

plt.title("Smoothed Game Quality Over Time by Genre (with Confidence Intervals)")
plt.xlabel("Ownership Stage (low → high)")
plt.ylabel("Average User Score")
plt.legend(title="Genre")
```

Out[ ]: <matplotlib.legend.Legend at 0x231a4978ee0>



In [ ]:
```
df = pd.read_csv(r"C:\Users\Tense\Documents\Flatiron\Data sets\Gaming Profiles 2025 Ste
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")
df = df[["estimated_owners", "user_score", "recommendations", "genres"]].dropna()
df["genres"] = df["genres"].str.split(",")
df = df.explode("genres")
df["genres"] = df["genres"].str.strip()


top_genres = df["genres"].value_counts().head(5).index
df = df[df["genres"].isin(top_genres)]


df = df[df["estimated_owners"] > 0]
df["recommendation_rate"] = df["recommendations"] / df["estimated_owners"]
df["ownership_stage"] = pd.qcut(df["estimated_owners"], q=10, labels=False, duplicates=
```

```
grouped = df.groupby(["genres", "ownership_stage"]).agg({
    "user_score": "mean",
    "recommendation_rate": "mean"
}).reset_index()


score_pivot = grouped.pivot(index="ownership_stage", columns="genres", values="user_sco
rec_pivot = grouped.pivot(index="ownership_stage", columns="genres", values="recommenda


plt.figure(figsize=(14, 6))

for genre in score_pivot.columns:
    plt.plot(score_pivot.index, score_pivot[genre], label=f"{genre} - Score", linestyle
    plt.plot(rec_pivot.index, rec_pivot[genre], label=f"{genre} - Recs", linestyle='--'

plt.title("User Score vs. Recommendation Rate by Genre Over Ownership Stages")
plt.xlabel("Ownership Stage (low → high)")
plt.ylabel("User Score / Recommendation Rate")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', title="Metric")
plt.grid(True)
plt.tight_layout()
plt.show()
```



As games become more mainstream, both perceived quality and enthusiasm to recommend decline.

Adventure Highly recommendable even as score dips — fun or compelling

Action Balanced, but both perception and hype fade

Casual “Okay” to play, not hype-worthy

Indie Early love, but struggles to scale