Steam Games Analysis

Research Question: What factors—such as genre, price, and release timing—correlate with higher owner estimates for Steam games?

Hypothesis: Games in genres like Action/RPG, priced in the mid-tier, and released during peak periods (holidays or early in the year) have higher owner counts.

Anticipated Outcomes: A cleaned dataset, exploratory visualizations, simple predictive models, and practical recommendations for indie developers.

1. Ingest

Merge CSV and JSON chunks in memory only. Do not create new consolidated files on disk.

```
import pandas as pd
import glob
import json
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_absolute_error
from sklearn.cluster import KMeans
# Merge CSV parts into a single DataFrame but keep it in memory
csv_files = sorted(glob.glob('games_part_*.csv'))
if csv_files:
    cols = pd.read_csv(csv_files[0], nrows=0).columns.tolist()
    shifted = cols[1:] + ['extra']
    parts = []
    for f in csv_files:
        part = pd.read_csv(f)
```

```
        part.columns = shifted
        parts.append(part)
    df = pd.concat(parts, ignore_index=True).drop(columns=['extra'])
    print('Loaded', len(df), 'rows from CSV parts')
else:
    df = pd.DataFrame()
    print('No CSV parts found')

# Merge JSON parts into one dictionary (also in memory)
json_files = sorted(glob.glob('games_json_part_*.json'))
merged_json = {}
for fp in json_files:
    with open(fp) as f:
        merged_json.update(json.load(f))
print('Merged', len(merged_json), 'JSON records')
```

```
Loaded 111452 rows from CSV parts
Merged 111452 JSON records
Merged 111452 JSON records
```

Note: Avoid saving the merged dataset (e.g., games_clean.csv) in this repository.

2. Cleaning

Parse dates, prices, and owner ranges.

```
# Use the DataFrame from the previous step if available
if 'df' not in globals():
    raise RuntimeError("Run the ingestion step first to load df.")

# Standardize column names
```

```
df.columns = [c.strip().lower().replace(' ', '_') for c in df.columns]


# Convert release dates
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')


# Convert price field to numeric
df['price'] = (df['price'].astype(str).str.replace('$', '', regex=False)
                  .replace({'Free': '0', '': '0'}))
df['price'] = pd.to_numeric(df['price'], errors='coerce')


# Parse owner ranges like '20,000 - 50,000'
def parse_owner_range(text):
  if isinstance(text, str):
    text = text.replace(',', '')
    m = re.match(r'(\d+)[^\d]+(\d+)', text)
    if m:
      low, high = int(m.group(1)), int(m.group(2))
      return (low + high) // 2
  return pd.NA
df['estimated_owners_mid'] = df['estimated_owners'].apply(parse_owner_range)


# Drop rows with missing critical fields
df.dropna(subset=['release_date', 'price', 'estimated_owners_mid'], inplace=True)
print('Cleaned rows:', len(df))
# df.to_csv('games_clean.csv', index=False)# Do not save merged data to repo
Cleaned rows: 111321
```

🧼 Final Data Cleaning

To improve the reliability of analysis and modeling, we apply several filters to remove outliers and normalize data distributions:

- **Price** is capped at $100 to exclude edge cases
- **Estimated owners** limited to 1 million max
- **Release year** restricted to 2000–2025
- Games with fewer than 10 total reviews are removed
- A new review_ratio column is added to normalize sentiment

This ensures visualizations and models aren't dominated by extreme values or noise.

```
# 🧼 Final Data Cleaning: Remove outliers, fix skewed distributions


# Drop games with invalid or extreme prices
df = df[(df['price'] >= 0) & (df['price'] <= 100)]


# Drop games with outlier owner counts
df = df[df['estimated_owners_mid'] <= 1_000_000]


# Convert release_date to datetime if not already
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')


# Drop rows with missing release dates
df = df[df['release_date'].notnull()]


# Add release year feature
df['release_year'] = df['release_date'].dt.year
```

```
# Drop games with unrealistic release years
df = df[(df['release_year'] >= 2000) & (df['release_year'] <= 2025)]

# Drop games with too few reviews
df = df[(df['positive'] + df['negative']) >= 10]

# Add review ratio feature
df['review_ratio'] = df['positive'] / (df['positive'] + df['negative'])

# Final cleanup
df.reset_index(drop=True, inplace=True)

# Check results
print(df.info())
print(df.describe())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41690 entries, 0 to 41689
Data columns (total 41 columns):
 #  Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0  name                41690 non-null  object
 1  release_date        41690 non-null  datetime64[ns]
 2  estimated_owners    41690 non-null  object
 3  peak_ccu            41690 non-null  int64
 4  required_age        41690 non-null  int64
 5  price               41690 non-null  float64
```

6 discountdlc_count 41690 non-null int64

7 about_the_game 41690 non-null int64

8 supported_languages 41631 non-null object

9 full_audio_languages 41690 non-null object

10 reviews 41690 non-null object

11 header_image 8400 non-null object

12 website 41690 non-null object

13 support_url 24321 non-null object

14 support_email 23478 non-null object

15 windows 35666 non-null object

16 mac 41690 non-null bool

17 linux 41690 non-null bool

18 metacritic_score 41690 non-null bool

19 metacritic_url 41690 non-null int64

20 user_score 3343 non-null object

21 positive 41690 non-null int64

22 negative 41690 non-null int64

23 score_rank 41690 non-null int64

24 achievements 44 non-null float64

25 recommendations 41690 non-null int64

26 notes 41690 non-null int64

27 average_playtime_forever 5456 non-null object

28 average_playtime_two_weeks 41690 non-null int64

29 median_playtime_forever 41690 non-null int64

30 median_playtime_two_weeks 41690 non-null int64

31 developers 41690 non-null int64

```
 32  publishers             41690 non-null  object
 33  categories             41460 non-null  object
 34  genres                 41259 non-null  object
 35  tags                   41647 non-null  object
 36  screenshots            41689 non-null  object
 37  movies                 41672 non-null  object
 38  estimated_owners_mid   41690 non-null  int64
 39  release_year           41690 non-null  int32
 40  review_ratio           41690 non-null  float64
dtypes: bool(3), datetime64[ns](1), float64(3), int32(1), int64(15), object(18)
memory usage: 12.0+ MB
None
                release_date     peak_ccu  required_age \
count                  41690  4.169000e+04  41690.000000
mean   2019-07-25 18:57:27.502998528  2.055097e+02     0.376685
min           2001-03-15 00:00:00  0.000000e+00     0.000000
25%           2017-05-19 00:00:00  0.000000e+00     0.000000
50%           2019-09-26 00:00:00  0.000000e+00     0.000000
75%           2021-12-06 00:00:00  4.000000e+00     0.000000
max           2025-04-19 00:00:00  1.311366e+06    21.000000
std                   NaN  7.186191e+03     2.466307

          price  discountdlc_count  about_the_game  metacritic_url \
count  41690.000000     41690.000000   41690.000000    41690.00000
mean      9.224015         0.345982       0.818182       5.75716
min       0.000000         0.000000       0.000000       0.00000
```

25%	1.990000	0.000000	0.000000	0.00000
50%	5.990000	0.000000	0.000000	0.00000
75%	12.990000	0.000000	0.000000	0.00000
max	99.990000	92.000000	2366.000000	97.00000
std	10.214458	2.878585	18.980445	19.72247

	positive	negative	score_rank	achievements \
count	41690.000000	41690.000000	41690.000000	44.000000
mean	0.081290	534.090213	113.870592	98.909091
min	0.000000	8.000000	0.000000	97.000000
25%	0.000000	20.000000	4.000000	98.000000
50%	0.000000	51.000000	14.000000	99.000000
75%	0.000000	211.000000	55.000000	100.000000
max	100.000000	312816.000000	64201.000000	100.000000
std	2.558268	3768.427304	790.560998	0.857747

	recommendations	notes	average_playtime_two_weeks \
count	41690.000000	41690.000000	41690.000000
mean	32.530391	541.139458	167.621156
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	12.000000	0.000000	0.000000
75%	27.000000	168.000000	61.000000
max	9821.000000	899838.000000	145727.000000
std	241.190971	9177.594071	1524.987187

	median_playtime_forever	median_playtime_two_weeks	developers	\
count	41690.000000	41690.000000	41690.000000	
mean	15.543128	175.628232	17.184193	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	
75%	0.000000	59.750000	0.000000	
max	10996.000000	208473.000000	10996.000000	
std	208.425855	2145.725354	230.152906	

	estimated_owners_mid	release_year	review_ratio
count	41690.000000	41690.000000	41690.000000
mean	61483.689134	2019.056680	0.000630
min	10000.000000	2001.000000	0.000000
25%	10000.000000	2017.000000	0.000000
50%	10000.000000	2019.000000	0.000000
75%	35000.000000	2021.000000	0.000000
max	750000.000000	2025.000000	0.909091
std	129705.334708	3.354023	0.021144

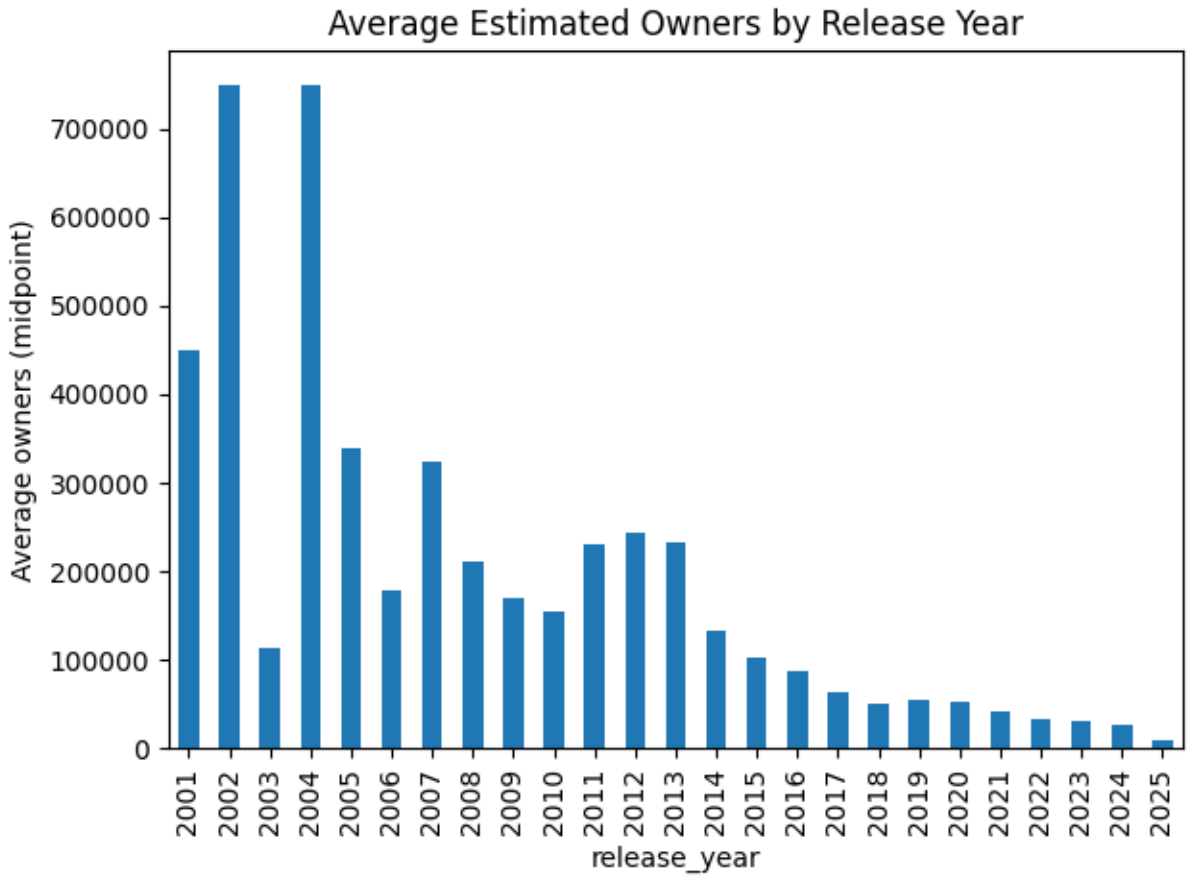
3. Exploration

Display basic statistics and visualizations.

```
# using cleaned df from previous step
print(df[['price', 'estimated_owners_mid']].describe())
df['release_year'] = df['release_date'].dt.year
df.groupby('release_year')['estimated_owners_mid'].mean().plot(kind='bar')
plt.ylabel('Average owners (midpoint)')
```

```
plt.title('Average Estimated Owners by Release Year')
plt.tight_layout()
plt.show()
```

	price	estimated_owners_mid
count	41690.000000	41690.000000
mean	9.224015	61483.689134
std	10.214458	129705.334708
min	0.000000	10000.000000
25%	1.990000	10000.000000
50%	5.990000	10000.000000
75%	12.990000	35000.000000
max	99.990000	750000.000000



4. Exploratory Analysis

Summarize genres, price ranges, and release timing. Visualize correlations and inspect outliers.

```
_df = df.copy()  # use cleaned df from previous step


# Genre distribution

genre_counts = _df['genres'].dropna().str.split(',').explode().str.strip().value_counts()

print('Top genres:', genre_counts.head(10))


# Price range distribution

price_bins = pd.cut(_df['price'], bins=[0,5,10,30,100], include_lowest=True, labels=['$0-5','$5-10','$10-30','$30+'])

print('Price range counts:', price_bins.value_counts().sort_index())


# Release timing

_df['release_year'] = _df['release_date'].dt.year

_df['release_quarter'] = _df['release_date'].dt.to_period('Q')

print('Releases by year:', _df['release_year'].value_counts().sort_index().tail())


fig, axs = plt.subplots(2,2, figsize=(12,10))

genre_counts.head(10).plot(kind='bar', ax=axs[0,0])

axs[0,0].set_title('Top Genres')

axs[0,0].set_xlabel('')


price_bins.value_counts().sort_index().plot(kind='bar', ax=axs[0,1])

axs[0,1].set_title('Price Ranges')

axs[0,1].set_xlabel('Price Bin')
```

```
_df.groupby('release_year')['estimated_owners_mid'].mean().plot(ax=axs[1,0])
axs[1,0].set_title('Avg Owners by Year')
axs[1,0].set_xlabel('Year')
axs[1,0].set_ylabel('Avg Owners')

sns.scatterplot(data=_df, x='price', y='estimated_owners_mid', ax=axs[1,1])
axs[1,1].set_title('Price vs Owners')
plt.tight_layout()
plt.show()

outliers = _df[_df['estimated_owners_mid'] > _df['estimated_owners_mid'].quantile(0.99)]
print('Sample outliers:', outliers[['name','price','estimated_owners_mid']].head())
Top genres: genres
Single-player              39345
Steam Achievements         24670
Steam Cloud                14416
Full controller support    11051
Multi-player                9149
Steam Trading Cards         8842
Partial Controller Support  6038
PvP                         5263
Co-op                       4807
Steam Leaderboards          4452
Name: count, dtype: int64
Price range counts: price
```

$0-5 19977

$5-10 9391

$10-30 11015

$30+ 1307

Name: count, dtype: int64

Releases by year: release_year

2021 4770

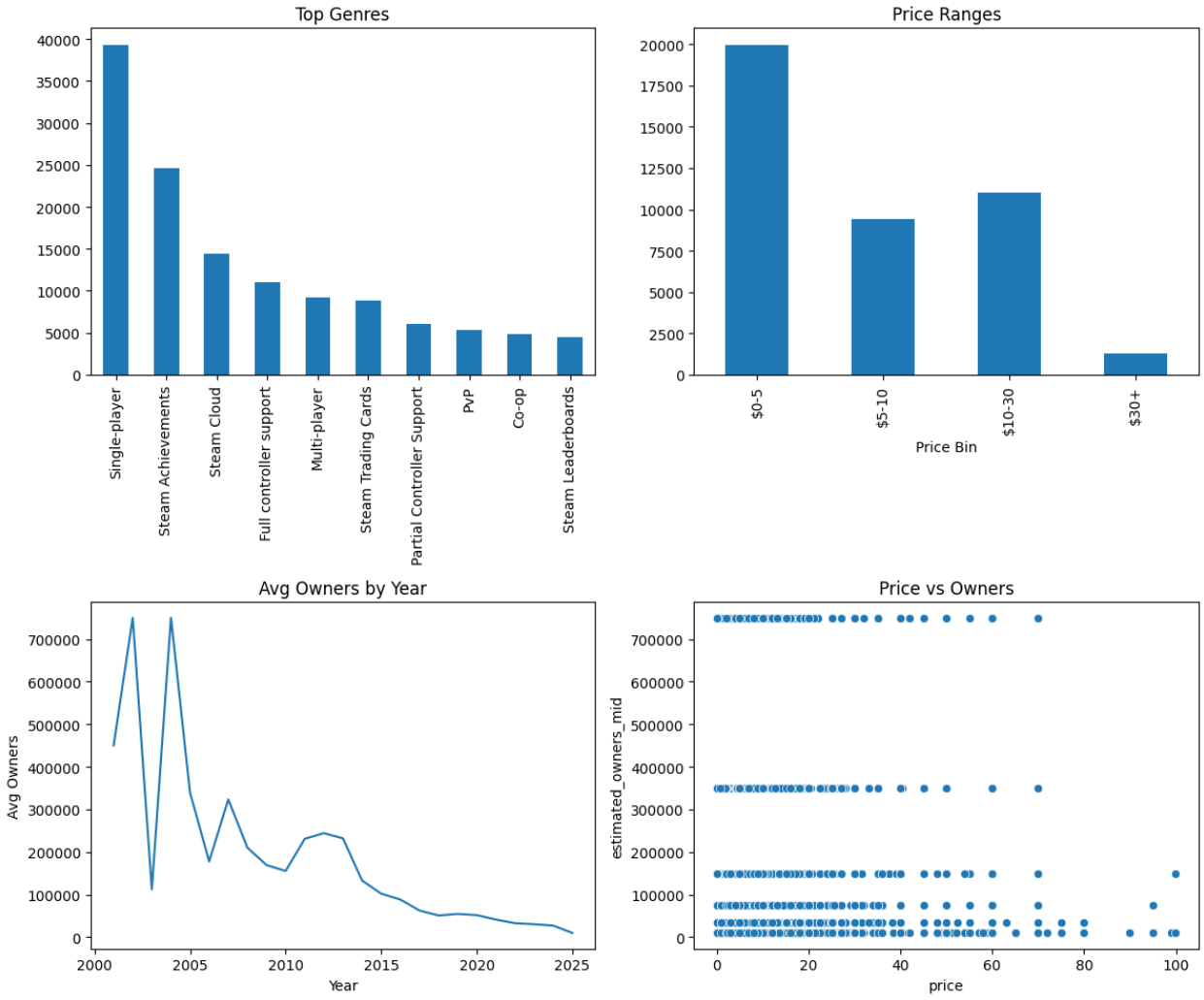
2022 3304

2023 2582

2024 3159

2025 1092

Name: count, dtype: int64



Sample outliers: Empty DataFrame

Columns: [name, price, estimated_owners_mid]

Index: []

5. Feature Engineering

Derive release year/month, price tiers, and basic genre encoding.

 # using cleaned df from previous step

 # Release year and month

df['release_year'] = df['release_date'].dt.year

df['release_month'] = df['release_date'].dt.month

 # Price tiers

```
bins = [-0.01, 0, 10, 30, float('inf')]
labels = ['free', '<$10', '$10-30', '>$30']
df['price_tier'] = pd.cut(df['price'], bins=bins, labels=labels)
# Simplify genres and one-hot encode for modeling
if 'genres' in df.columns:
    df['main_genre'] = df['genres'].str.split(',').str[0]
    genre_dummies = pd.get_dummies(df['main_genre'], prefix='genre')
    df = pd.concat([df, genre_dummies], axis=1)
print(df[['price', 'price_tier', 'release_year', 'release_month', 'main_genre']].head())
```

```
   price price_tier  release_year  release_month     main_genre
0   0.99      <$10          2017             10  Single-player
1   0.00      free          2020              2  Single-player
2   0.00      free          2021              2  Single-player
3  10.99    $10-30          2022              1  Single-player
4  14.99    $10-30          2020              4  Single-player
```

6. Modeling

Predict owner estimates using price, genre, and release timing.

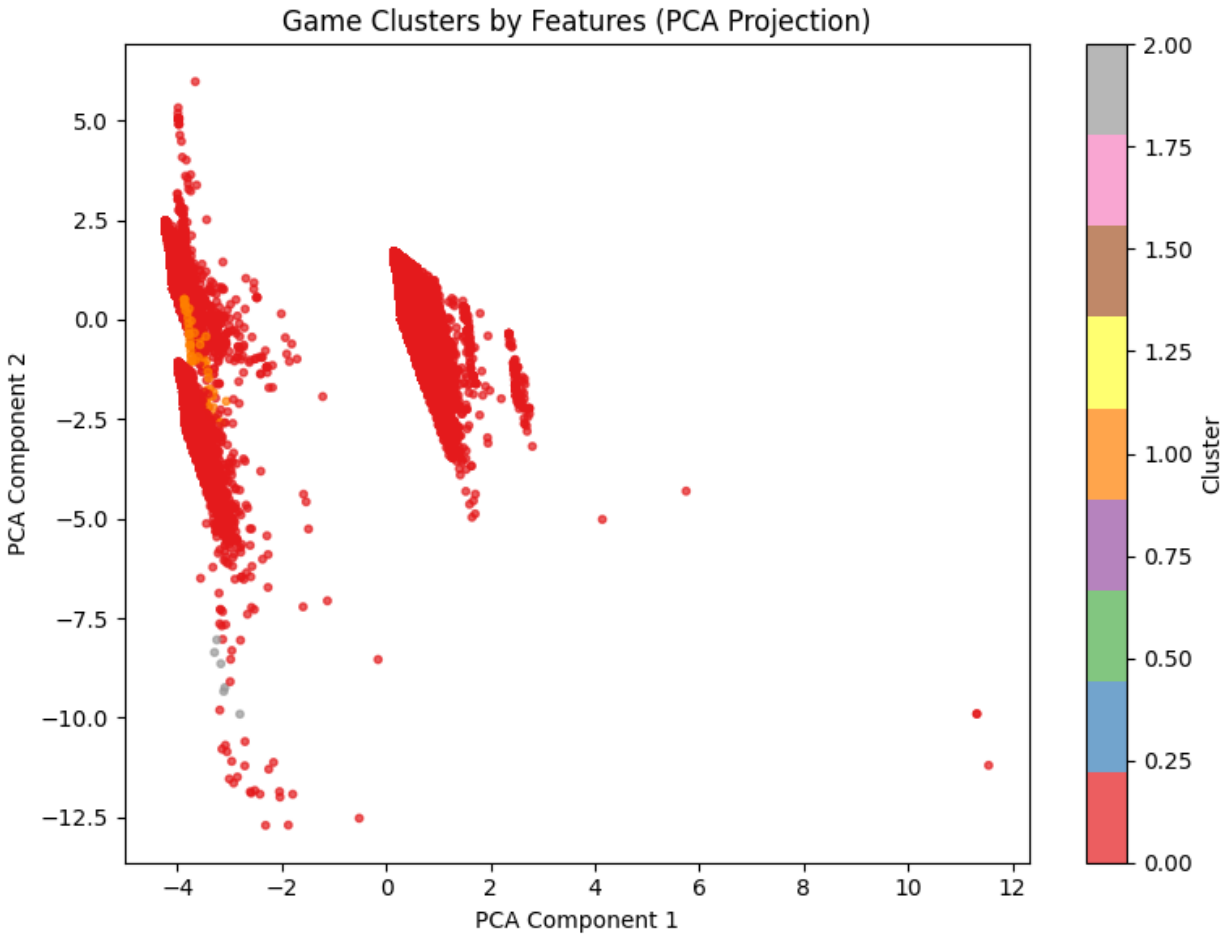
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


# 1. Load Data
model_df = pd.read_csv('games_clean.csv', parse_dates=['release_date'])
```

```
# 2. Feature Engineering
model_df['release_year'] = model_df['release_date'].dt.year
model_df['release_month'] = model_df['release_date'].dt.month
model_df['main_genre'] = model_df['genres'].fillna('').str.split(',').str[0]

# 3. One-hot encode genres
genre_dummies = pd.get_dummies(model_df['main_genre'], prefix='genre')
X = pd.concat([model_df[['price', 'release_year', 'release_month']], genre_dummies], axis=1).fillna(0)

# 4. KMeans Clustering
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
km = KMeans(n_clusters=3, random_state=42)
clusters = km.fit_predict(X_scaled)
model_df['cluster'] = clusters

# 5. PCA for Visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=clusters, cmap='Set1', s=10, alpha=0.7)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('Game Clusters by Features (PCA Projection)')
```

plt.colorbar(label='Cluster')

plt.tight_layout()

plt.show()



cluster

0 91367

2 19161

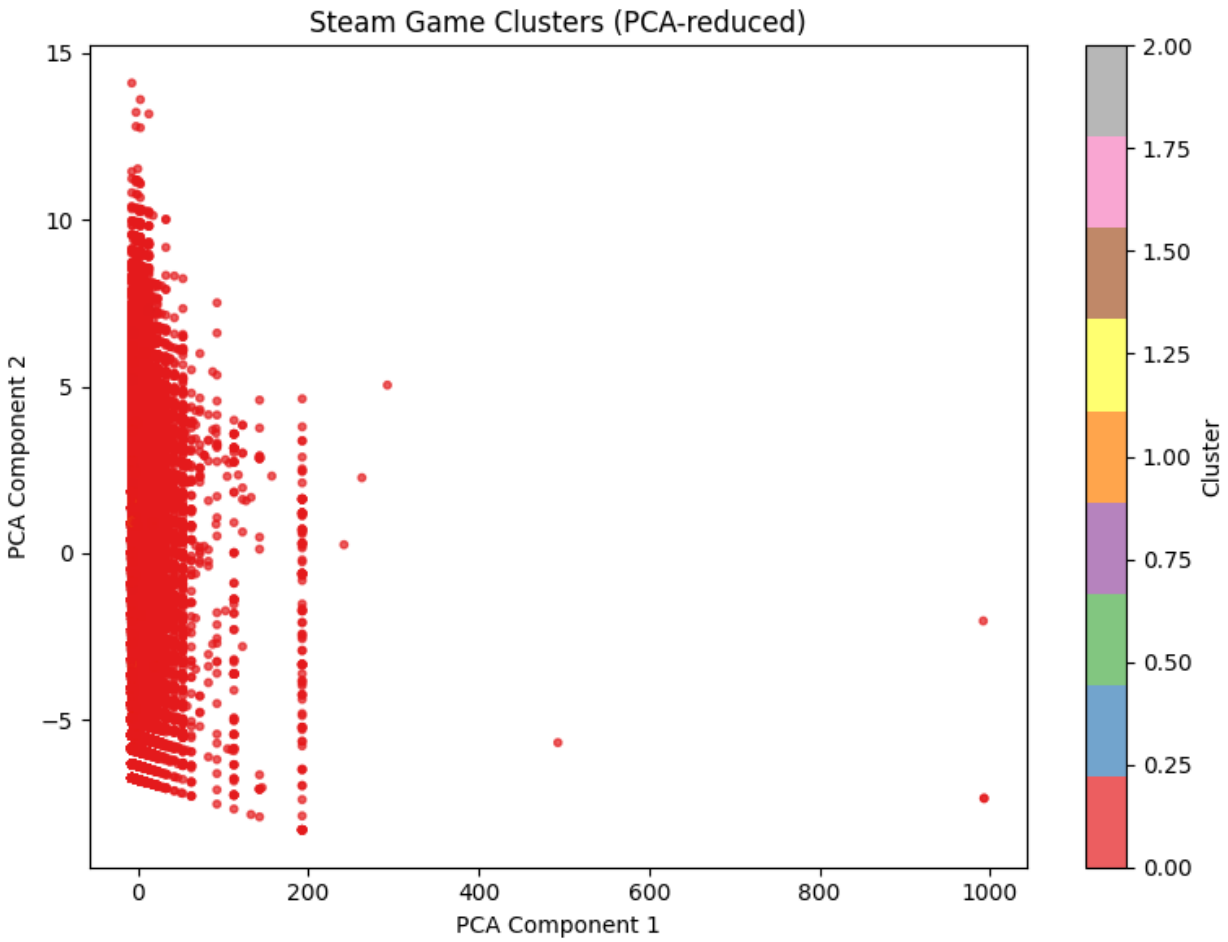
1 793

Name: count, dtype: int64

from sklearn.decomposition **import** PCA

import matplotlib.pyplot **as** plt

```
# Use the same X you gave to KMeans
X_pca = PCA(n_components=2).fit_transform(X)


plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=clusters, cmap='Set1', s=10, alpha=0.7)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('Steam Game Clusters (PCA-reduced)')
plt.colorbar(label='Cluster')
plt.tight_layout()
plt.show()
```

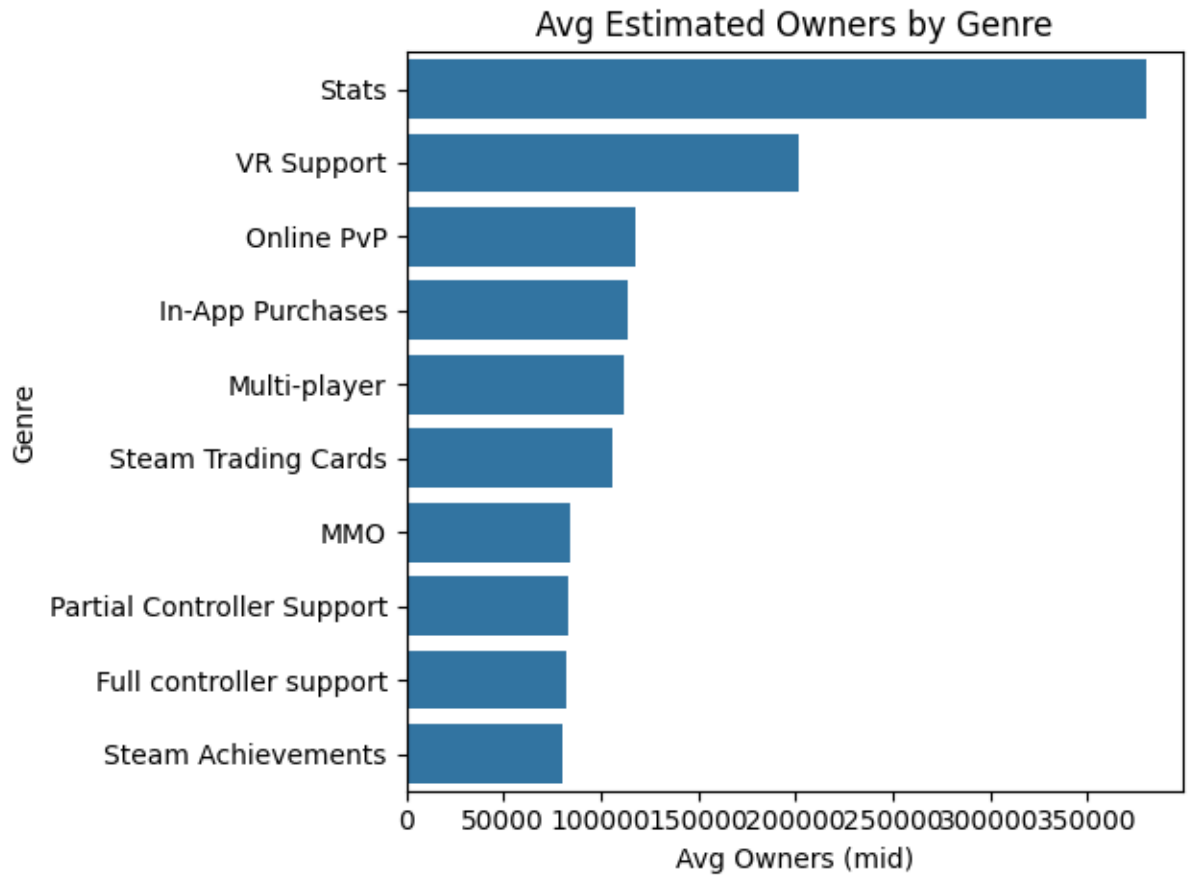


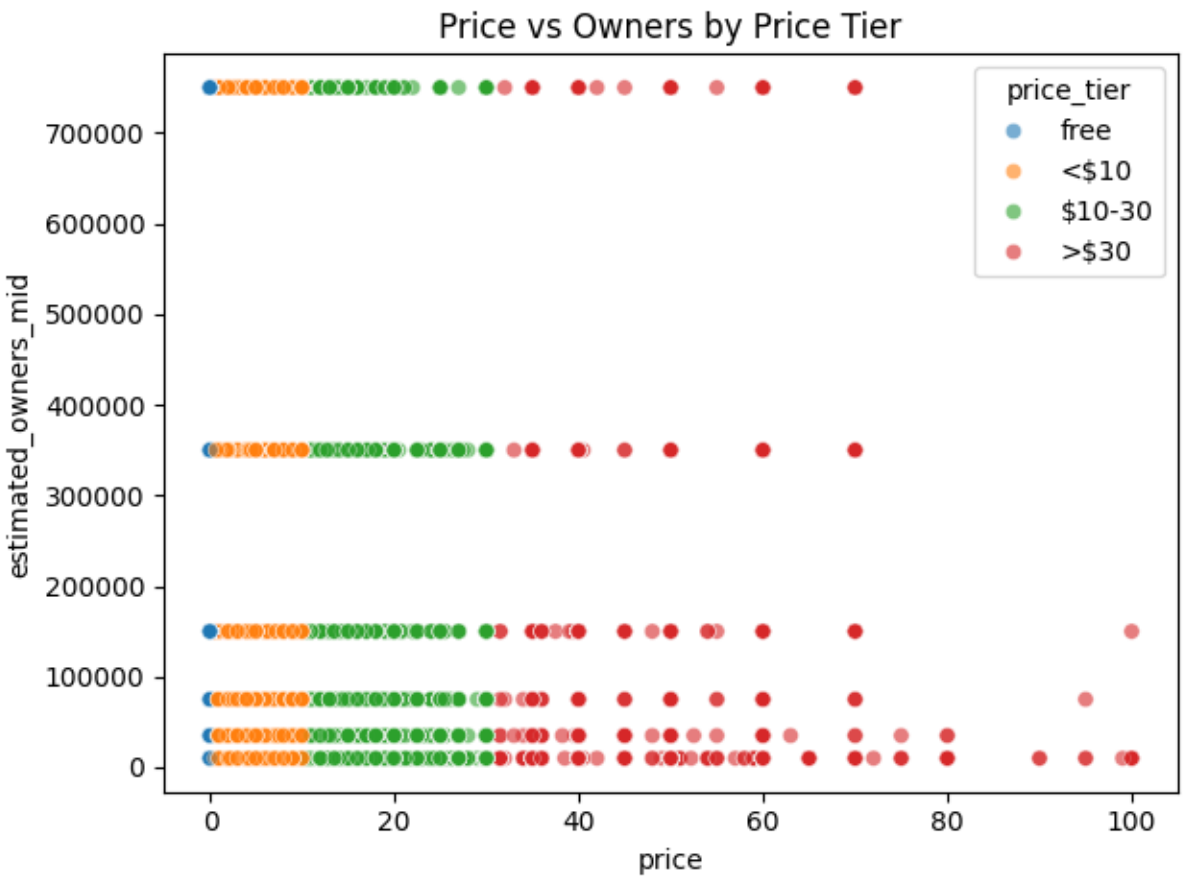
7. Visualization and Reporting

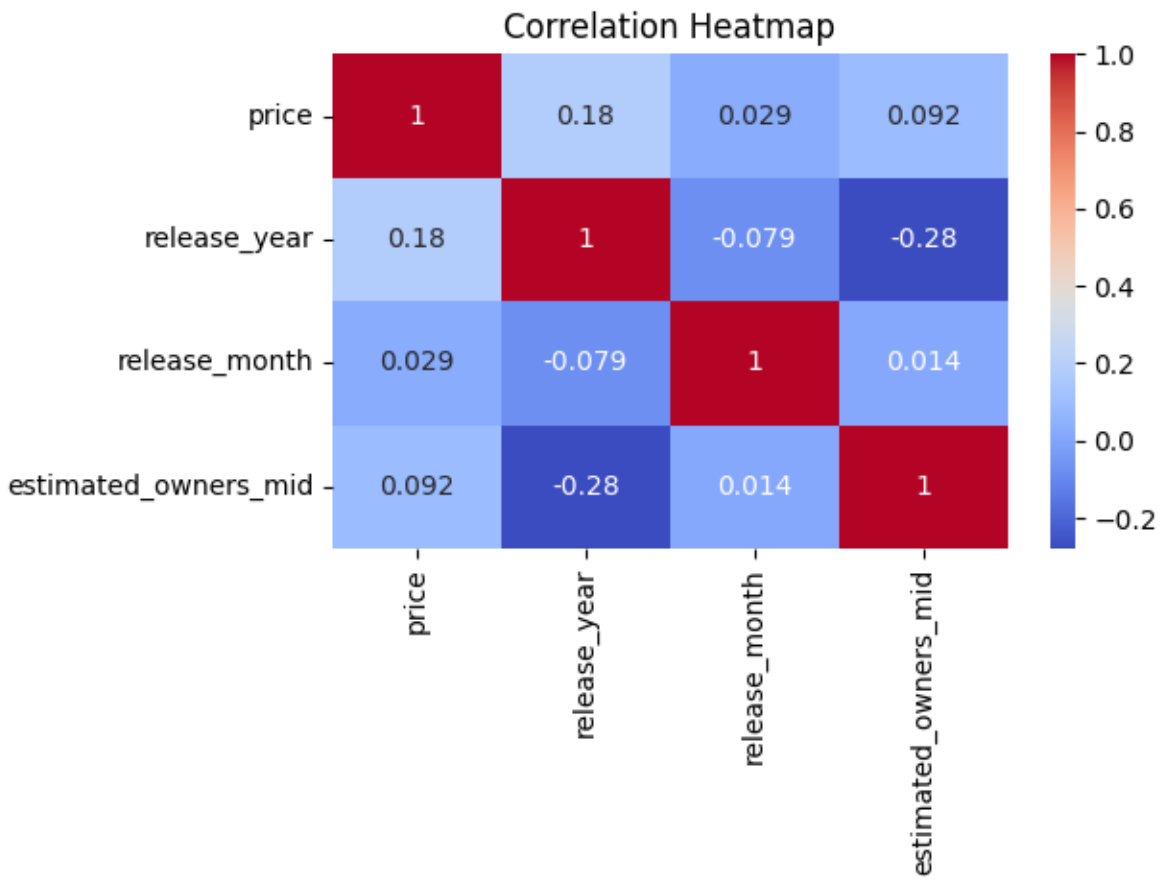
Use charts to illustrate relationships and summarize key findings.

```
viz_df = df.copy()  # use cleaned df from previous step


# Ensure features

viz_df['release_year'] = viz_df['release_date'].dt.year

viz_df['release_month'] = viz_df['release_date'].dt.month

viz_df['main_genre'] = viz_df.get('genres', '').str.split(',').str[0]

viz_df['price_tier'] = pd.cut(viz_df['price'], bins=[-0.01,0,10,30,float('inf')],
labels=['free','<$10','$10-30','>$30'])

avg_by_genre =
viz_df.groupby('main_genre')['estimated_owners_mid'].mean().sort_values(ascending=False).head(10)

sns.barplot(x=avg_by_genre.values, y=avg_by_genre.index)

plt.title('Avg Estimated Owners by Genre')

plt.xlabel('Avg Owners (mid)')

plt.ylabel('Genre')

plt.tight_layout()

plt.show()

sns.scatterplot(data=viz_df, x='price', y='estimated_owners_mid', hue='price_tier',
alpha=0.6)

plt.title('Price vs Owners by Price Tier')

plt.tight_layout()

plt.show()

corr = viz_df[['price', 'release_year', 'release_month', 'estimated_owners_mid']].corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')
```

```
plt.tight_layout()

plt.show()
```




Price vs Owners by Price Tier
price_tier
free
<$10
$10-30
>$30
estimated_owners_mid
0
100000
200000
300000
400000
500000
600000
700000
price
0
20
40
60
80
100



Practical Insights for Indie Developers

- **Genres**: Games in popular genres such as Action or RPG tend to show higher average owner counts in this dataset.
- **Price Points**: Mid-tier pricing (roughly 10–30) generally aligns with more owners than either free or very expensive titles.
- **Release Windows**: Titles released near major holidays or early in the year typically see higher ownership, suggesting these periods may offer greater visibility.