AD654 - Group 3, Team B - Semester Project

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

In [ ]:
```
df_disney_movies = pd.read_csv ('/Users/hrishitagoyal/Desktop/AD654/disney_movies.csv'
```

In [ ]:
```
df_disney_movies
```

Out[ ]:

	movie_title	release_date	genre	mpaa_rating	total_gross	inflation_adjusted_gross
0	Snow White and the Seven Dwarfs	1937-12-21	Musical	G	184925485	5228953251
1	Pinocchio	1940-02-09	Adventure	G	84300000	2188229052
2	Fantasia	1940-11-13	Musical	G	83320000	2187090808
3	Song of the South	1946-11-12	Adventure	G	65000000	1078510579
4	Cinderella	1950-02-15	Drama	G	85000000	920608730
...	...	...	...	...	...	...
574	The Light Between Oceans	2016-09-02	Drama	PG-13	12545979	12545979
575	Queen of Katwe	2016-09-23	Drama	PG	8874389	8874389
576	Doctor Strange	2016-11-04	Adventure	PG-13	232532923	232532923
577	Moana	2016-11-23	Adventure	PG	246082029	246082029
578	Rogue One: A Star Wars Story	2016-12-16	Adventure	PG-13	529483936	529483936

579 rows × 6 columns

In [ ]:
```
#Data Overview:
df_disney_movies.head()
```

Out[ ]:

	movie_title	release_date	genre	mpaa_rating	total_gross	inflation_adjusted_gross
0	Snow White and the Seven Dwarfs	1937-12-21	Musical	G	184925485	5228953251
1	Pinocchio	1940-02-09	Adventure	G	84300000	2188229052
2	Fantasia	1940-11-13	Musical	G	83320000	2187090808
3	Song of the South	1946-11-12	Adventure	G	65000000	1078510579
4	Cinderella	1950-02-15	Drama	G	85000000	920608730

In [ ]:
```
#Data Overview - structure
df_disney_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 579 entries, 0 to 578
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   movie_title              579 non-null    object
 1   release_date             579 non-null    object
 2   genre                    562 non-null    object
 3   mpaa_rating              523 non-null    object
 4   total_gross              579 non-null    int64 
 5   inflation_adjusted_gross  579 non-null    int64 
dtypes: int64(2), object(4)
memory usage: 27.3+ KB
```

In [ ]:
```
#Descriptive stats of the dataset
df_disney_movies.describe()
```

Out[ ]:

	total_gross	inflation_adjusted_gross
count	5.790000e+02	5.790000e+02
mean	6.470179e+07	1.187625e+08
std	9.301301e+07	2.860853e+08
min	0.000000e+00	0.000000e+00
25%	1.278886e+07	2.274123e+07
50%	3.070245e+07	5.515978e+07
75%	7.570903e+07	1.192020e+08
max	9.366622e+08	5.228953e+09

In [ ]:
```
#Distribution of categorial values
df_disney_movies[['genre', 'mpaa_rating']].value_counts()
```

Out[ ]:

genre	mpaa_rating	
Comedy	PG	77
Adventure	PG	57
	G	42
Drama	PG-13	37
Comedy	PG-13	37
Drama	R	33
Comedy	R	30
Drama	PG	28
Action	PG-13	19
Comedy	G	17
Adventure	PG-13	17
Thriller/Suspense	PG-13	13
Action	R	12
Romantic Comedy	PG-13	12
Documentary	G	10
Thriller/Suspense	R	9
Musical	G	9
Romantic Comedy	PG	7
Action	PG	5
Drama	G	5
Documentary	PG	5
Western	PG-13	4
Horror	R	4
Musical	PG	4
Adventure	R	3
Black Comedy	R	3
Romantic Comedy	R	2
Concert/Performance	G	2
Western	R	2
Romantic Comedy	G	1
Comedy	Not Rated	1
Musical	PG-13	1
Thriller/Suspense	PG	1
Musical	Not Rated	1
Documentary	Not Rated	1
Western	PG	1
Horror	PG-13	1

dtype: int64

In [ ]:
```
#Display sum of Null values for each column
df_disney_movies.isnull().sum()
```

Out[ ]:
```
movie_title                0
release_date               0
genre                     17
mpaa_rating               56
total_gross                0
inflation_adjusted_gross   0
dtype: int64
```

In [ ]:
```
#Drop NaN values
df_disney_nona = df_disney_movies.dropna()
```

In [ ]:
```
#Verifying
df_disney_nona.isnull().sum()
```

Out[ ]:
```
movie_title                0
release_date               0
genre                      0
mpaa_rating                0
total_gross                0
inflation_adjusted_gross   0
dtype: int64
```

In [ ]:
```
#Removing duplicates
df_disney_nona.drop_duplicates()
```

Out[ ]:

	movie_title	release_date	genre	mpaa_rating	total_gross	inflation_adjusted_gross
0	Snow White and the Seven Dwarfs	1937-12-21	Musical	G	184925485	5228953251
1	Pinocchio	1940-02-09	Adventure	G	84300000	2188229052
2	Fantasia	1940-11-13	Musical	G	83320000	2187090808
3	Song of the South	1946-11-12	Adventure	G	65000000	1078510579
4	Cinderella	1950-02-15	Drama	G	85000000	920608730
...	...	...	...	...	...	...
574	The Light Between Oceans	2016-09-02	Drama	PG-13	12545979	12545979
575	Queen of Katwe	2016-09-23	Drama	PG	8874389	8874389
576	Doctor Strange	2016-11-04	Adventure	PG-13	232532923	232532923
577	Moana	2016-11-23	Adventure	PG	246082029	246082029
578	Rogue One: A Star Wars Story	2016-12-16	Adventure	PG-13	529483936	529483936

513 rows × 6 columns

In [ ]:
```
# Grouping and Aggregation by MPAA Rating
mpaa_rating_group = df_disney_nona.groupby('mpaa_rating').agg({
    'inflation_adjusted_gross': ['mean', 'median', 'sum'],
    'total_gross': ['mean', 'median', 'sum']
})

# Grouping and Aggregation by Genre
genre_group = df_disney_nona.groupby('genre').agg({
    'inflation_adjusted_gross': ['mean', 'median', 'sum'],
    'total_gross': ['mean', 'median', 'sum']
})

# Grouping and Aggregation by Year
df_disney_nona['release_year'] = pd.to_datetime(df_disney_nona['release_date']).dt.yea
year_group = df_disney_nona.groupby('release_year').agg({
    'inflation_adjusted_gross': ['mean', 'median', 'sum'],
    'total_gross': ['mean', 'median', 'sum']
})

# Grouping and Aggregation by Genre and MPAA Rating
genre_rating_group = df_disney_nona.groupby(['genre', 'mpaa_rating']).agg({
    'inflation_adjusted_gross': ['mean', 'median', 'sum'],
```

```
    'total_gross': ['mean', 'median', 'sum']
})

# Display the results
print("Grouping by MPAA Rating:")
print(mpaa_rating_group.head())

print("\nGrouping by Genre:")
print(genre_group.head())

print("\nGrouping by Year:")
print(year_group.head())

print("\nGrouping by Genre and MPAA Rating:")
print(genre_rating_group.head())
```

Grouping by MPAA Rating:

	inflation_adjusted_gross			total_gross	
	mean	median	sum	mean	median
mpaa_rating					
G	2.912610e+08	103154765.5	25048445571	9.209061e+07	65140890.5
Not Rated	2.998734e+08	109581646.0	899620238	5.046259e+07	9230769.0
PG	1.026074e+08	69540672.0	18982370383	7.440529e+07	50236831.0
PG-13	1.056656e+08	57544453.0	14898852206	8.338368e+07	37036404.0
R	5.741288e+07	35482801.5	5626462048	3.048986e+07	19568958.5

	total_gross
	sum
mpaa_rating	
G	7919792693
Not Rated	151387769
PG	13764978891
PG-13	11757098250
R	2988006302

Grouping by Genre:

	inflation_adjusted_gross		
	mean	median	sum
genre			
Action	1.486012e+08	80824686.0	5349644857
Adventure	1.971801e+08	109310377.0	23464429224
Black Comedy	5.224349e+07	51579764.0	156730475
Comedy	8.731064e+07	52864513.5	14144323456
Concert/Performance	5.741084e+07	57410839.0	114821678

	total_gross		
	mean	median	sum
genre			
Action	1.145136e+08	50792379.5	4122489314
Adventure	1.359535e+08	81612565.0	16178462965
Black Comedy	3.251440e+07	28084357.0	97543212
Comedy	4.808848e+07	29952419.0	7790333078
Concert/Performance	5.172823e+07	51728233.0	103456466

Grouping by Year:

	inflation_adjusted_gross			total_gross	
	mean	median	sum	mean	median
release_year					
1937	5.228953e+09	5.228953e+09	5228953251	184925485.0	184925485.0
1940	2.187660e+09	2.187660e+09	4375319860	83810000.0	83810000.0
1946	1.078511e+09	1.078511e+09	1078510579	65000000.0	65000000.0
1950	9.206087e+08	9.206087e+08	920608730	85000000.0	85000000.0
1955	1.236036e+09	1.236036e+09	1236035515	93600000.0	93600000.0

	total_gross	
	sum	
release_year		
1937	184925485	
1940	167620000	
1946	65000000	
1950	85000000	
1955	93600000	

Grouping by Genre and MPAA Rating:

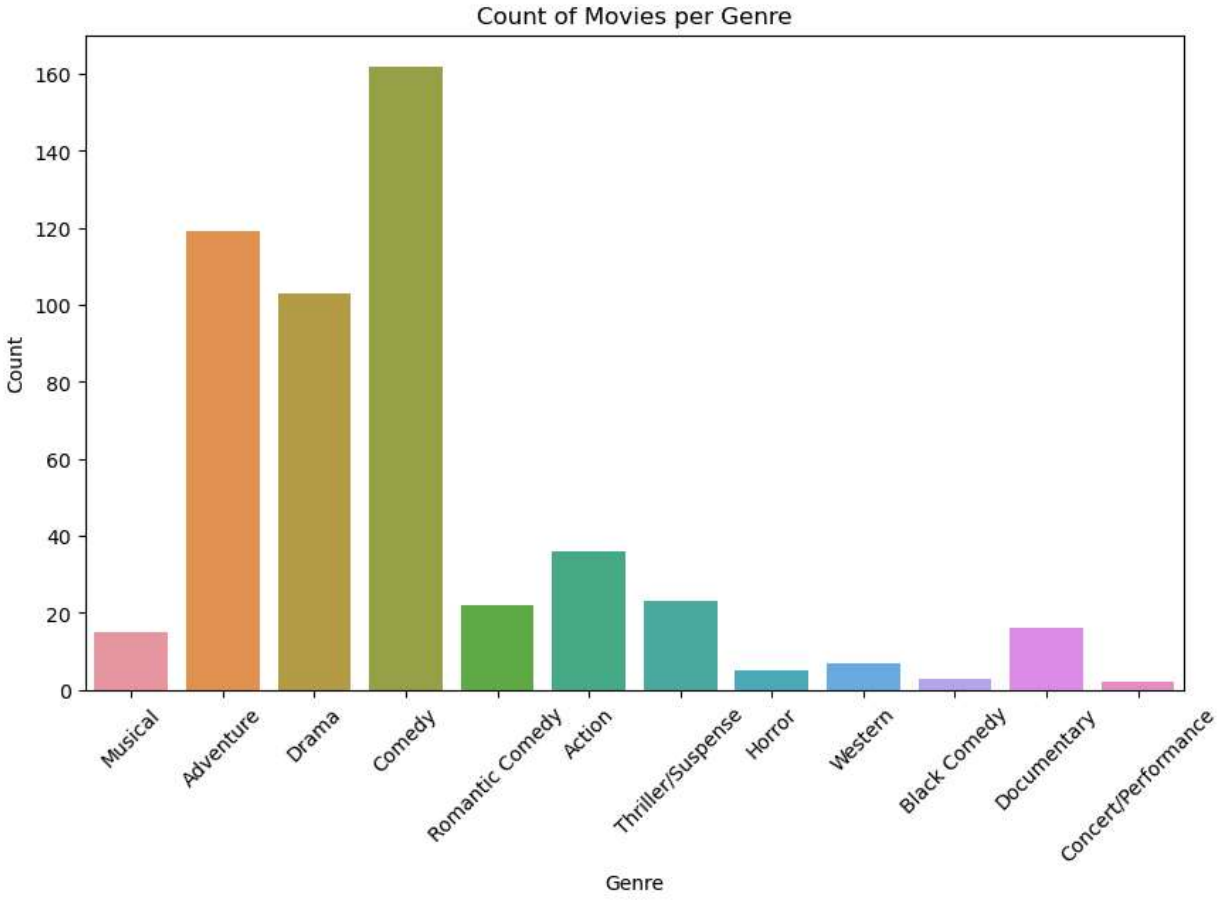
	inflation_adjusted_gross		
	mean	median	sum

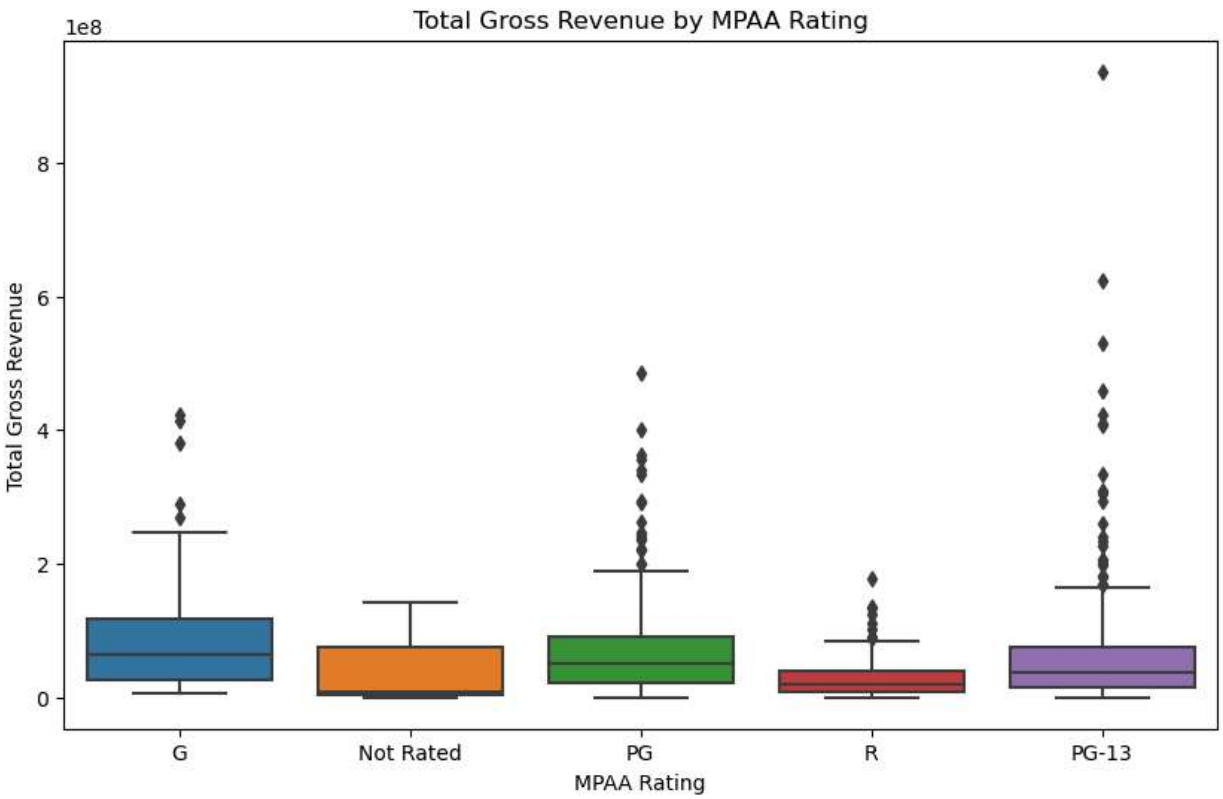
```
genre      mpaa_rating
Action     PG                 8.523316e+07   58965304.0    426165810
           PG-13              1.850755e+08   96971361.0   3516434321
           R                  1.172537e+08   63772347.5   1407044726
Adventure  G                  2.413334e+08  119539461.0  10136004565
           PG                 1.513291e+08   95208344.0   8625757598

                         total_gross
                                mean      median         sum
genre      mpaa_rating
Action     PG           4.238592e+07  29028000.0   211929620
           PG-13        1.664052e+08  85463309.0  3161698234
           R            6.240512e+07  33693405.5   748861460
Adventure  G            1.131831e+08  84818138.5  4753689326
           PG           1.289495e+08  77042381.0  7350121958
```

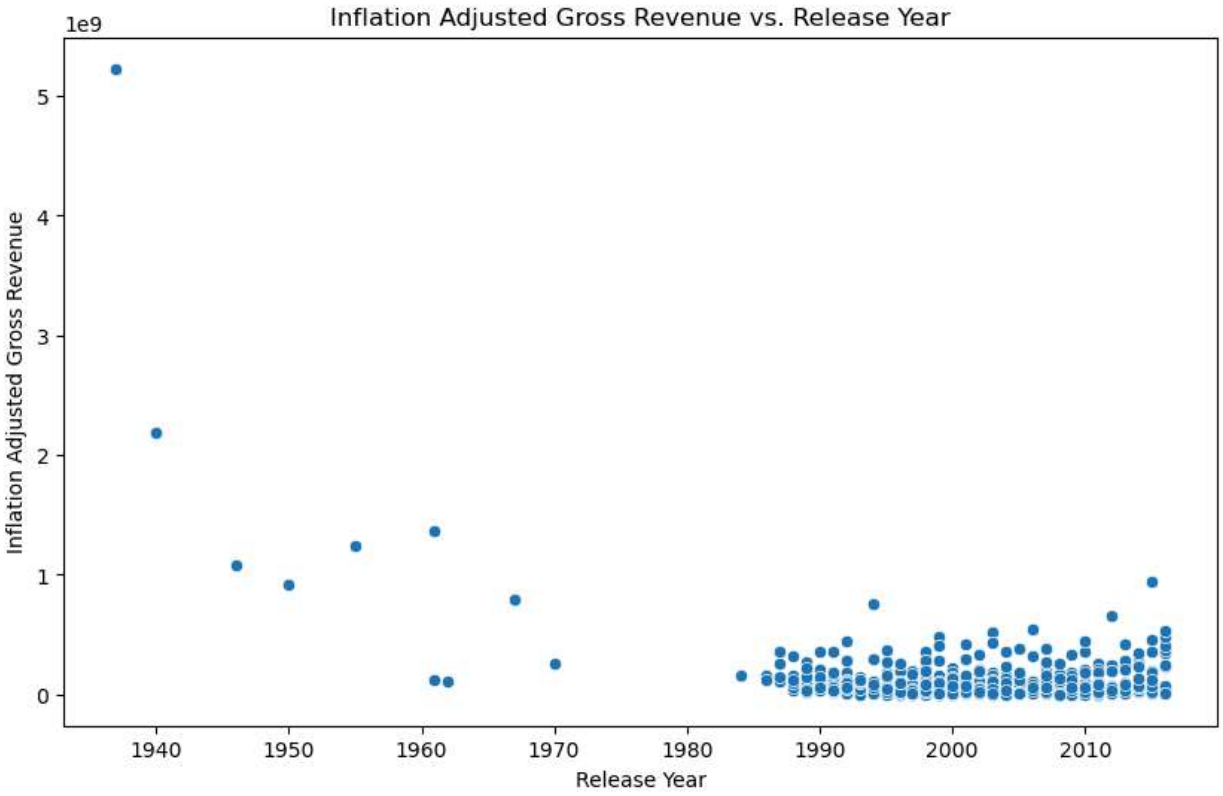
```
/var/folders/_m/wnrnbqz55qg39y6fgnd15ljr0000gp/T/ipykernel_59290/3227201047.py:14: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_disney_nona['release_year'] = pd.to_datetime(df_disney_nona['release_date']).dt.
year
```

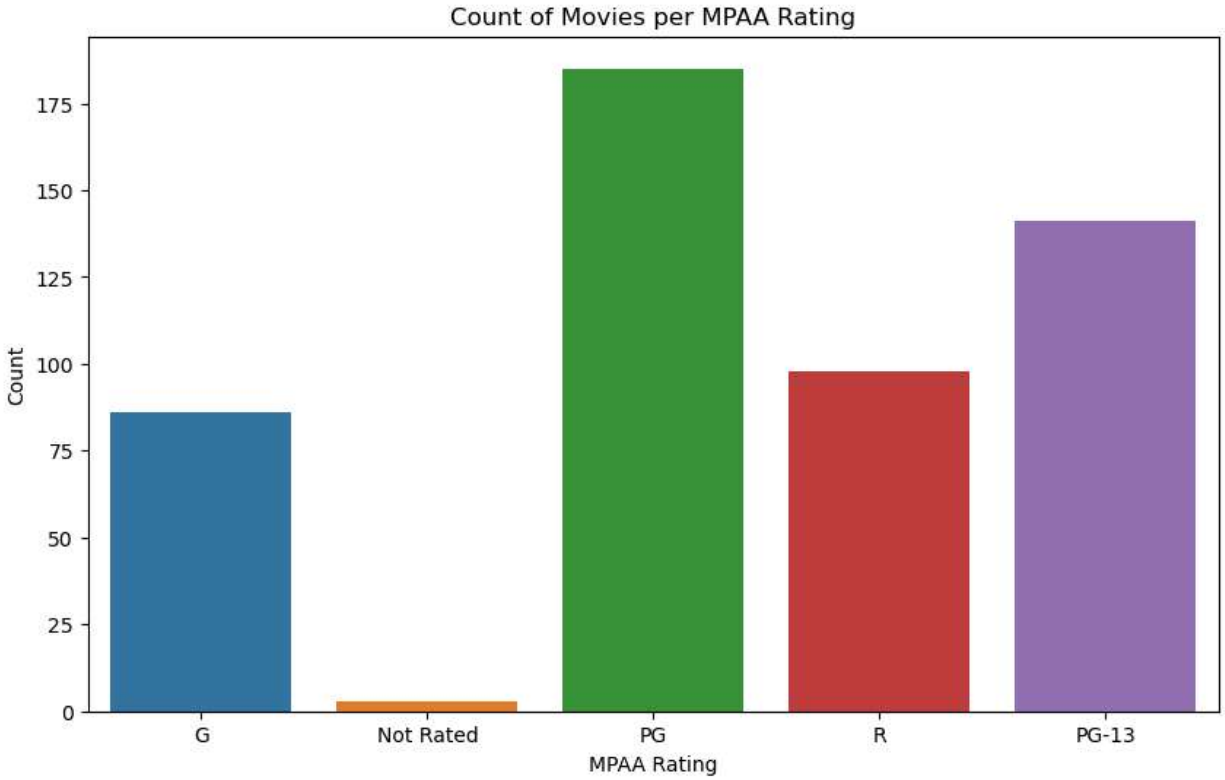
In [ ]:

```
# Visualization 1: Bar Plot - Count of Movies per Genre
plt.figure(figsize=(10, 6))
sns.countplot(data=df_disney_nona, x='genre')
plt.title('Count of Movies per Genre')
plt.xticks(rotation=45)
plt.xlabel('Genre')
plt.ylabel('Count')
plt.show()

# Visualization 2: Box Plot - Total Gross Revenue by MPAA Rating
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_disney_nona, x='mpaa_rating', y='total_gross')
plt.title('Total Gross Revenue by MPAA Rating')
plt.xlabel('MPAA Rating')
plt.ylabel('Total Gross Revenue')
plt.show()

# Visualization 3: Scatter Plot - Inflation Adjusted Gross Revenue vs. Release Year
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_disney_nona, x='release_year', y='inflation_adjusted_gross')
plt.title('Inflation Adjusted Gross Revenue vs. Release Year')
plt.xlabel('Release Year')
plt.ylabel('Inflation Adjusted Gross Revenue')
plt.show()

# Visualization 4: Bar Plot - Count of Movies per MPAA Rating
plt.figure(figsize=(10, 6))
sns.countplot(data=df_disney_nona, x='mpaa_rating')
plt.title('Count of Movies per MPAA Rating')
plt.xlabel('MPAA Rating')
plt.ylabel('Count')
plt.show()
```


Count of Movies per Genre
Count
Genre
0
20
40
60
80
100
120
140
160
Musical
Adventure
Drama
Comedy
Romantic Comedy
Action
Thriller/Suspense
Horror
Western
Black Comedy
Documentary
Concert/Performance


Total Gross Revenue by MPAA Rating
1e8
Total Gross Revenue
MPAA Rating
0
2
4
6
8
G
Not Rated
PG
R
PG-13





In [ ]:
```
# Summary Statistic 1: Average Inflation-Adjusted Gross Revenue
average_inflation_adjusted_gross = df_disney_nona['inflation_adjusted_gross'].mean()

# Summary Statistic 2: Maximum Total Gross Revenue
max_total_gross = df_disney_nona['total_gross'].max()

# Summary Statistic 3: Average Gross Revenue by Genre
average_gross_by_genre = df_disney_nona.groupby('genre')['total_gross'].mean()
```

```
# Summary Statistic 4: Number of Movies per MPAA Rating
movies_per_rating = df_disney_nona['mpaa_rating'].value_counts()

# Summary Statistic 5: Median Release Year
median_release_year = df_disney_nona['release_year'].median()

# Summary Statistic 6: Total Revenue for Each MPAA Rating
total_revenue_by_rating = df_disney_nona.groupby('mpaa_rating')['total_gross'].sum()

# Display Summary Statistics
print("Summary Statistics:")
print(f"1. Average Inflation-Adjusted Gross Revenue: ${average_inflation_adjusted_gros
print(f"2. Maximum Total Gross Revenue: ${max_total_gross:.2f}")
print("3. Average Gross Revenue by Genre:")
print(average_gross_by_genre)
print("4. Number of Movies per MPAA Rating:")
print(movies_per_rating)
print(f"5. Median Release Year: {median_release_year:.0f}")
print("6. Total Revenue for Each MPAA Rating:")
print(total_revenue_by_rating)
```

```
Summary Statistics:
1. Average Inflation-Adjusted Gross Revenue: $127594055.45
2. Maximum Total Gross Revenue: $936662225.00
3. Average Gross Revenue by Genre:
genre
Action                 1.145136e+08
Adventure              1.359535e+08
Black Comedy           3.251440e+07
Comedy                 4.808848e+07
Concert/Performance    5.172823e+07
Documentary            1.129285e+07
Drama                  3.896867e+07
Horror                 1.628256e+07
Musical                7.596087e+07
Romantic Comedy        5.093559e+07
Thriller/Suspense      6.061305e+07
Western                5.128735e+07
Name: total_gross, dtype: float64
4. Number of Movies per MPAA Rating:
PG           185
PG-13        141
R             98
G             86
Not Rated      3
Name: mpaa_rating, dtype: int64
5. Median Release Year: 1999
6. Total Revenue for Each MPAA Rating:
mpaa_rating
G             7919792693
Not Rated      151387769
PG           13764978891
PG-13        11757098250
R             2988006302
Name: total_gross, dtype: int64
```

A comprehensive analysis of the disney_movies.csv dataset has yielded several insightful findings that hold valuable implications for effective management strategies. The dataset comprises a diverse range of movies, each with unique revenue and rating attributes.

Examining the summary statistics, we discover that the average inflation-adjusted gross revenue for these Disney movies stands at approximately $127,594,055.45, refelecting a solid overall revenue performance across the dataset. The maximum total gross revenue reaches an impressive 936,662,225.00 dollars, underscoring the potential for substantial revenue generation within the industry.

By delving into the average gross revenue by genre, we recognize certain trends that can guide strategic decision-making. The genres "Adventure," "Musical," and "Action" emerge as high-earning categories, suggesting the market's receptiveness to engaging and visually appealing content. Conversely, genres like "Documentary" and "Horror" exhibit relatively lower average revenues, signaling areas where targeted marketing or content enhancement may be beneficial.

Analyzing the distribution of movies based on MPAA ratings, we note that the majority falls under the "PG" and "PG-13" categories. This insight highlights the significance of family-oriented and broadly appealing content, indicating that such content has resonated well with audiences and contributed to higher revenue generation. In contrast, "Not Rated" movies constitute a small proportion, potentially indicating niche or specialized content that appeals to a more selective audience.

The median release year of 1999 indicates the temporal spread of movies in the dataset, with a potential opportunity for leveraging nostalgia to engage audiences. Moreover, examining the total revenue for each MPAA rating category underscores the dominance of "PG" and "PG-13" movies in revenue generation, while "G" and "R" categories contribute substantial revenue as well.

Thus, our exploratory analysis of the disney_movies.csv dataset offers valuable insights for management. The prominence of certain genres and rating categories in revenue generation underscores the importance of aligning content creation and marketing efforts with these preferences. Management can strategically allocate resources, prioritize specific genres, and tailor marketing campaigns to optimize revenue potential. These findings provide actionable intelligence that empowers Disney to remain competitive and successful in the dynamic landscape of the film industry.

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

In [ ]:
```
df_family=pd.read_csv('/family_segments.csv')
```

In [ ]:
```
df_family
```

Out[ ]:

	hhold_ID	est_net_worth	est_inc	domestic	number_visits	number_children	oldest_child_age
0	1	7525	7825	1	0	2	9.0
1	2	639815	449775	1	1	2	11.0
2	3	288847	204750	1	6	3	13.0
3	4	191791	136730	1	1	1	8.0
4	5	96527	72165	0	3	0	NaN
...	...	...	...	...	...	...	...
9245	9246	129634	95375	1	1	2	11.0
9246	9247	5039615	3526325	1	8	2	17.0
9247	9248	259279	184580	1	0	2	9.0
9248	9249	43471	33375	1	1	2	13.0
9249	9250	184765	134550	1	2	2	12.0

9250 rows × 12 columns

In [ ]:
```
df_family.describe()
```

Out[ ]:

	hhold_ID	est_net_worth	est_inc	domestic	number_visits	number_children	oldest
count	9250.000000	9.250000e+03	9.250000e+03	9250.000000	9250.000000	9250.000000	87
mean	4625.500000	6.739056e+05	4.758774e+05	0.724216	2.669946	2.224000	
std	2670.389329	1.335709e+06	8.764279e+05	0.446933	3.295406	1.124236	
min	1.000000	5.100000e+01	3.700000e+02	0.000000	0.000000	0.000000	
25%	2313.250000	4.264950e+04	3.327000e+04	0.000000	1.000000	1.000000	
50%	4625.500000	1.666520e+05	1.224650e+05	1.000000	2.000000	2.000000	
75%	6937.750000	6.488230e+05	4.553300e+05	1.000000	3.000000	3.000000	
max	9250.000000	9.883250e+06	5.497405e+06	1.000000	47.000000	7.000000	

In [ ]:
```
df_family.isnull().values.any()
```

Out[ ]:
```
True
```

In [ ]:
```
df_family.isna().sum()
```

Out[ ]:
```
hhold_ID               0
est_net_worth          0
est_inc                0
domestic               0
number_visits          0
number_children        0
oldest_child_age     526
dis_plus_sub           0
online_merch_avg       0
est_annual_travel      0
est_annual_leisure     0
est_hhold_FICO         0
dtype: int64
```

In [ ]:
```
df_fam_nona = df_family.dropna()
```

In [ ]:
```
df_fam_nona.isna().sum()
```

Out[ ]:
```
hhold_ID             0
est_net_worth        0
est_inc              0
domestic             0
number_visits        0
number_children      0
oldest_child_age     0
dis_plus_sub         0
online_merch_avg     0
est_annual_travel    0
est_annual_leisure   0
est_hhold_FICO       0
dtype: int64
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_fam_nona)
df_fam_norm = scaler.transform(df_fam_nona)
df_fam_norm = pd.DataFrame(data=df_fam_nona)
df_fam_norm.round(3)
```

Out[ ]:

	hhold_ID	est_net_worth	est_inc	domestic	number_visits	number_children	oldest_child_age
0	1	7525	7825	1	0	2	9.0
1	2	639815	449775	1	1	2	11.0
2	3	288847	204750	1	6	3	13.0
3	4	191791	136730	1	1	1	8.0
5	6	2435034	1705830	1	7	2	14.0
...	...	...	...	...	...	...	...
9245	9246	129634	95375	1	1	2	11.0
9246	9247	5039615	3526325	1	8	2	17.0
9247	9248	259279	184580	1	0	2	9.0
9248	9249	43471	33375	1	1	2	13.0
9249	9250	184765	134550	1	2	2	12.0

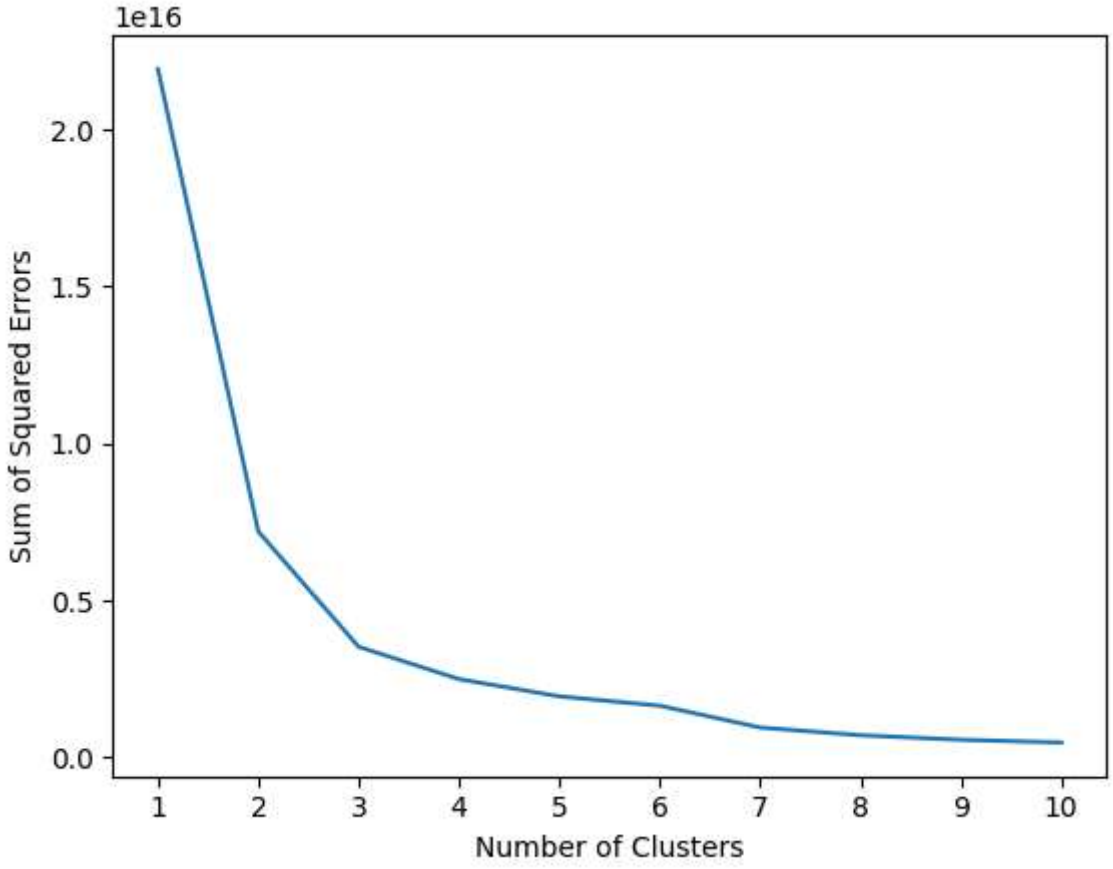
8724 rows × 12 columns

In [ ]:
```
from sklearn.cluster import KMeans
```

In [ ]:
```
kmeans_kwargs = {"init": "random", "n_init": 10, "random_state": 1,}
```

In [ ]:
```
sum_squared_errors = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(df_fam_norm)
    sum_squared_errors.append(kmeans.inertia_)
```

In [ ]:
```
plt.plot(range(1, 11), sum_squared_errors)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("Sum of Squared Errors")
plt.show()
```



In [ ]:
```
k_means = KMeans(init="random", n_clusters=3, n_init=10, random_state=1)
k_means.fit(df_fam_norm)
k_means.labels_
```

Out[ ]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [ ]:
```
df_fam_norm['cluster'] = k_means.labels_
df_fam_norm
```

Out[ ]:

	hhold_ID	est_net_worth	est_inc	domestic	number_visits	number_children	oldest_child_age
0	1	7525	7825	1	0	2	9.0
1	2	639815	449775	1	1	2	11.0
2	3	288847	204750	1	6	3	13.0
3	4	191791	136730	1	1	1	8.0
5	6	2435034	1705830	1	7	2	14.0
...	...	...	...	...	...	...	...
9245	9246	129634	95375	1	1	2	11.0
9246	9247	5039615	3526325	1	8	2	17.0
9247	9248	259279	184580	1	0	2	9.0
9248	9249	43471	33375	1	1	2	13.0
9249	9250	184765	134550	1	2	2	12.0

8724 rows × 13 columns

In [ ]:
```
df_fam_kmeans = df_fam_norm.groupby(['cluster'])
df_fam_kmeans.describe()
```

Out[ ]:

	hhold_ID								est_net_worth	
	count	mean	std	min	25%	50%	75%	max	count	mean
cluster										
0	1148.0	4643.639373	2647.867130	6.0	2353.75	4573.0	7002.75	9241.0	1148.0	1.937973e+06
1	7256.0	4604.876378	2673.733123	1.0	2282.75	4596.5	6917.25	9250.0	7256.0	2.228949e+05
2	320.0	4941.668750	2596.126776	39.0	2957.50	5081.0	7122.00	9247.0	320.0	6.240259e+06

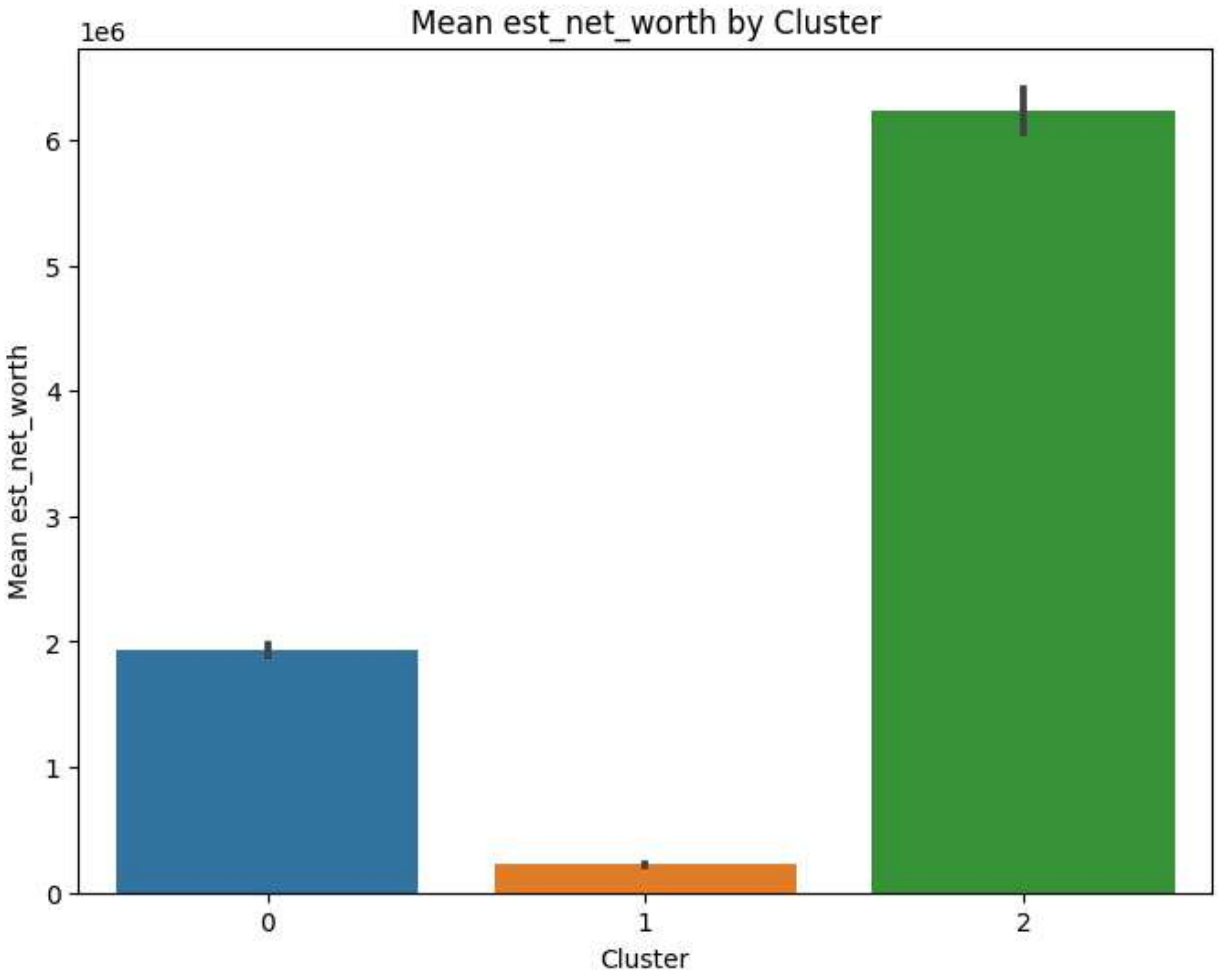
3 rows × 96 columns

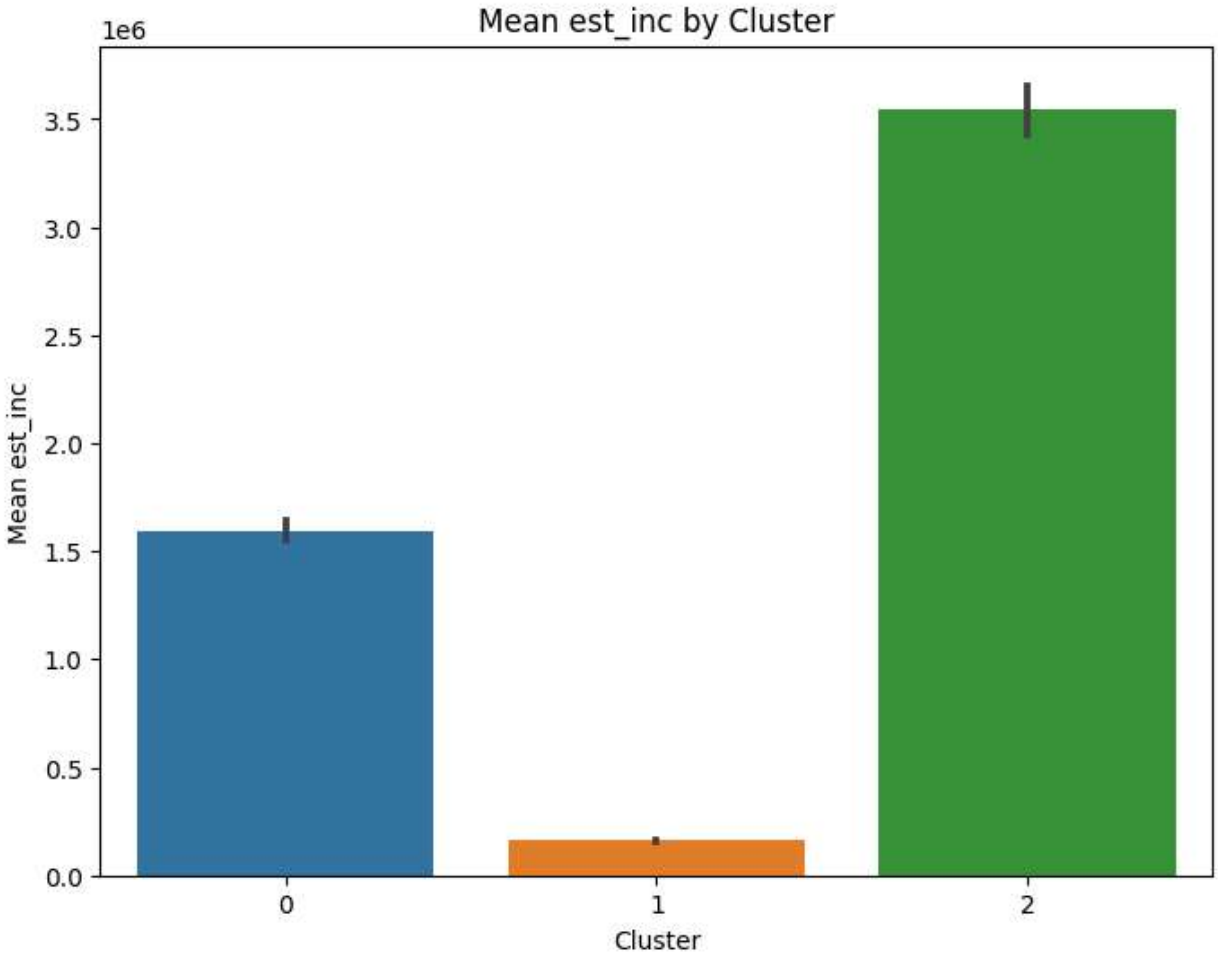
In [ ]:
```
df_fam_kmeans.agg({
    'hhold_ID':'mean', 'est_net_worth':'mean', 'est_inc':'mean', 'domestic':'mean', 'r
}).round(1)
```

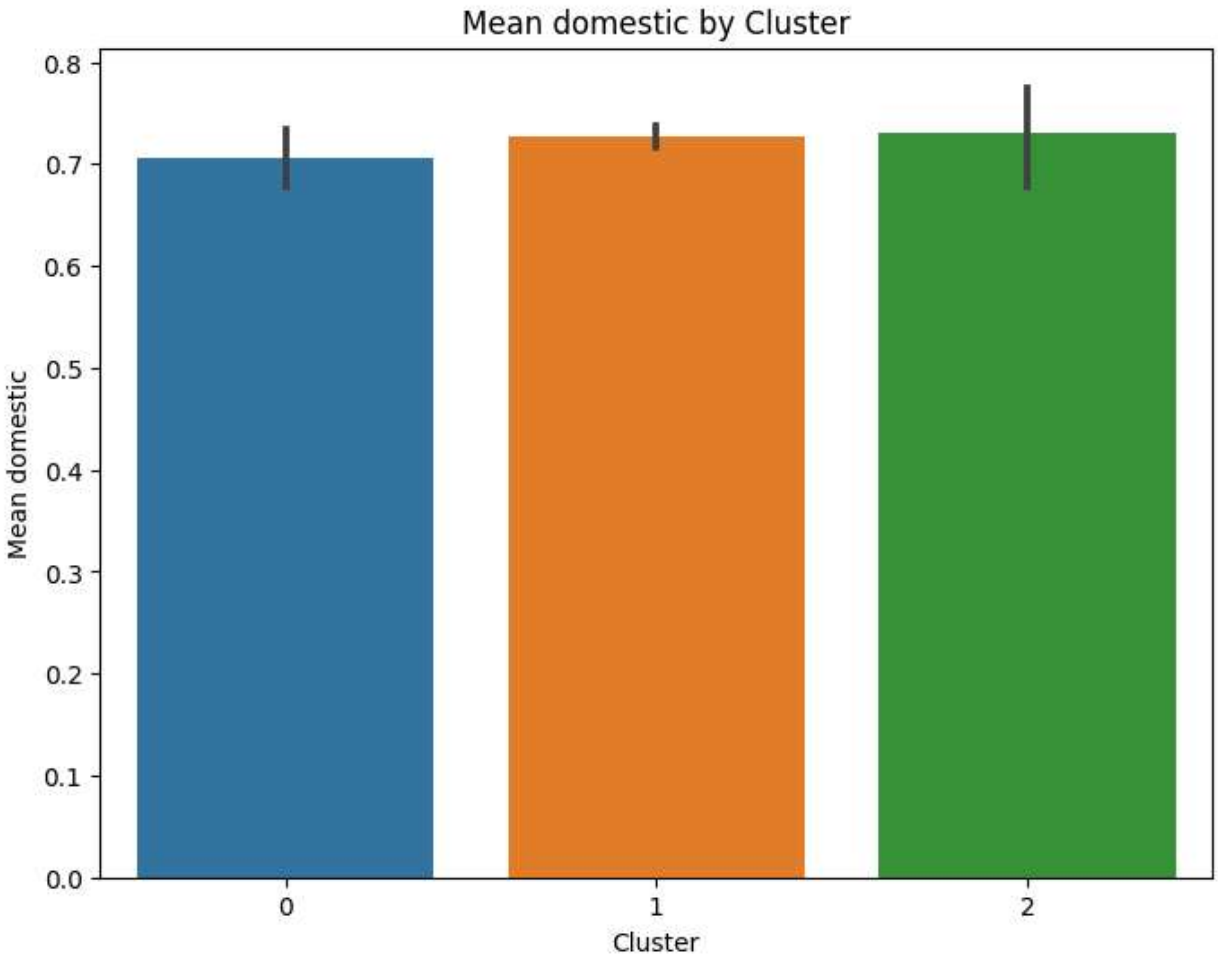
Out[ ]:

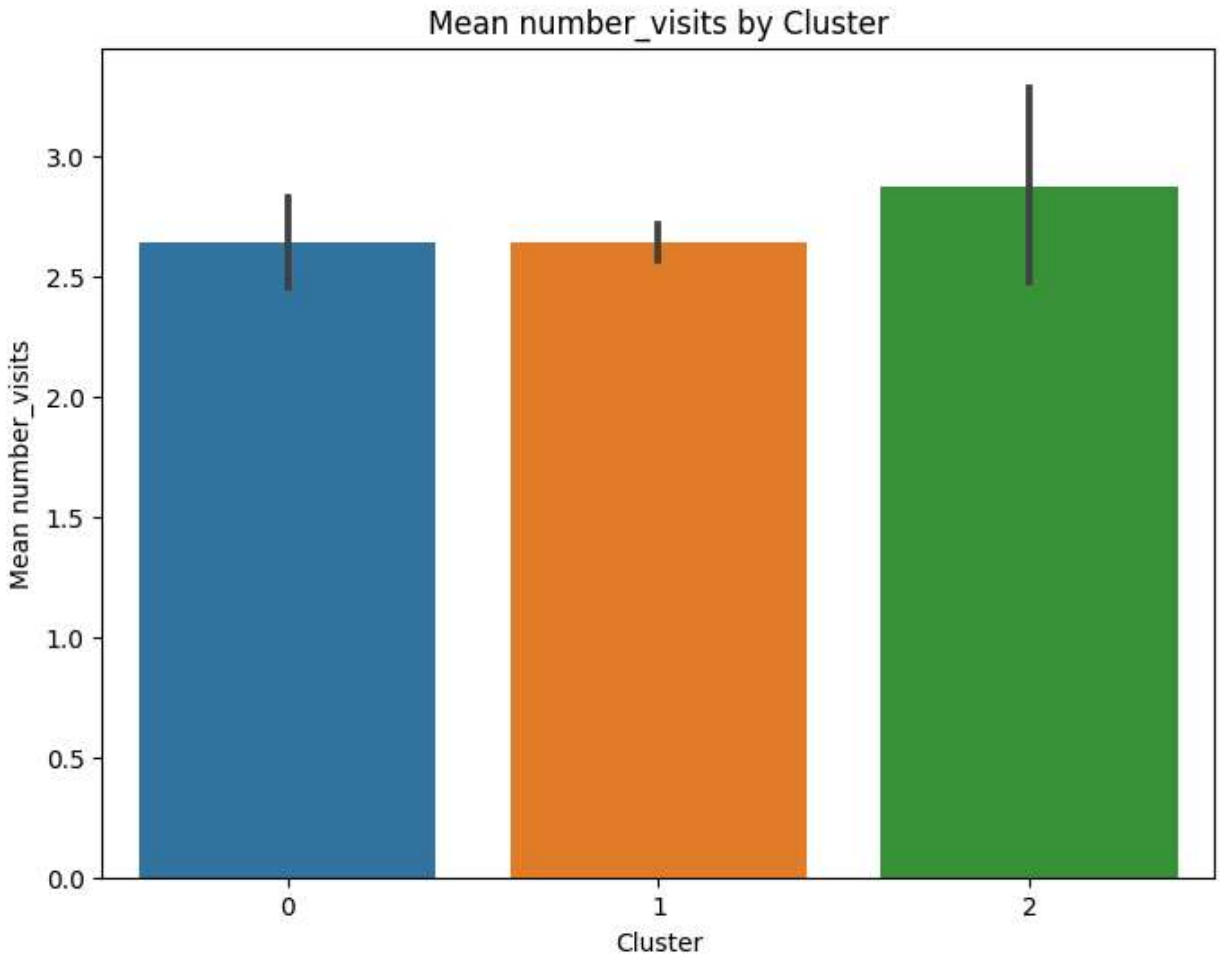
cluster	hhold_ID	est_net_worth	est_inc	domestic	number_visits	number_children	oldest_child_ag
0	4643.6	1937973.3	1595801.2	0.7	2.6	2.4	11
1	4604.9	222894.9	160273.2	0.7	2.6	2.4	12
2	4941.7	6240259.0	3543241.7	0.7	2.9	2.3	12

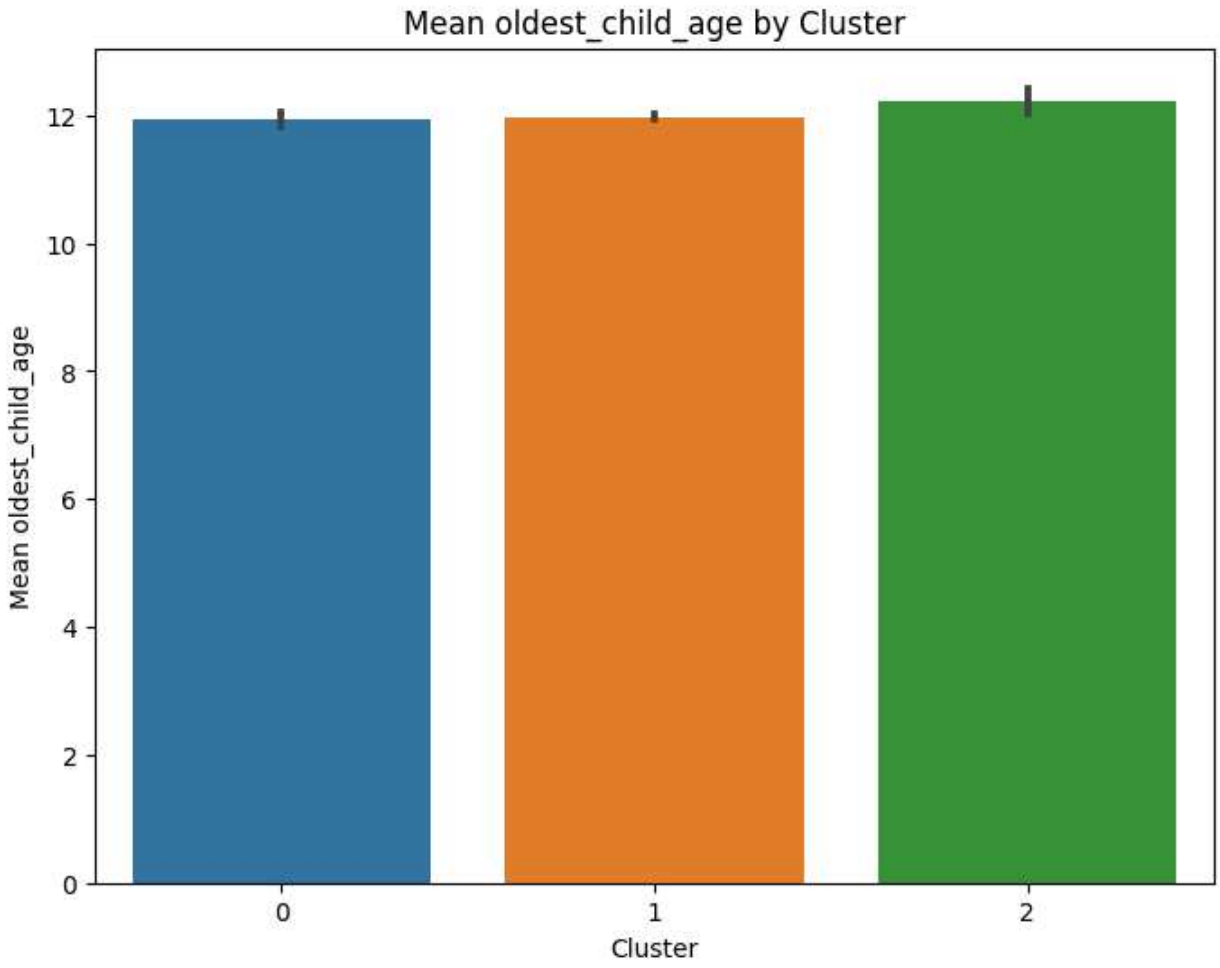
In [ ]:
```
attributes_to_visualize = ['est_net_worth', 'est_inc', 'domestic','number_visits', 'ol

for attribute in attributes_to_visualize:
    plt.figure(figsize=(8, 6))
    sns.barplot(x='cluster', y=attribute, data=df_fam_norm)
    plt.title(f"Mean {attribute} by Cluster")
    plt.xlabel("Cluster")
    plt.ylabel(f"Mean {attribute}")
    plt.show()
```

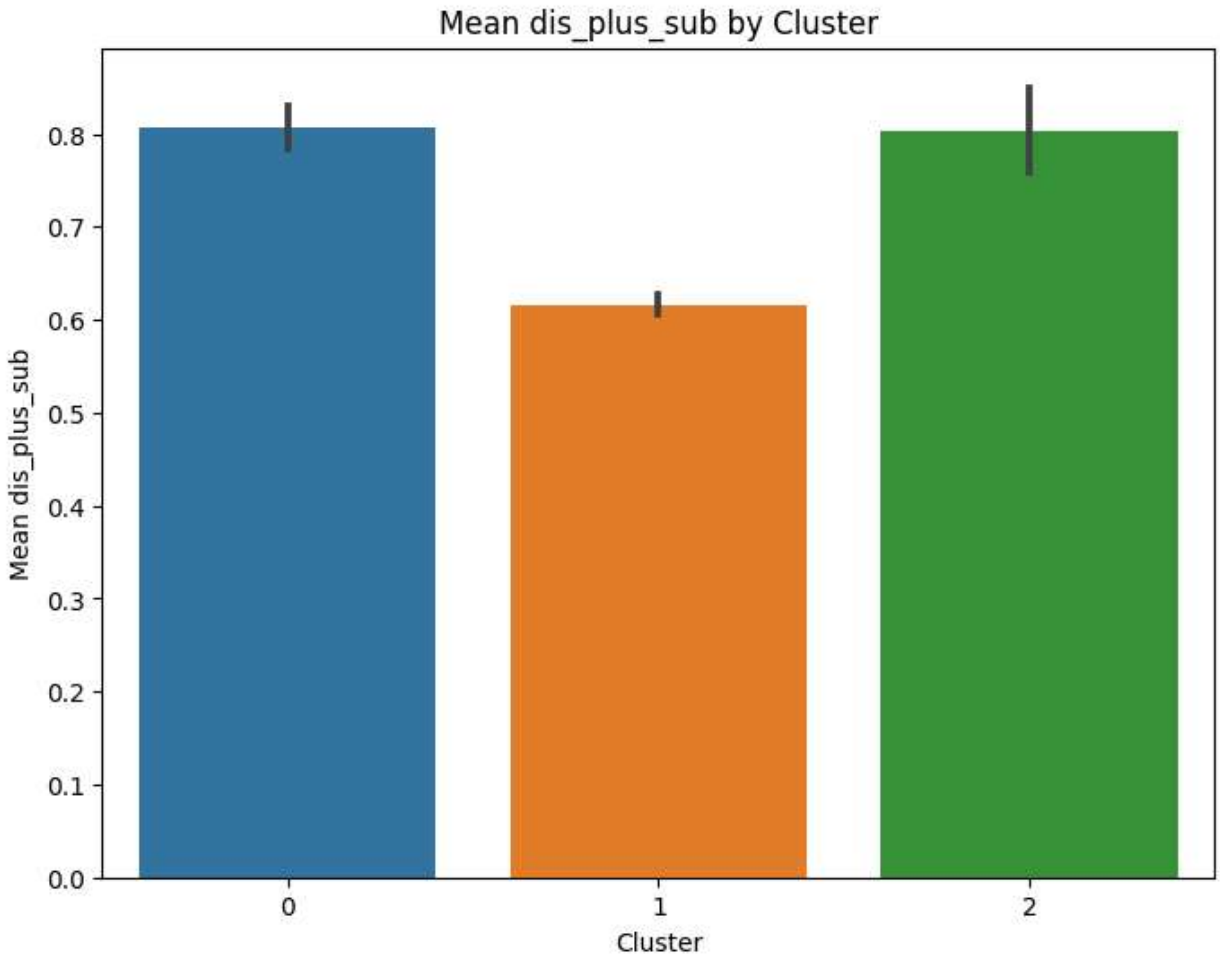


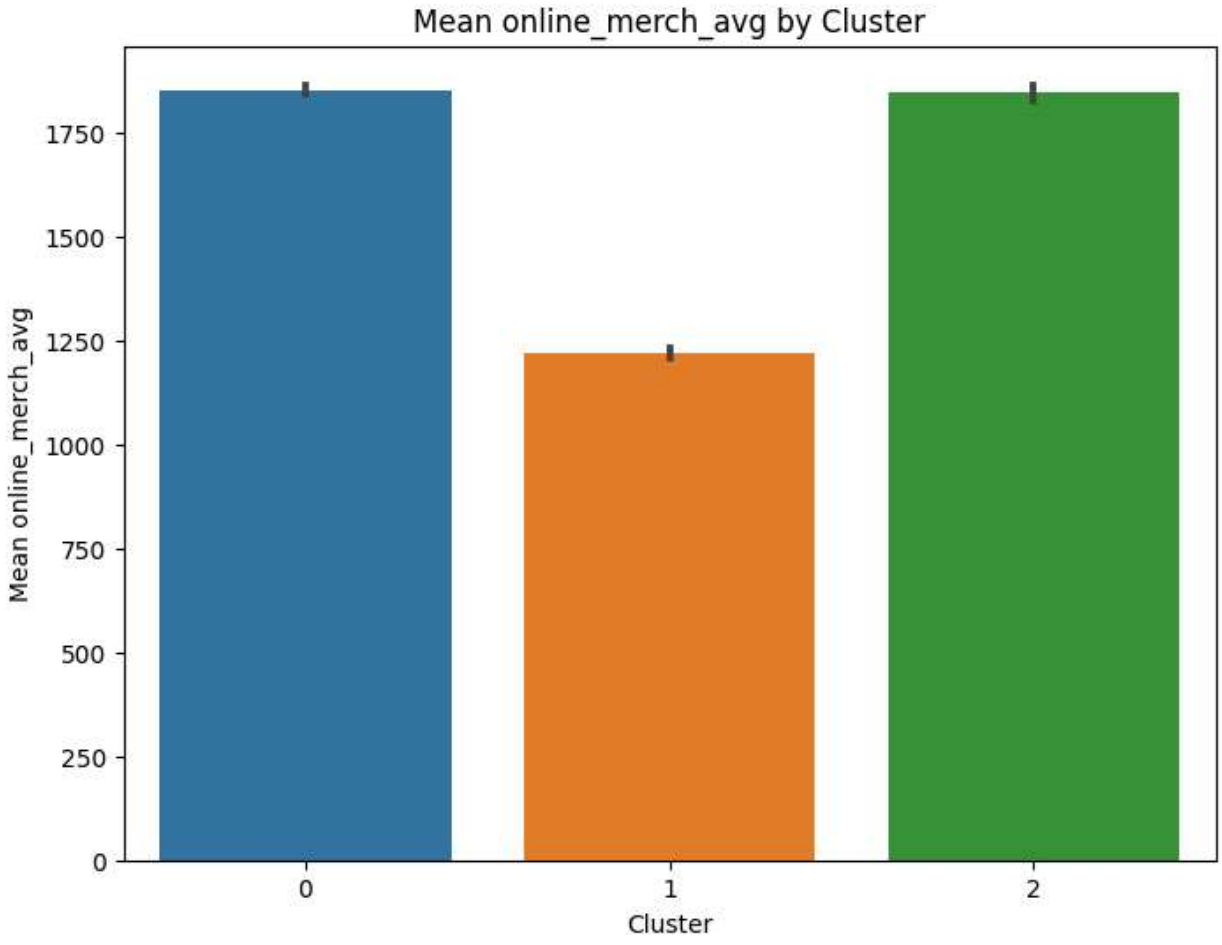

Mean est_inc by Cluster
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Mean est_inc
0
1
2
Cluster

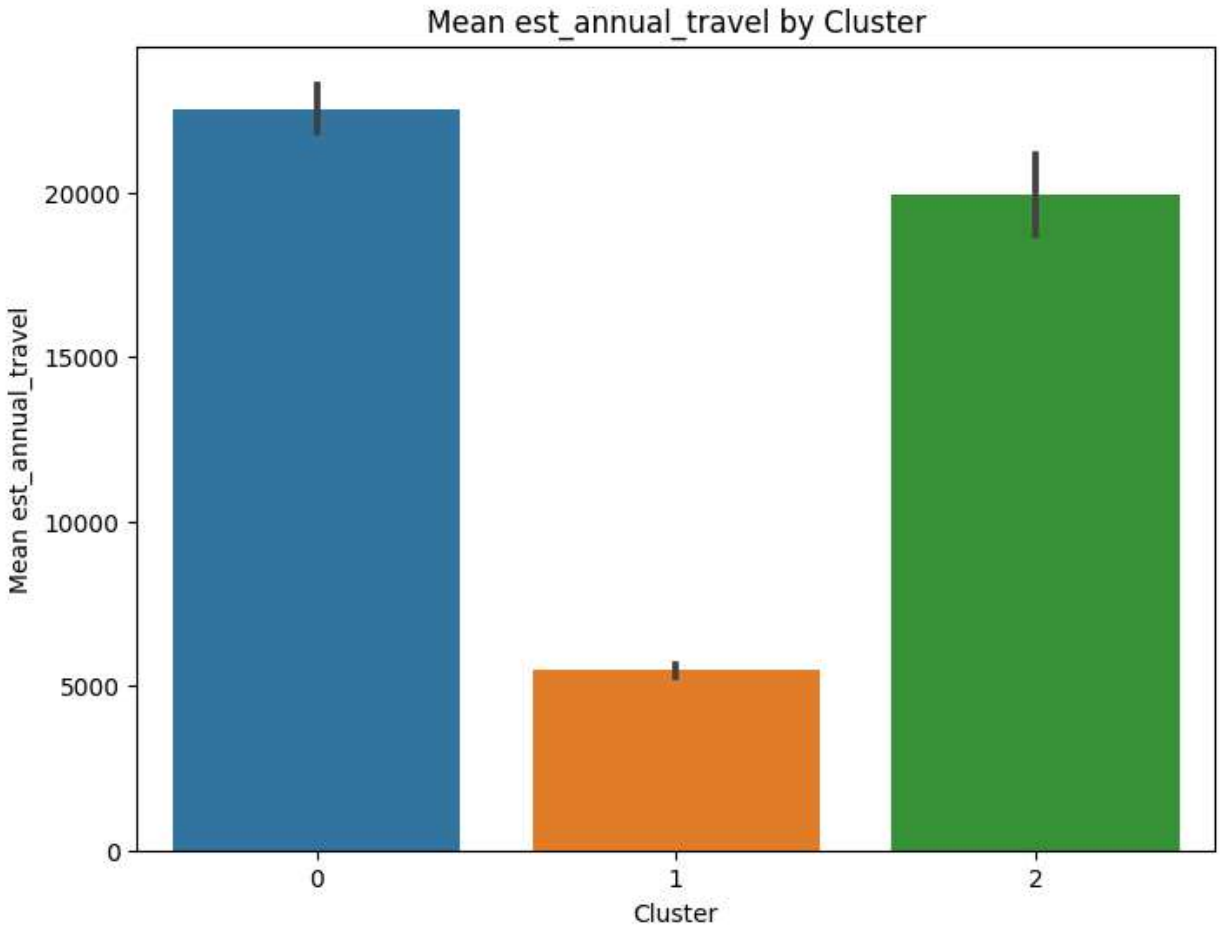

Mean domestic by Cluster
Mean domestic
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
2
Cluster

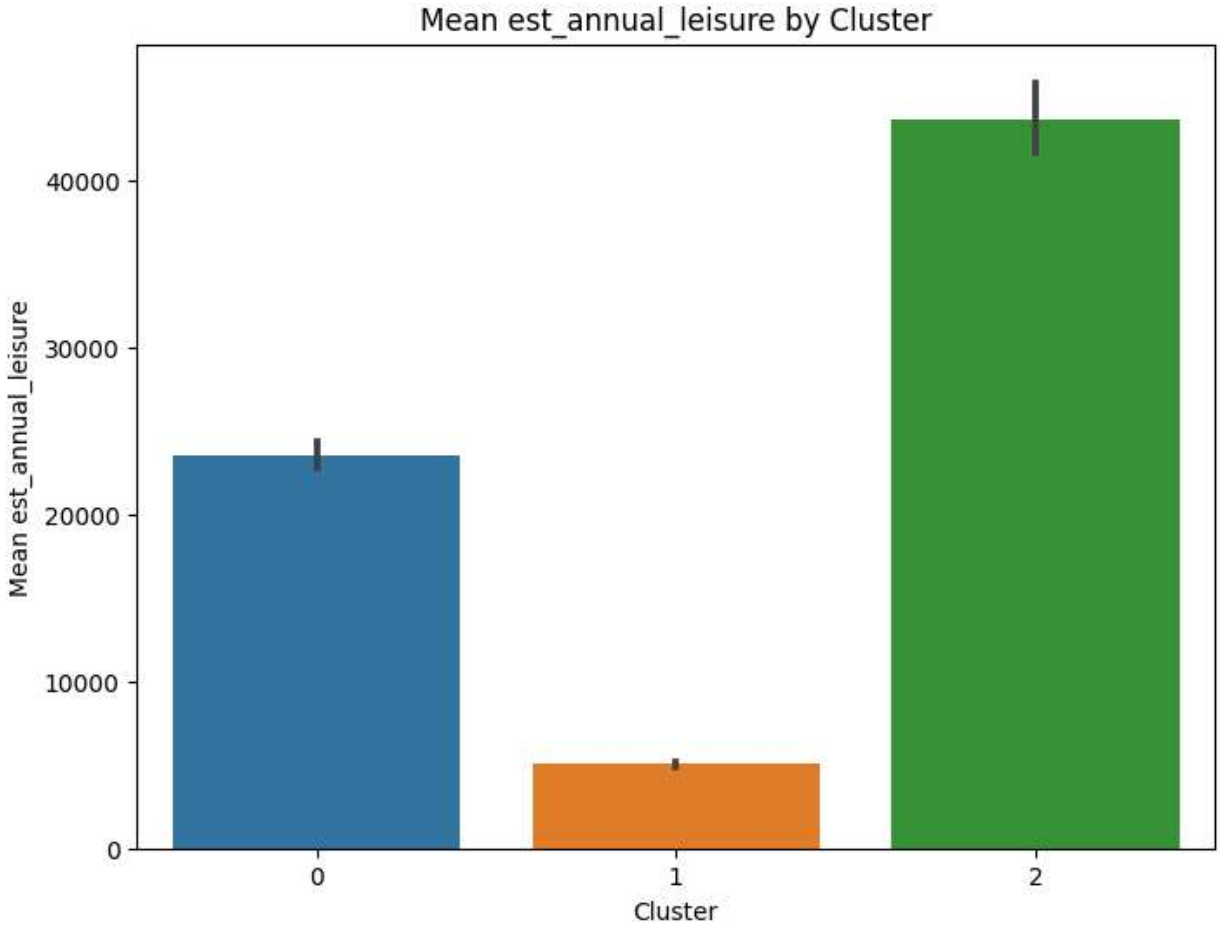

Mean number_visits by Cluster
Mean number_visits
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
Cluster

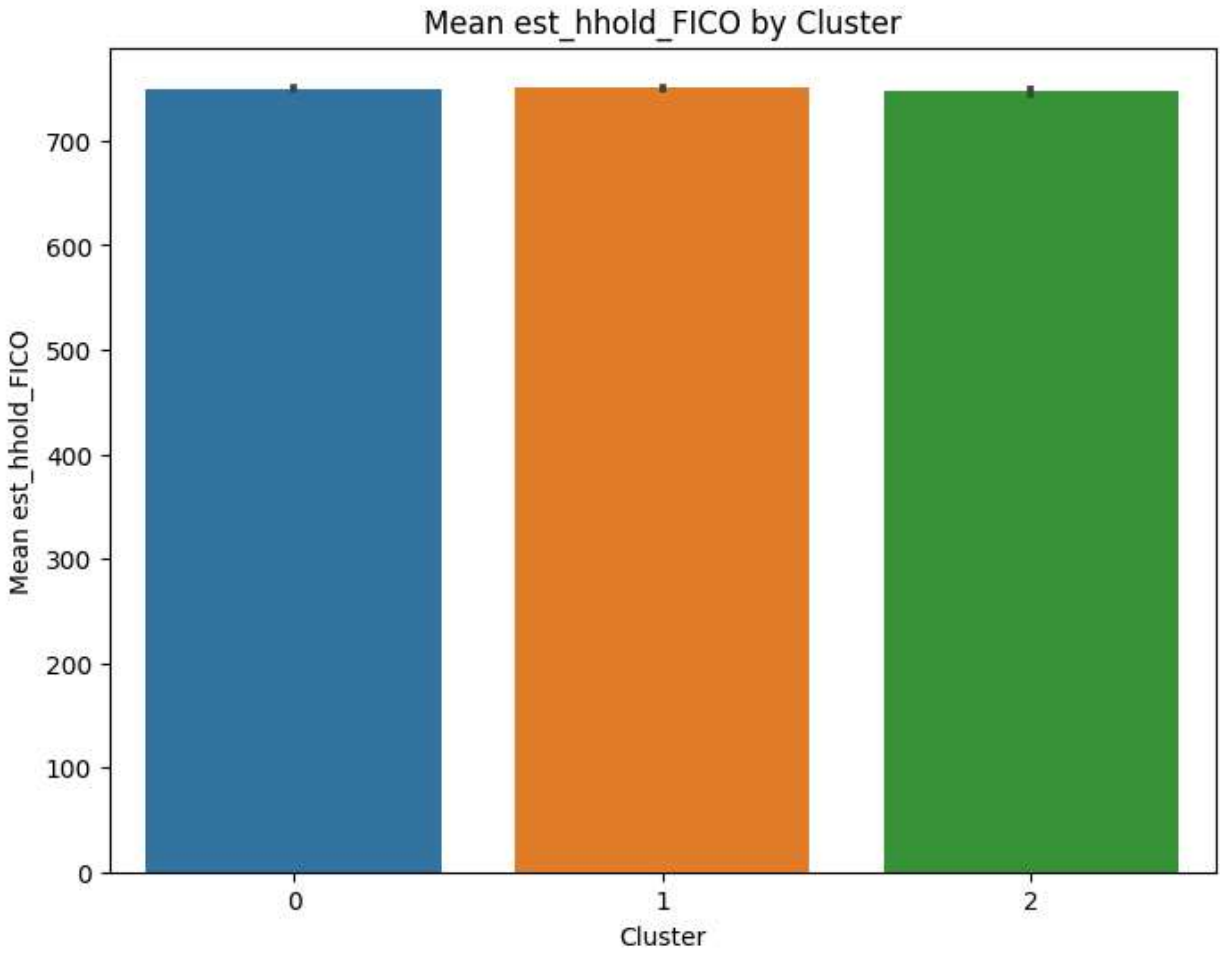

Mean oldest_child_age by Cluster
Mean oldest_child_age
0
2
4
6
8
10
12
0
1
2
Cluster


Mean dis_plus_sub by Cluster
Mean dis_plus_sub
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
2
Cluster

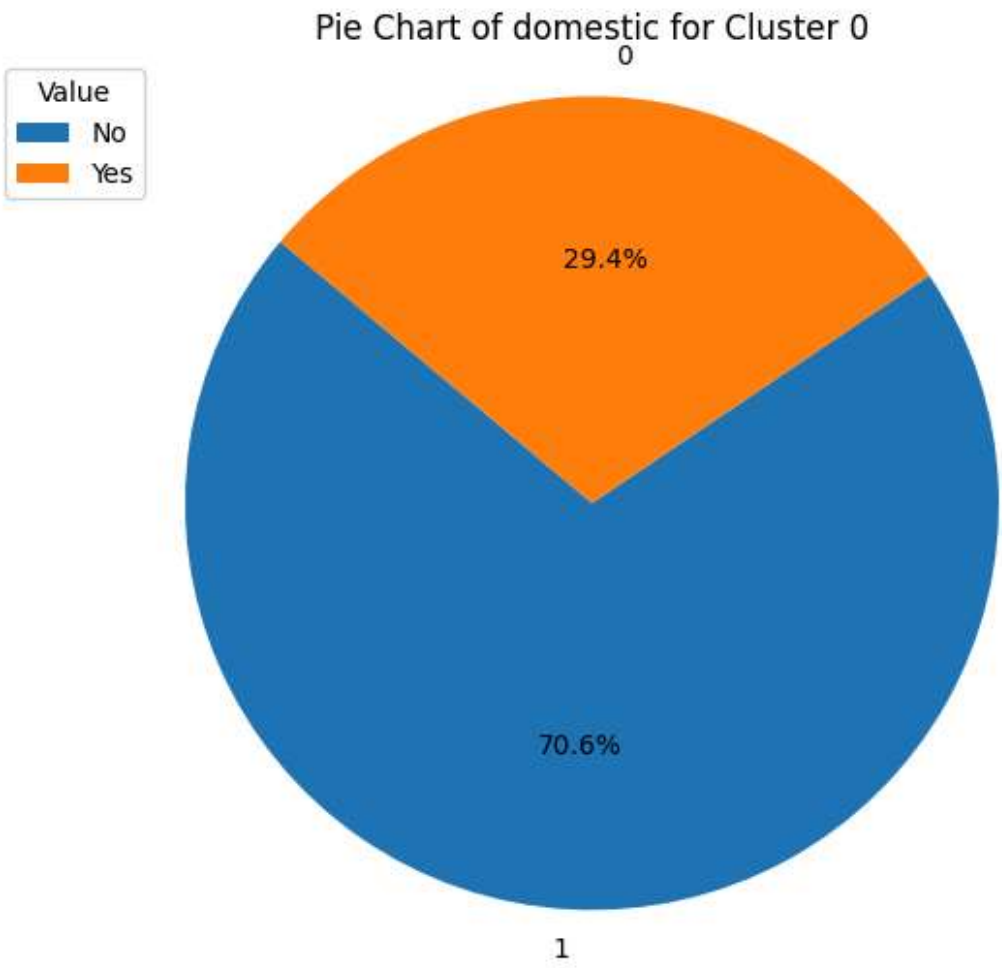

Mean online_merch_avg by Cluster
Mean online_merch_avg
0
250
500
750
1000
1250
1500
1750
0
1
2
Cluster

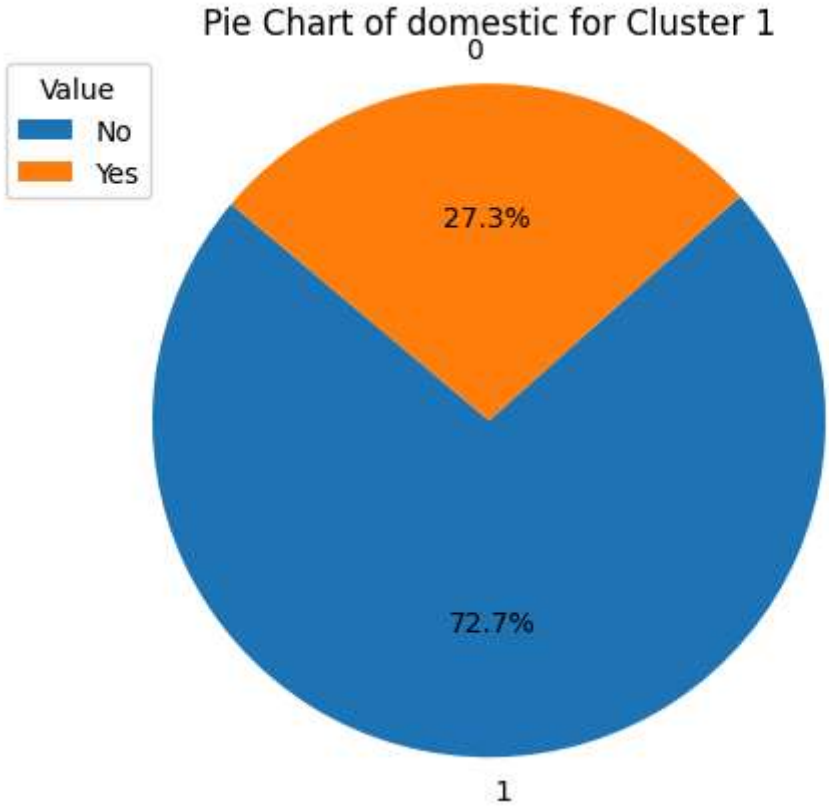

Mean est_annual_travel by Cluster
Mean est_annual_travel
0
5000
10000
15000
20000
0
1
2
Cluster

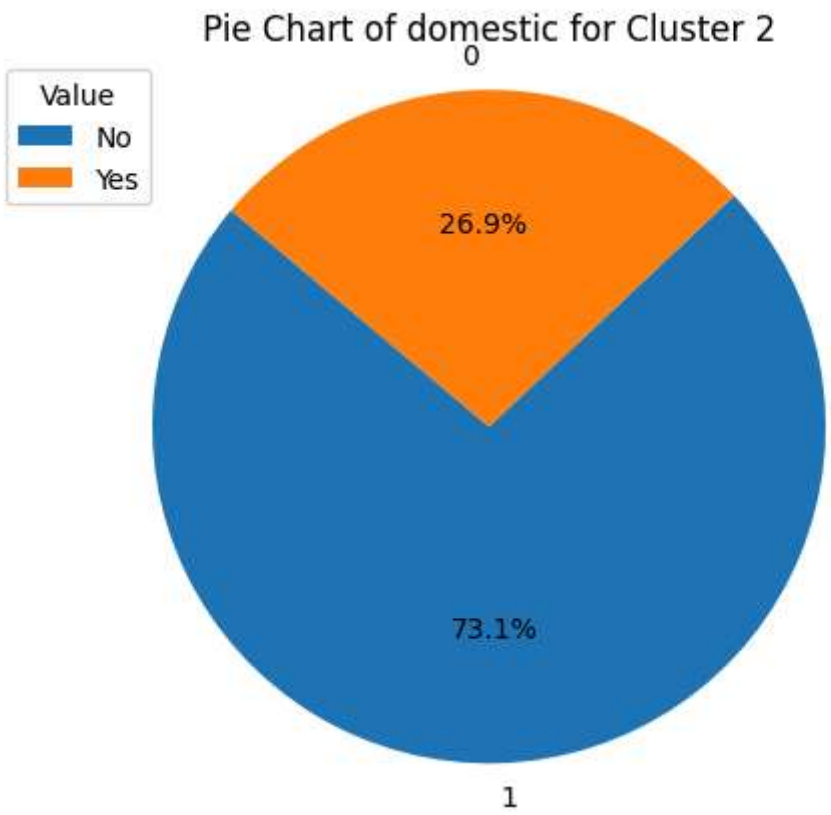

Mean est_annual_leisure by Cluster
Mean est_annual_leisure
0
10000
20000
30000
40000
0
1
2
Cluster

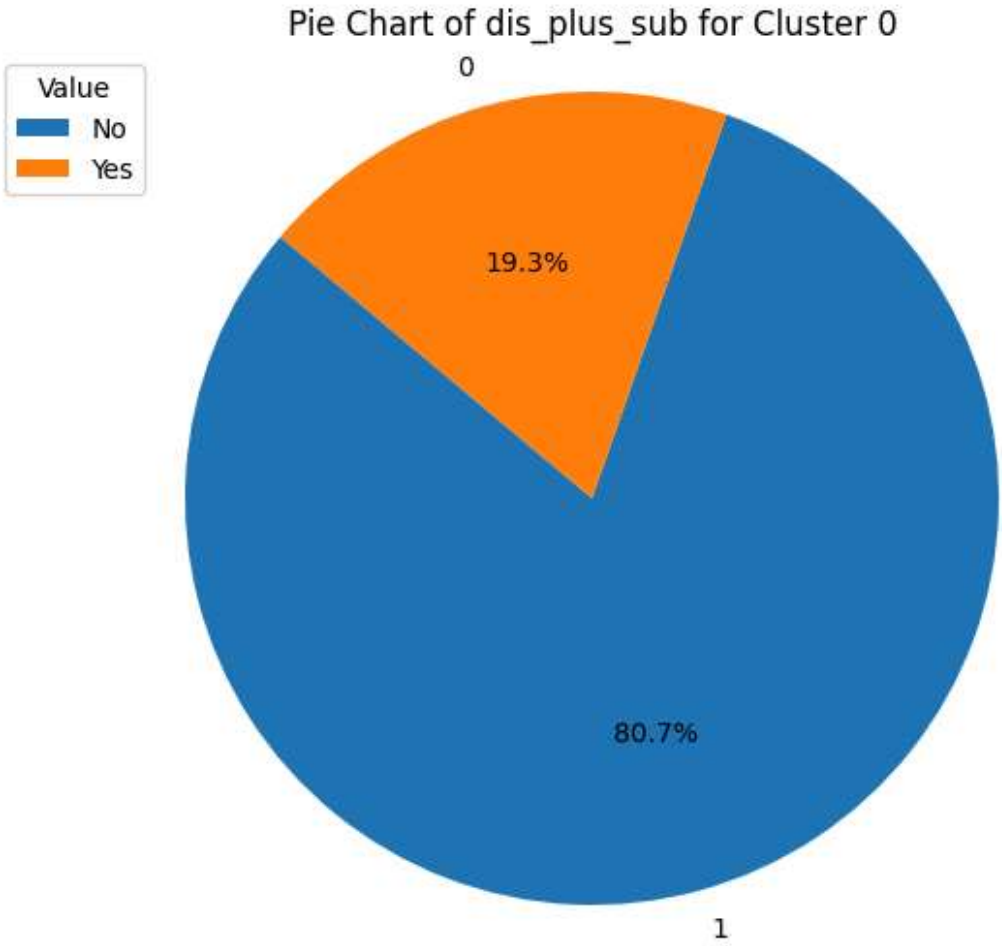


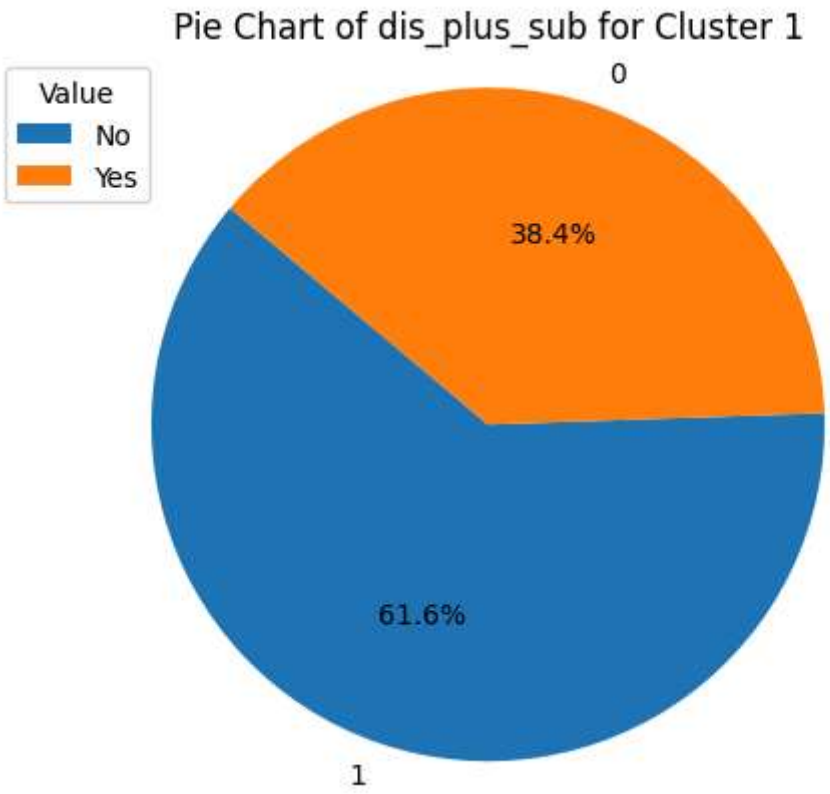
In [ ]:
```
categorical_columns = ['domestic', 'dis_plus_sub']

for column in categorical_columns:
    plt.figure(figsize=(8, 6))
    for cluster_label in range(3):
        cluster_data = df_fam_norm[df_fam_norm['cluster'] == cluster_label]
        counts = cluster_data[column].value_counts()
        colors = ['tab:blue', 'tab:orange']  # Customize colors for '0' and '1'
        plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=140, colors
        plt.title(f"Pie Chart of {column} for Cluster {cluster_label}")
        plt.axis('equal')

        # Create color legend
        legend_labels = ['No', 'Yes']  # Customize legend labels
        plt.legend(title="Value", labels=legend_labels, loc='upper left')  # Customize
        plt.show()
```

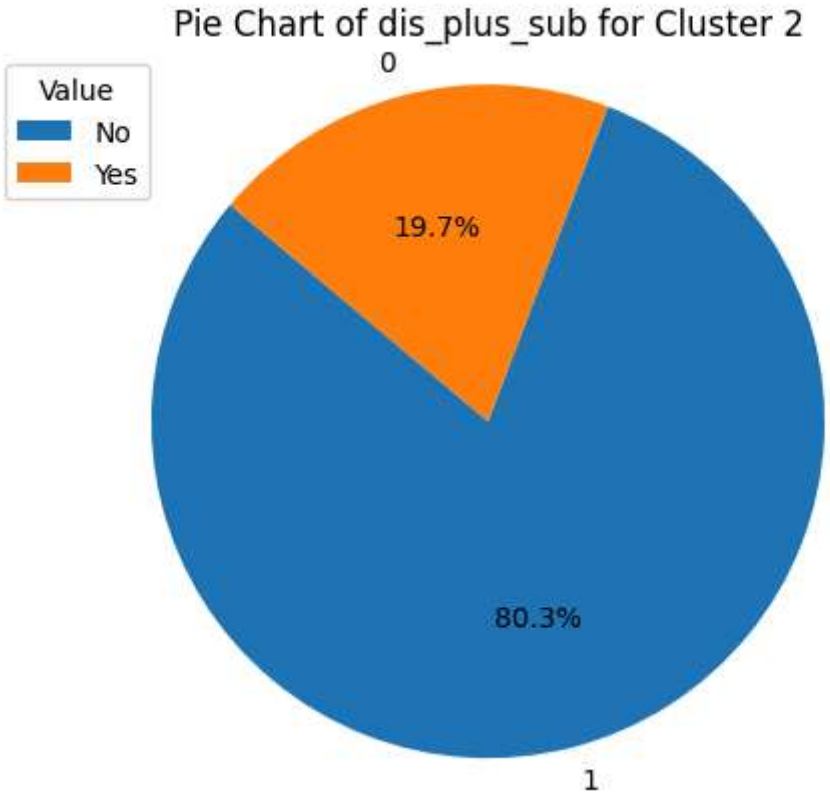

Pie Chart of domestic for Cluster 0
0
Value
No
Yes
29.4%
70.6%
1


Pie Chart of domestic for Cluster 1
0
Value
No
Yes
27.3%
72.7%
1


Pie Chart of domestic for Cluster 2
0
Value
No
Yes
26.9%
73.1%
1

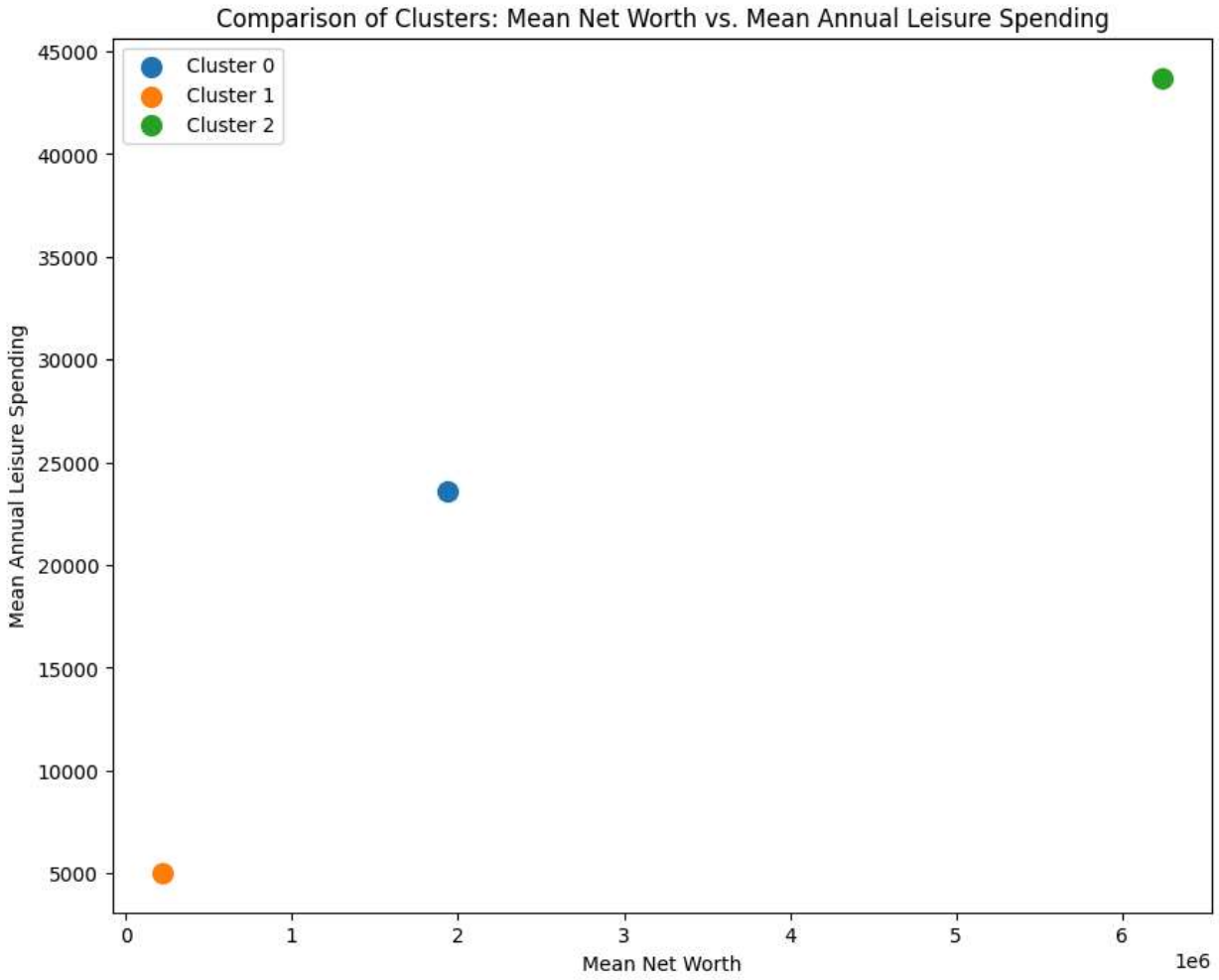

Pie Chart of dis_plus_sub for Cluster 0
0
Value
No
Yes
19.3%
80.7%
1



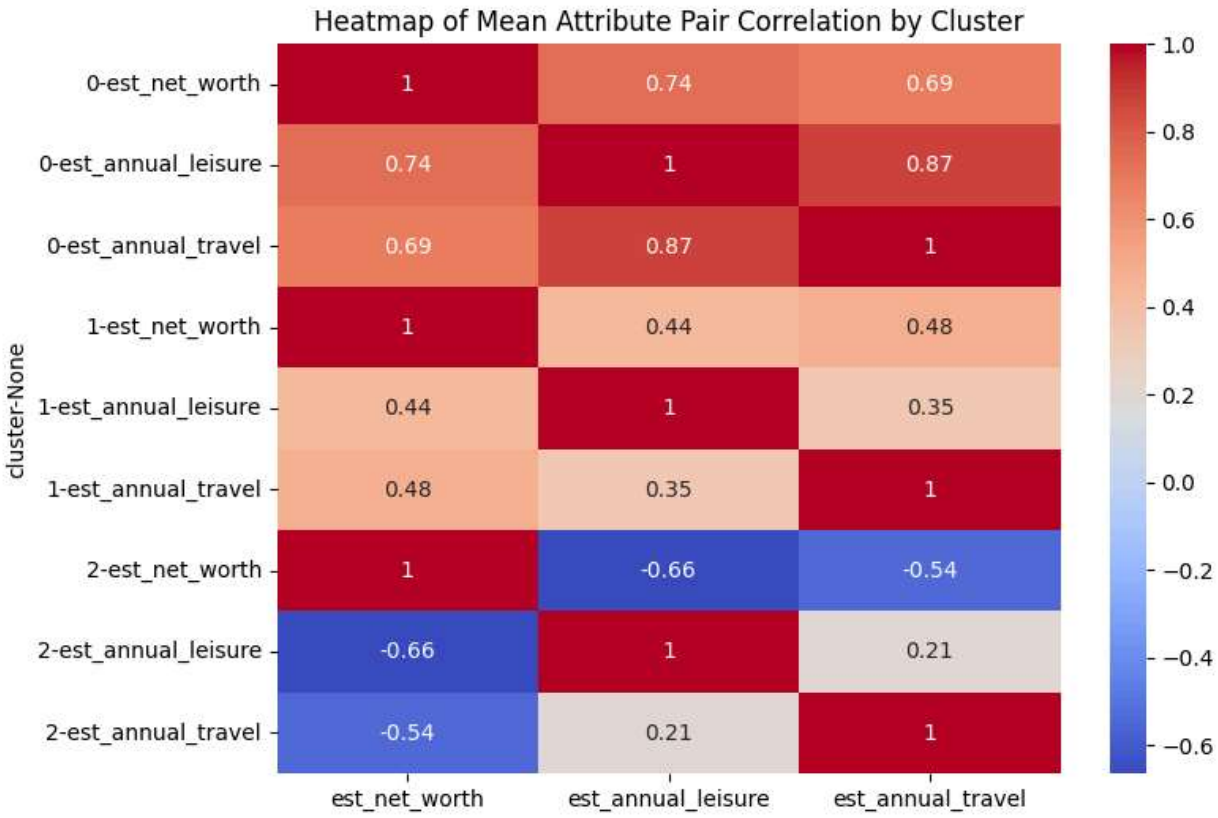


In [ ]:
```
plt.figure(figsize=(10, 8))

for cluster_label in range(3):  # Assuming you have 3 clusters
    cluster_data = df_fam_norm[df_fam_norm['cluster'] == cluster_label]
    mean_net_worth = cluster_data['est_net_worth'].mean()
    mean_annual_leisure = cluster_data['est_annual_leisure'].mean()

    plt.scatter(mean_net_worth, mean_annual_leisure, label=f'Cluster {cluster_label}',
```

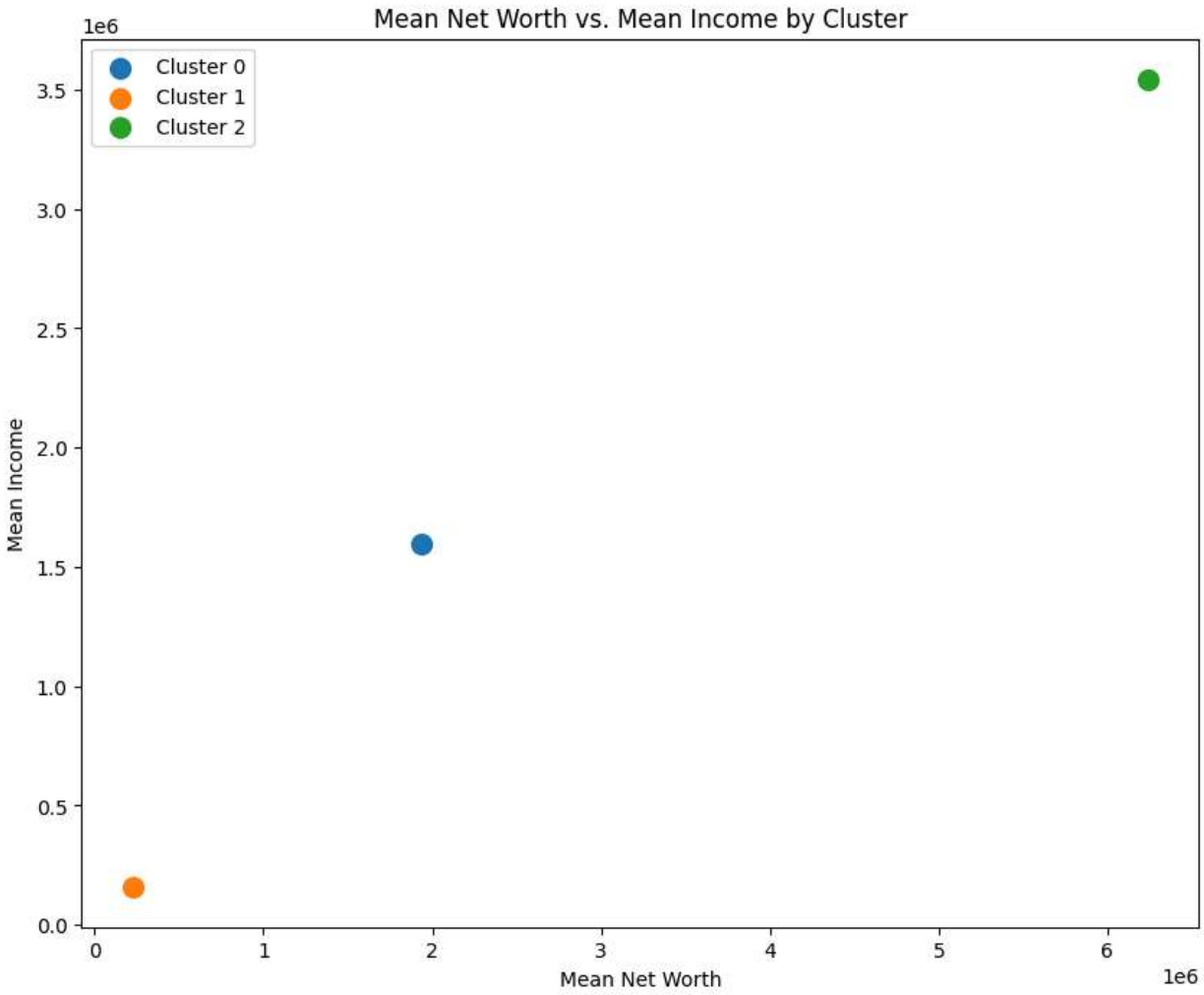
```
plt.title("Comparison of Clusters: Mean Net Worth vs. Mean Annual Leisure Spending")
plt.xlabel("Mean Net Worth")
plt.ylabel("Mean Annual Leisure Spending")
plt.legend()
plt.show()
```



In [ ]:
```
attributes_to_compare = ['est_net_worth', 'est_annual_leisure', 'est_annual_travel']
correlation_matrix = df_fam_kmeans[attributes_to_compare].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Heatmap of Mean Attribute Pair Correlation by Cluster")
plt.show()
```



In [ ]:
```
plt.figure(figsize=(10, 8))

for cluster_label in range(3):
    cluster_data = df_fam_norm[df_fam_norm['cluster'] == cluster_label]
    plt.scatter(cluster_data['est_net_worth'].mean(), cluster_data['est_inc'].mean(),

plt.title("Mean Net Worth vs. Mean Income by Cluster")
plt.xlabel("Mean Net Worth")
plt.ylabel("Mean Income")
plt.legend()
plt.show()
```



Cluster 0: Standard - Disney Parks & Resorts vacationers that fall within the '*Standard*' cluster are the vacationers with low to mid-range net worths and household incomes relative to the other two clusters. Vacationers within this cluster spend moderately on leisure and significantly on travel. Standard vacationers spend comparably to vacationers in cluster 1 in regards to online merchandise. Roughly 70% of vacationers in this cluster are from the United States. Eighty-one percent of vacationers in this cluster are also Disney+ subscribers.

To engage this cluster, Disney could offer customizable vacation packages. By offering customizable vacation packages Disney is offering prospective vacationers with the ability to experience Disney in a way that suits their unique style and budget.

Cluster 1: Economical - Disney Parks & Resorts vacationers that fall within the '*Economical*' cluster are the vacationers that spend the least on travel, leisure and online merchandise. Vacationers within this cluster have the lowest net worth and household income relative to the other two clusters which likely influences their spending, or lack thereof, on travel, leisure and online merchandise. Roughly 73% of vacationers in this cluster are from the United States. Sixty-two percent of vacationers in this cluster are also Disney+ subscribers.

To engage this cluster, Disney could offer discounted rates on park admission and hotel accommodations during 'off-season' times. This approach could incentive vacationers in this

cluster to visit Disney Parks & Resorts and increases attendance for Disney during periods of time when attendance typically lulls.

Cluster 2: Luxury - Disney Parks & Resorts vacationers that fall within the '*Luxury*' cluster are the vacationers with the highest net worth and household income relative to the other two clusters. Vacationers within this cluster spend the most on leisure and visit the parks and resorts at a slightly higher frequency than the other clusters. Luxury vacationers spend comparably to vacationers in cluster 0 in regards to online merchandise, but slightly less on travel. Roughly 73% of vacationers in this cluster are from the United States. Eighty percent of vacationers in this cluster are also Disney+ subscribers.

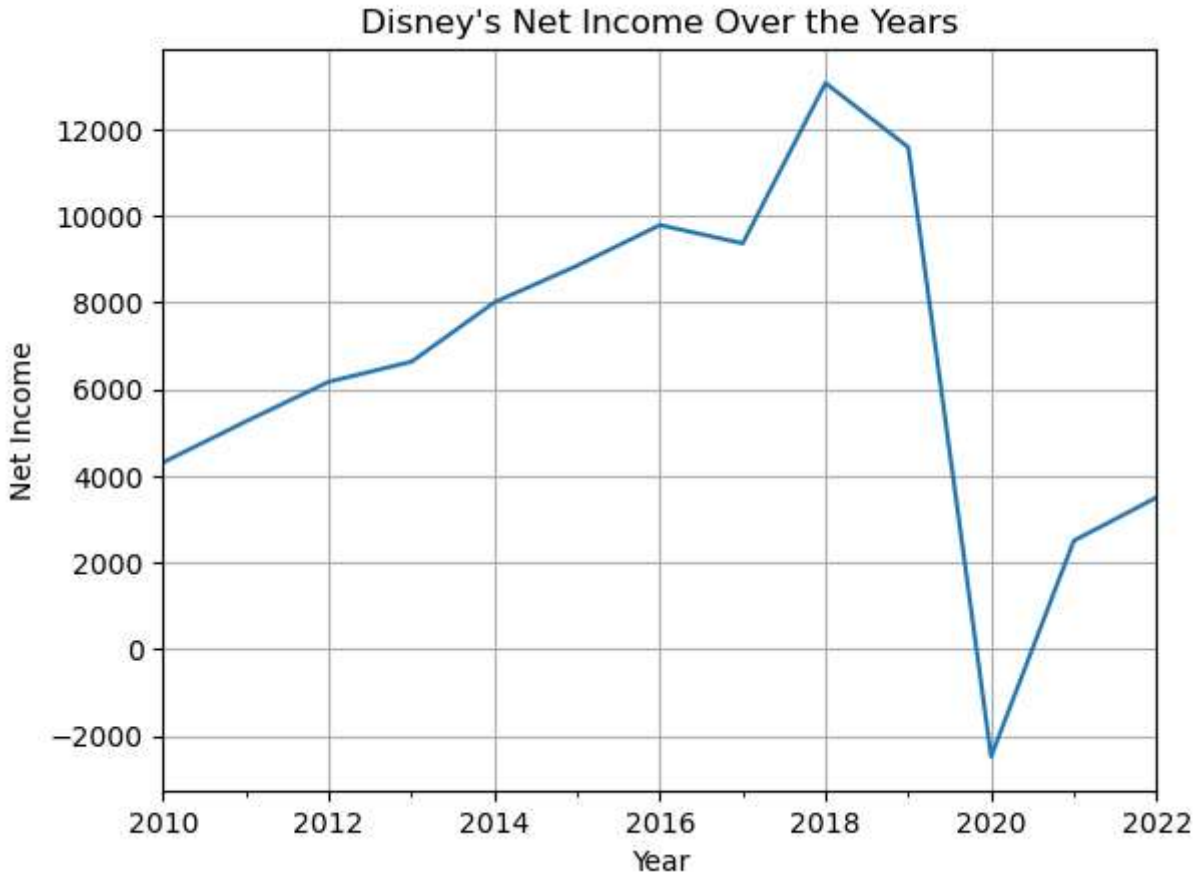
To engage this cluster, Disney could offer packages that increase the quality of the experience at their parks and resorts. Vacationers in this cluster value leisure and have a demonstrated willingness to spend money to achieve it. An example of a package disney could offer is a VIP-style park package. Disney could provide visitors with knowledgeable guides to explore the park with, door-to-door private transportation services, access to personalized perks, etc.

In [ ]:

In [ ]:
```
import pandas as pd
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
import matplotlib.pyplot as plt
```

In [ ]:
```
# Walt Disney historical net income data
years = pd.date_range(start='2010', end='2023', freq='A')
net_income = [4313, 5258, 6173, 6636, 8004, 8852, 9790, 9366, 13066, 11584, -2474, 250
data = pd.Series(net_income, index=years)
```

In [ ]:
```
# Plotting
data.plot()
plt.title("Disney's Net Income Over the Years")
plt.xlabel('Year')
plt.ylabel('Net Income')
plt.grid(True)
```



In [ ]:
```
# alpha = 0.2
ses1 = SimpleExpSmoothing(data)
model1 = ses.fit(smoothing_level = 0.2, optimized = False)

forecast1 = model1.forecast(1)


# alpha = 0.4
ses2 = SimpleExpSmoothing(data)
model2 = ses.fit(smoothing_level = 0.4, optimized = False)

forecast2 = model2.forecast(1)


# alpha = 0.6
ses3 = SimpleExpSmoothing(data)
model3 = ses.fit(smoothing_level = 0.6, optimized = False)

forecast3 = model3.forecast(1)


# alpha = 0.8
ses4 = SimpleExpSmoothing(data)
model4 = ses.fit(smoothing_level = 0.8, optimized = False)

forecast4 = model4.forecast(1)
```

```
2023-12-31    5506.370701
Freq: A-DEC, dtype: float64
2023-12-31    4011.241679
Freq: A-DEC, dtype: float64
2023-12-31    3207.785945
Freq: A-DEC, dtype: float64
2023-12-31    3219.823388
Freq: A-DEC, dtype: float64
```

In [ ]:
```
forecast_table = pd.DataFrame({'Alpha':[0.2,0.4,0.6,0.8],
                               '2023 Forecast':[forecast1[0],forecast2[0],forecast3[0],
print(forecast_table)
```

```
   Alpha  2023 Forecast
0    0.2    5506.370701
1    0.4    4011.241679
2    0.6    3207.785945
3    0.8    3219.823388
```

We obtained Walt Disney's historical net revenue data from 2010 through 2022 through an online search and integrated it into our data framework. By visualizing the time series graph, we found no significant seasonality or trends. Therefore, we chose Simple Exponential Smoothing (SES) as our forecasting method. In addition, there is a clear upward trend from 2010 to 2018, followed by a huge decline to 2020. This change suggests that methods that emphasize recent data, such as SES, may be more relevant. We used four different alpha levels to forecast net income in 2023. Larger alpha levels mean that more recent years of data influence the forecast more. The table above shows the results of these forecasts.

In [ ]:
```
import pandas as pd
```

In [ ]:
```
h_df = pd.read_csv('/Users/jacobyin16/Desktop/AD654/hotel.csv')
a_df = pd.read_csv('/Users/jacobyin16/Desktop/AD654/amentity.csv')
```

In [ ]:
```
h_df
```

Out[ ]:

	WiFi_Network	breakfast	parking	gym	flex_check	shuttle_bus	air_pure	jacuzzi	VIP_shop
0	Basic	None	Valet	None	No	No	No	No	No
1	Basic	None	Valet	None	No	No	No	No	No
2	Basic	None	Valet	None	No	No	No	No	No
3	Basic	None	Valet	None	No	No	No	No	Yes
4	Basic	None	Valet	None	No	No	No	No	Yes
...	...	...	...	...	...	...	...	...	...
6907	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	No
6908	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	No
6909	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	Yes
6910	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	Yes
6911	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	Yes

6912 rows × 11 columns

In [ ]:
```
a_df
```

Out[ ]:

	Amenity	Level	Estimated Incremental Cost,\nPer Visitor/Per Night
0	WiFi_Network	Basic	11.75
1	WiFi_Network	Strong	16.25
2	WiFi_Network	Best in Class	19.15
3	breakfast	None	0.00
4	breakfast	Continental	13.25
5	breakfast	Full Buffet	22.45
6	parking	Valet	60.00
7	parking	Open Lot	15.00
8	gym	None	0.00
9	gym	Basic	10.00
10	gym	Advanced	35.00
11	gym	Super	65.00
12	flex_check	No	0.00
13	flex_check	Yes	12.00
14	shuttle_bus	No	0.00
15	shuttle_bus	Yes	75.00
16	air_pure	No	0.00
17	air_pure	Yes	12.85
18	jacuzzi	No	0.00
19	jacuzzi	Yes	40.00
20	VIP_shop	No	0.00
21	VIP_shop	Yes	12.00
22	pool temp	76	15.00
23	pool temp	80	35.00
24	pool temp	84	45.00

In [ ]:
```
h_df.isna().sum()
```

Out[ ]:
```
WiFi_Network    0
breakfast       0
parking         0
gym             0
flex_check      0
shuttle_bus     0
air_pure        0
jacuzzi         0
VIP_shop        0
pool_temp       0
avg_rating      0
dtype: int64
```

In [ ]:
```
h_df.columns
```

Out[ ]:
```
Index(['WiFi_Network', 'breakfast', 'parking', 'gym', 'flex_check',
       'shuttle_bus', 'air_pure', 'jacuzzi', 'VIP_shop', 'pool_temp',
       'avg_rating'],
      dtype='object')
```

In [ ]:
```
h_df['WiFi_Network'].unique()
```

Out[ ]:
```
array(['Basic', 'Strong', 'Best in Class'], dtype=object)
```

In [ ]:
```
h_df2 = pd.get_dummies(h_df, columns = ['WiFi_Network', 'breakfast', 'parking', 'gym',
       'shuttle_bus', 'air_pure', 'jacuzzi', 'VIP_shop', 'pool_temp'], drop_first = Tr
```

In [ ]:
```
h_df2
```

Out[ ]:

	avg_rating	WiFi_Network_Best in Class	WiFi_Network_Strong	breakfast_Full Buffet	breakfast_None	parking_V
0	4.57	0	0	0	1	
1	7.60	0	0	0	1	
2	5.66	0	0	0	1	
3	2.80	0	0	0	1	
4	4.56	0	0	0	1	
...	...	...	...	...	...	
6907	8.21	1	0	1	0	
6908	8.21	1	0	1	0	
6909	8.21	1	0	1	0	
6910	8.21	1	0	1	0	
6911	8.21	1	0	1	0	

6912 rows × 16 columns

In [ ]:
```
h_df2.columns
```

Out[ ]:
```
Index(['avg_rating', 'WiFi_Network_Best in Class', 'WiFi_Network_Strong',
       'breakfast_Full Buffet', 'breakfast_None', 'parking_Valet', 'gym_Basic',
       'gym_None', 'gym_Super', 'flex_check_Yes', 'shuttle_bus_Yes',
       'air_pure_Yes', 'jacuzzi_Yes', 'VIP_shop_Yes', 'pool_temp_80',
       'pool_temp_84'],
      dtype='object')
```

In [ ]:
```
X =h_df2[['WiFi_Network_Best in Class', 'WiFi_Network_Strong',
       'breakfast_Full Buffet', 'breakfast_None', 'parking_Valet', 'gym_Basic',
       'gym_None', 'gym_Super', 'flex_check_Yes', 'shuttle_bus_Yes',
       'air_pure_Yes', 'jacuzzi_Yes', 'VIP_shop_Yes', 'pool_temp_80',
       'pool_temp_84']]
y = h_df2['avg_rating']
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
regressor = LinearRegression()
regressor.fit(X, y)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
regressor.intercept_
```

Out[ ]:
```
5.527955729166588
```

In [ ]:
```
coef_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
coef_df
```

Out[ ]:

	Coefficient
WiFi_Network_Best in Class	1.726814
WiFi_Network_Strong	1.187700
breakfast_Full Buffet	0.500998
breakfast_None	-0.225924
parking_Valet	0.093678
gym_Basic	-0.146314
gym_None	-0.168484
gym_Super	0.044363
flex_check_Yes	0.478220
shuttle_bus_Yes	0.419939
air_pure_Yes	0.075258
jacuzzi_Yes	0.183909
VIP_shop_Yes	0.217925
pool_temp_80	0.074744
pool_temp_84	0.263806

In [ ]:
```
a_df
```

Out[ ]:

	Amenity	Level	Estimated Incremental Cost,\nPer Visitor/Per Night
0	WiFi_Network	Basic	11.75
1	WiFi_Network	Strong	16.25
2	WiFi_Network	Best in Class	19.15
3	breakfast	None	0.00
4	breakfast	Continental	13.25
5	breakfast	Full Buffet	22.45
6	parking	Valet	60.00
7	parking	Open Lot	15.00
8	gym	None	0.00
9	gym	Basic	10.00
10	gym	Advanced	35.00
11	gym	Super	65.00
12	flex_check	No	0.00
13	flex_check	Yes	12.00
14	shuttle_bus	No	0.00
15	shuttle_bus	Yes	75.00
16	air_pure	No	0.00
17	air_pure	Yes	12.85
18	jacuzzi	No	0.00
19	jacuzzi	Yes	40.00
20	VIP_shop	No	0.00
21	VIP_shop	Yes	12.00
22	pool temp	76	15.00
23	pool temp	80	35.00
24	pool temp	84	45.00

In [ ]:
```
h_df[h_df['jacuzzi']=='Yes']
```

Out[ ]:

	WiFi_Network	breakfast	parking	gym	flex_check	shuttle_bus	air_pure	jacuzzi	VIP_shop
6	Basic	None	Valet	None	No	No	No	Yes	No
7	Basic	None	Valet	None	No	No	No	Yes	No
8	Basic	None	Valet	None	No	No	No	Yes	No
9	Basic	None	Valet	None	No	No	No	Yes	Yes
10	Basic	None	Valet	None	No	No	No	Yes	Yes
...	...	...	...	...	...	...	...	...	...
6907	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	No
6908	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	No
6909	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	Yes
6910	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	Yes
6911	Best in Class	Full Buffet	Open Lot	Super	Yes	Yes	Yes	Yes	Yes

3456 rows × 11 columns

250 dollars per night for each room

According to the coef dataframe and amentities cost dataframe, I recommand Diseny Hotel to include the following amentites for their customers: Best In Class Wifi, Full Buffet Breakfeast, Open Lot Parking, Advanced Gym, Flex Check-In, Free Shuttle Bus, VIP Shop, and 84F Pool. All these amentities would cost 235.7 dollars each night per room.

In addition to suggesting the incorporation of certain amenities, there are several modifications that the hotel could consider for their current offerings. Based on the coefficient dataframe, it is evident that customers exhibit limited sensitivity towards the provision of air purifiers. Rather than offering an air purifier in the rooms, the hotel might contemplate alternative enhancements to augment the guest experience, such as larger televisions or adjustable beds. Furthermore, there appears to be minimal customer preference for valet parking. Discontinuing this service could potentially reduce operational costs. Conversely, the VIP shopping feature is highly favored by patrons. The hotel might capitalize on this preference by potentially housing upscale boutiques on the premises and generating additional revenue through sales commissions

In [ ]:

```
%cd /Users/claudiasmac/Desktop/BU ABA/AD 654/datasets

import numpy as np

import pandas as pd
```

```
import matplotlib as mpl

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import confusion_matrix

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
```

```
/Users/claudiasmac/Desktop/BU ABA/AD 654/datasets
```

Data Preparation

In [ ]:
```
cruise=pd.read_csv("cruise_returners.csv")
cruise.head()
```

Out[ ]:

	hhold_ID	discount_original	coupon_received	cabin_type	est_inc	season_of_year	ticket_spend
0	3598	1	1	Verandah	19280.985	Winter	2649
1	155	1	0	Concierge	825719.960	Spring	2941
2	353	1	0	Oceanview	3755246.459	Fall	2609
3	1474	1	1	Oceanview	96831.550	Winter	2367
4	3272	1	0	Verandah	11027.249	Spring	3021

In [ ]:
```
cruise.shape
```

Out[ ]:
```
(7500, 16)
```

In [ ]:
```
# check missing values
cruise.isnull().sum()
```

Out[ ]:
```
hhold_ID                0
discount_original       0
coupon_received         0
cabin_type              0
est_inc                 0
season_of_year          0
ticket_spend            0
cruise_theme         1250
incidental_spending     0
park_package            0
embark_port             0
res_to_port             0
number_children         0
oldest_child_age      370
dis_plus_sub            0
return_1065             0
dtype: int64
```

In [ ]:
```
print(cruise['cruise_theme'].unique())
```

```
['prin_fro' 'Star Wars' 'Marvel' nan 'starwars' 'Princess/Frozen']
```

In [ ]:
```
cruise[cruise['cruise_theme'].isnull()]
```

Out[ ]:

	hhold_ID	discount_original	coupon_received	cabin_type	est_inc	season_of_year	ticket_spe
9	3132	1	0	Inside	341862.729	Spring	21
16	1291	1	1	Oceanview	204182.628	Winter	19
17	2139	1	1	Oceanview	476286.319	Summer	18
26	5900	1	0	Inside	1799.388	Summer	25
27	6323	1	0	Oceanview	3732.670	Spring	21
...	...	...	...	...	...	...	
7476	7374	0	0	Concierge	137398.580	Fall	24
7481	7393	0	0	Oceanview	1428.722	Fall	27
7490	7444	0	0	Verandah	7287.179	Spring	19
7495	7464	0	0	Verandah	169648.634	Winter	20
7499	7499	0	0	Oceanview	33521.227	Winter	20

1250 rows × 16 columns

By looking at the records where 'cruise_theme' is NaN, we believe that the NaN is due to missing records, so we decided to delete those rows.

In [ ]:
```
cruise = cruise.dropna(subset=['cruise_theme'])
cruise.shape
```

Out[ ]:
```
(6250, 16)
```

In [ ]:
```
print(cruise['oldest_child_age'].unique())
```

```
[ 6.  8.  3.  7. 10.  5.  4. 12. 16. 15. 13. 11. 14.  9.  2. 18. 17. nan]
```

In [ ]:
```
cruise[cruise['oldest_child_age'].isnull()]
```

Out[ ]:

	hhold_ID	discount_original	coupon_received	cabin_type	est_inc	season_of_year	ticket_spe
307	2513	1	0	Inside	34337.100	Spring	15
319	3858	1	1	Concierge	3189.683	Spring	28
344	1816	1	0	Oceanview	99359.697	Winter	16
354	202	1	0	Verandah	765.008	Fall	25
385	2115	1	0	Inside	92.407	Summer	24
...	...	...	...	...	...	...	
7355	6754	1	0	Concierge	4185.057	Winter	31
7384	6922	0	1	Oceanview	73724.298	Spring	23
7400	7008	0	1	Concierge	243546.530	Winter	30
7402	7010	0	0	Concierge	58291.878	Fall	25
7494	7460	0	1	Verandah	141413.507	Spring	25

309 rows × 16 columns

In [ ]:
```
cruise[cruise['oldest_child_age'].isnull()]['number_children'].unique()
```

Out[ ]:
```
array([ 0, -1, -2])
```

By looking at records where "oldest_child_age" is NaN, we see that "number_children" is always 0, -1, or -2. Intuitively this makes sense since the family has no children, so the age of the oldest child may not known. We can assume these NAs to be 0.

In [ ]:
```
# Convert NAs in 'oldest_child_age' to be 0
cruise['oldest_child_age'].fillna(0, inplace=True)
print(cruise['oldest_child_age'].unique())
```

```
[ 6.  8.  3.  7. 10.  5.  4. 12. 16. 15. 13. 11. 14.  9.  2. 18. 17.  0.]
```

In [ ]:
```
cruise.describe()
```

Out[ ]:

	hhold_ID	discount_original	coupon_received	est_inc	ticket_spend	incidental_spendin
count	6250.000000	6250.000000	6250.000000	6.250000e+03	6250.000000	6250.00000
mean	3737.233600	0.385440	0.255040	5.716017e+05	2327.807840	7673.42866
std	2165.132722	0.486738	0.435919	1.424178e+06	567.144158	9702.15941
min	1.000000	0.000000	0.000000	9.070000e-01	1102.000000	18.26536
25%	1846.250000	0.000000	0.000000	7.164886e+03	1910.250000	2898.75434
50%	3738.500000	0.000000	0.000000	4.869115e+04	2325.000000	2907.97078
75%	5603.750000	1.000000	1.000000	3.286320e+05	2736.000000	9548.65287
max	7498.000000	1.000000	1.000000	9.938626e+06	3600.000000	69975.72106

Combined with data description and general knowledge, we found that 'number_children' should not be a negative number.

In [ ]:
```
cruise[cruise['number_children'] < 0].head()
```

Out[ ]:

	hhold_ID	discount_original	coupon_received	cabin_type	est_inc	season_of_year	ticket_spe
598	2083	1	0	Inside	608052.162	Spring	25
695	5366	1	1	Inside	613.695	Winter	25
861	2221	1	0	Inside	21462.565	Winter	14
1011	6432	1	0	Inside	23379.489	Fall	14
2919	2027	0	0	Verandah	277915.641	Winter	29

In [ ]:
```
cruise[cruise['number_children'] < 0]['oldest_child_age'].unique()
```

Out[ ]:
```
array([0.])
```

When "number_children" is negative, "oldest_child_age" is always 0. These negative numbers could be a type error, so we decided to change the negative numbers to 0. If the number of children is 0, the age of the oldest child will be 0 and vice versa.

In [ ]:
```
cruise.loc[cruise['number_children'] < 0, 'number_children'] = 0
cruise['number_children'].min()
```

Out[ ]:
```
0
```

Data Analysis and Visualization

Unique Identifier: hhold_ID

Numerical variables: est_inc, ticket_spend, incidental_spending, res_to_port, number_children, oldest_child_age

Categorical variables: discount_original, coupon_received, cabin_type, season_of_year, cruise_theme, park_package, embark_port, dis_plus_sub, return_1065

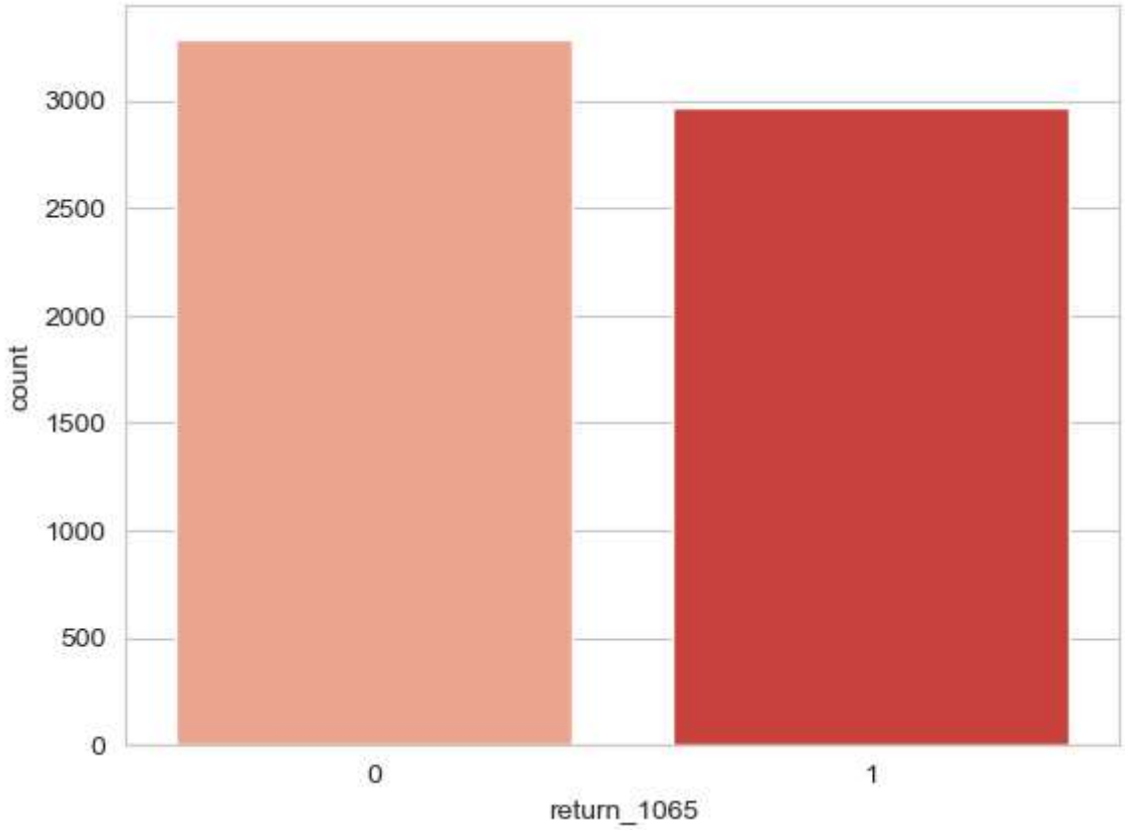
Since householdID is a unique identifier, we decided to remove it as it is not relevant to the results and also to avoid overfitting.

In [ ]:
```
cruise_clean = cruise.drop(columns = ['hhold_ID'])
cruise_clean.shape
```

Out[ ]:
```
(6250, 15)
```

In [ ]:
```
# Look at the distribution of outcome variable
sns.set_style('whitegrid')
sns.countplot(x='return_1065', data=cruise_clean, palette='Reds')
```

Out[ ]:
```
<AxesSubplot:xlabel='return_1065', ylabel='count'>
```

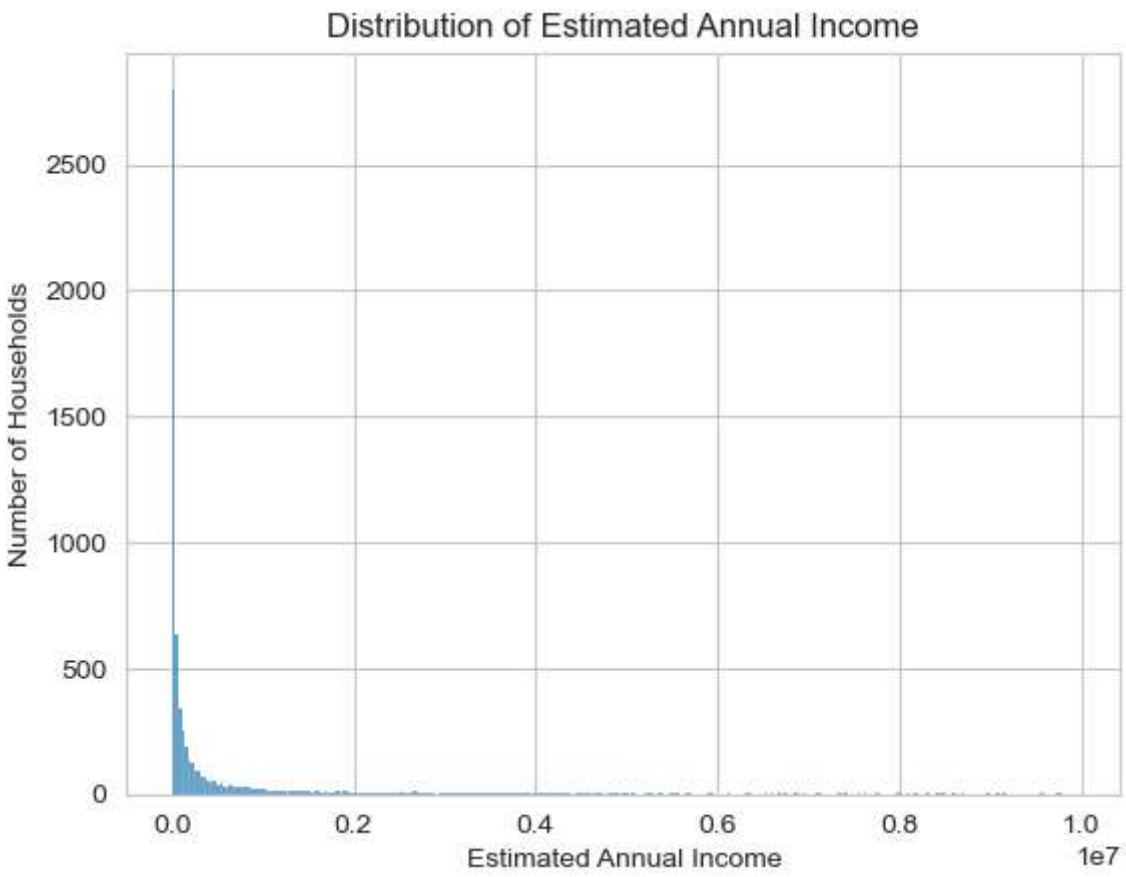


The dataset seems to be balanced.

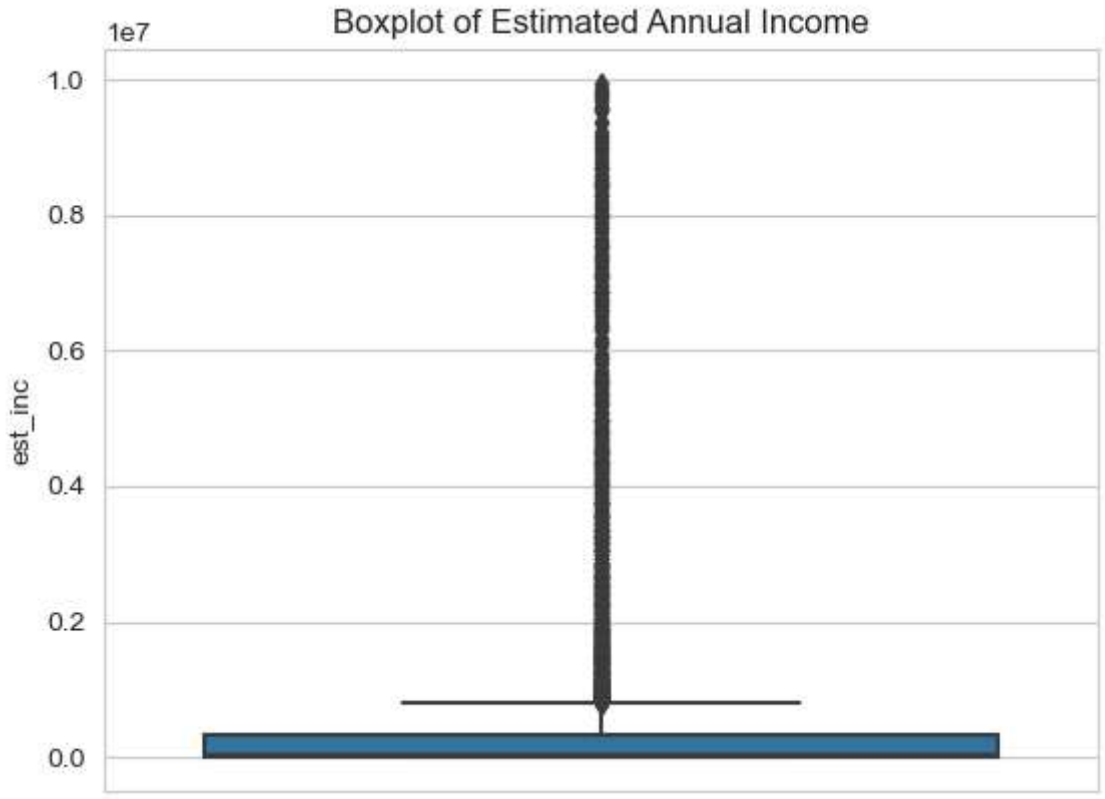
Visualizations of Numeric Variables

In [ ]:
```
# Distribution of Household Annual Income
sns.histplot(cruise_clean['est_inc'])
plt.title('Distribution of Estimated Annual Income')
plt.xlabel('Estimated Annual Income')
```

```
plt.ylabel('Number of Households')
plt.show()
```

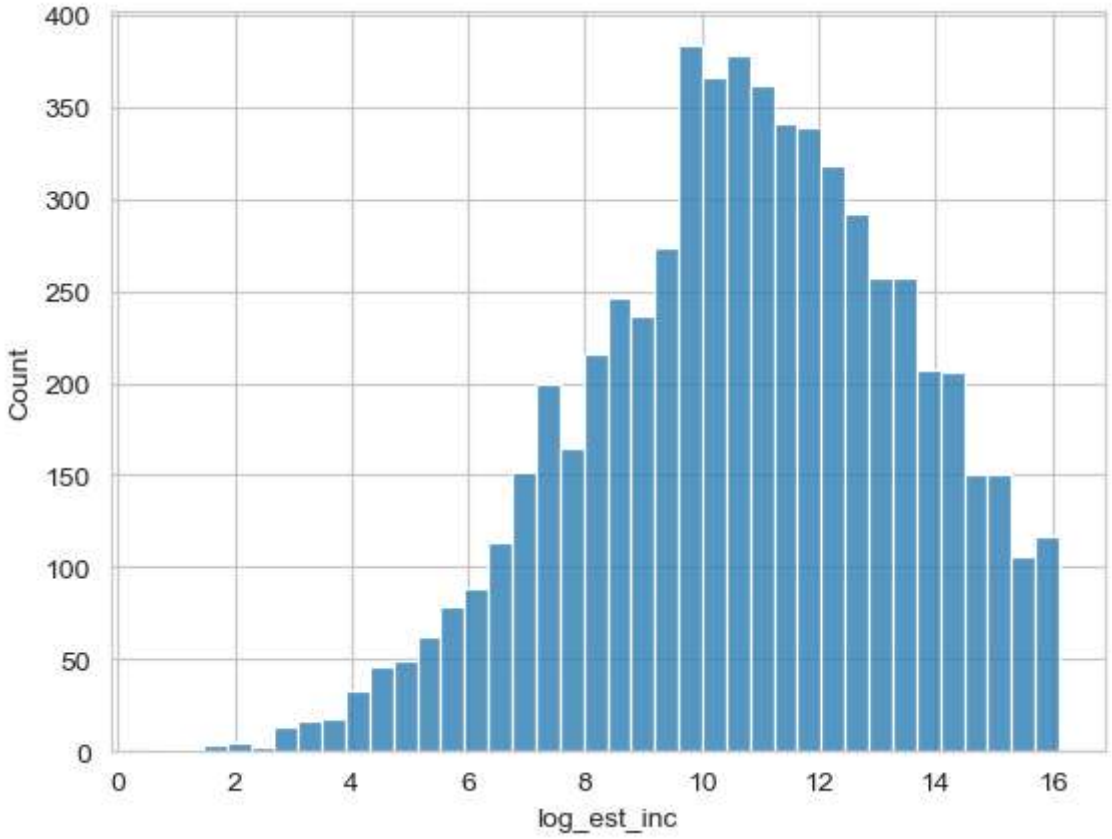


In [ ]:
```
sns.boxplot(y=cruise_clean['est_inc'])
plt.title('Boxplot of Estimated Annual Income')
plt.show()
```

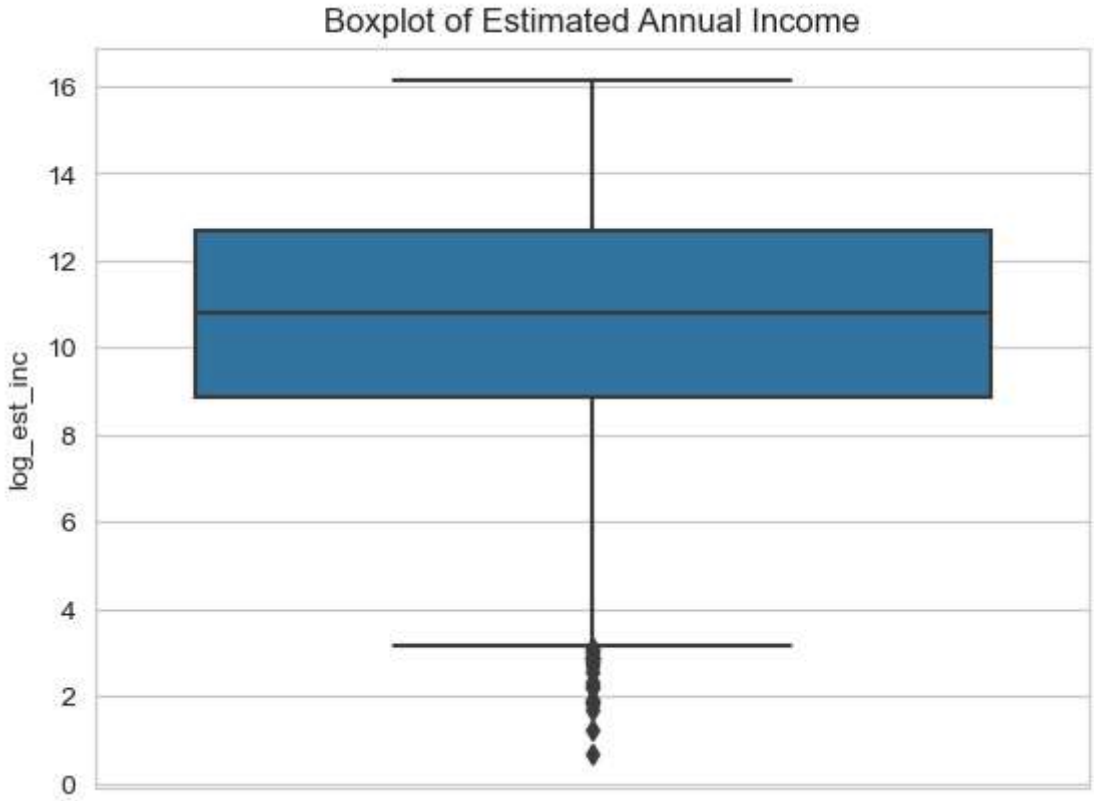


From the histogram of estimated household annual income, we see that it is right skewed and the range of the annual income is supper large which is not expected. The boxplot also shows a large amount outliers exist in the dataset. This might be that families are not telling their real household annual income, or units of record are not standardized. We expected the annual household income to be approximately normally distributed. So, we use log transformation to deal with the right skewed data.

In [ ]:
```
cruise_clean['log_est_inc'] = np.log(cruise_clean['est_inc'] + 1)  # Adding 1 to avoid
sns.histplot(cruise_clean['log_est_inc'])
plt.show()
```



In [ ]:
```
sns.boxplot(y=cruise_clean['log_est_inc'])
plt.title('Boxplot of Estimated Annual Income')
plt.show()
```

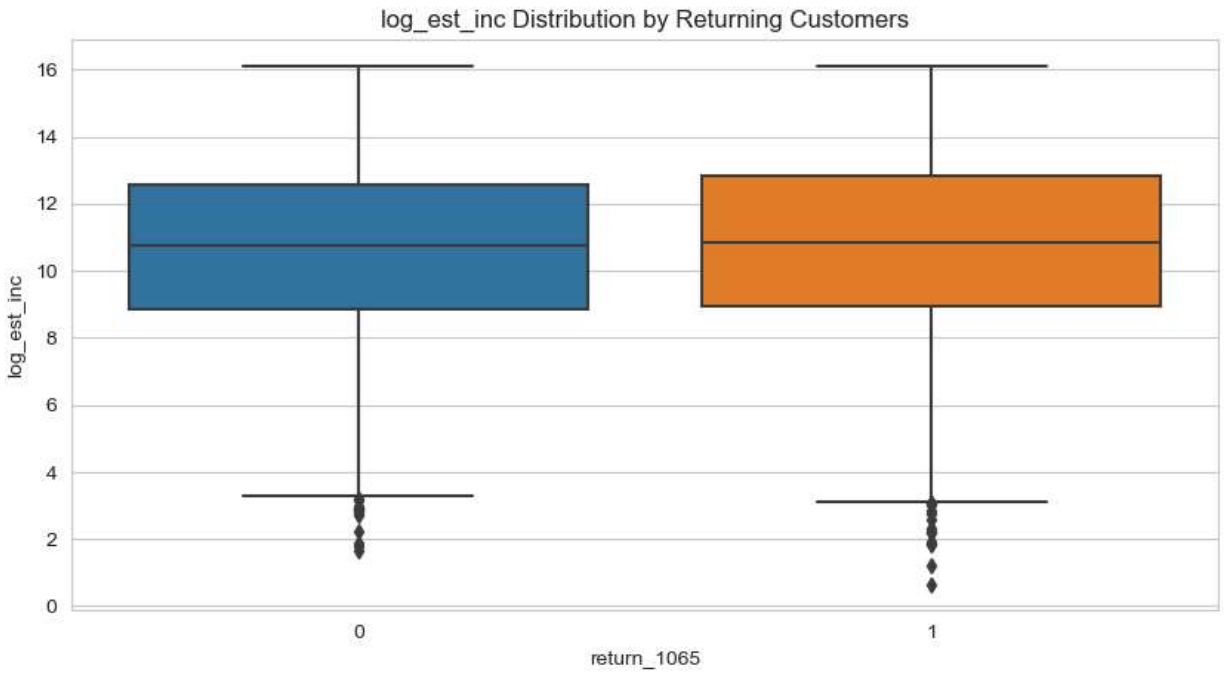


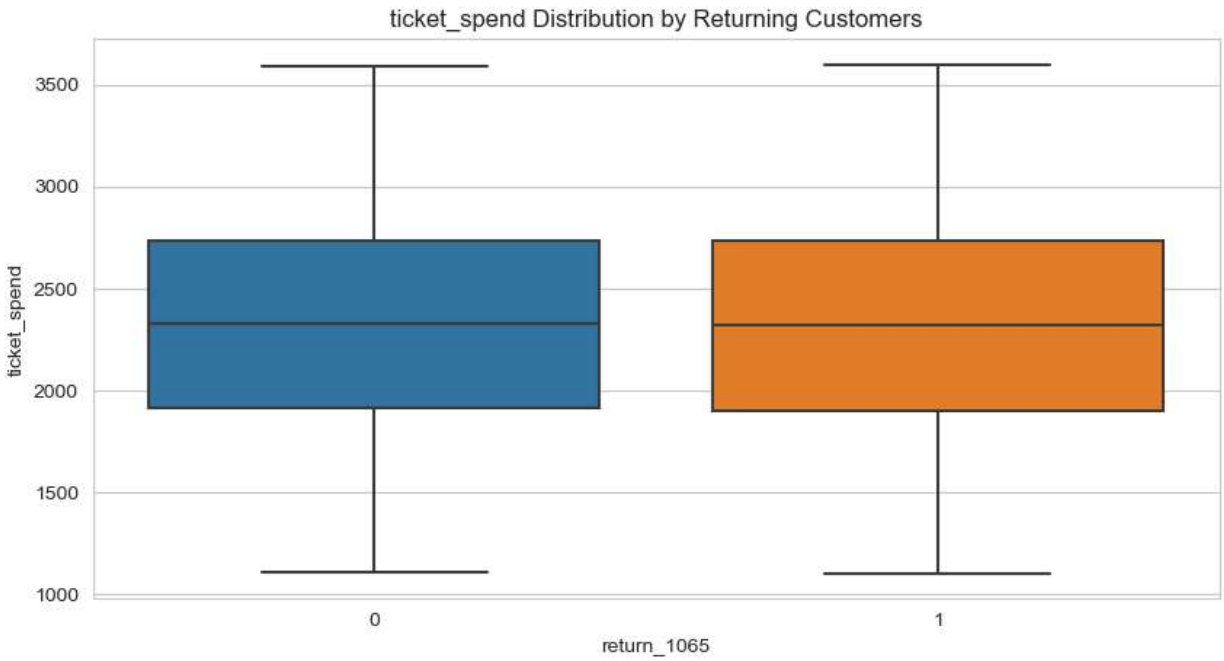
The problem of right-skewing annual household income is now solved. Let's assume that the unit is $10,000.

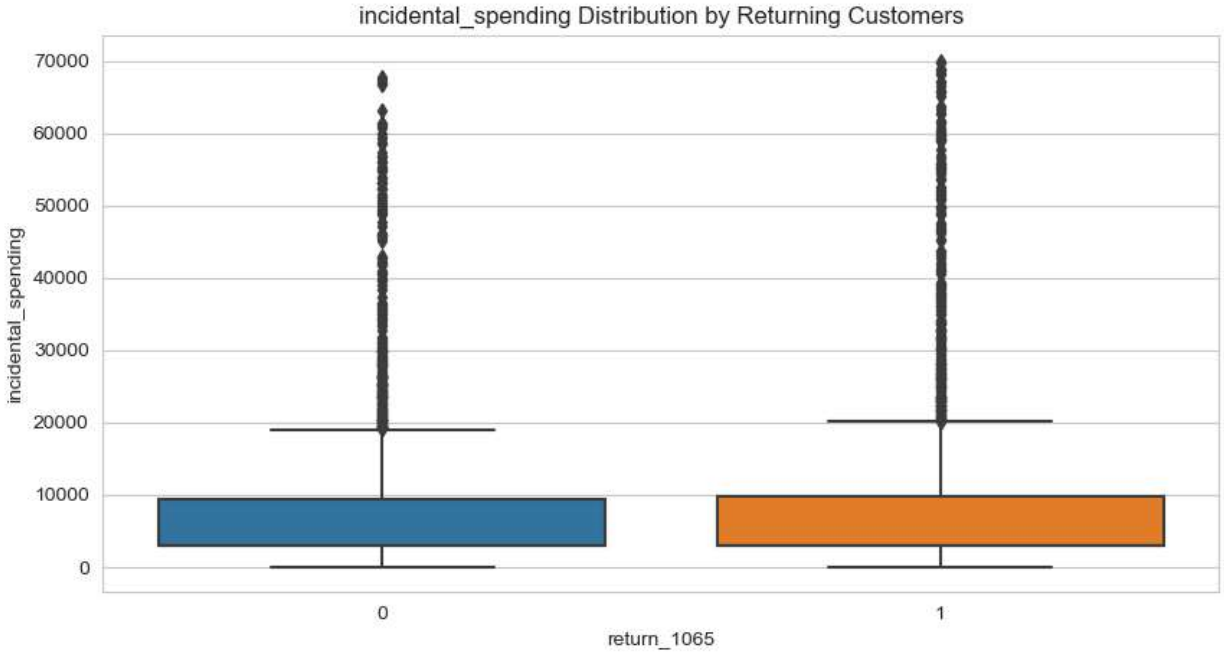
In [ ]:
```
cruise_clean = cruise_clean.drop(columns = ['est_inc'])
cruise_clean.columns
```

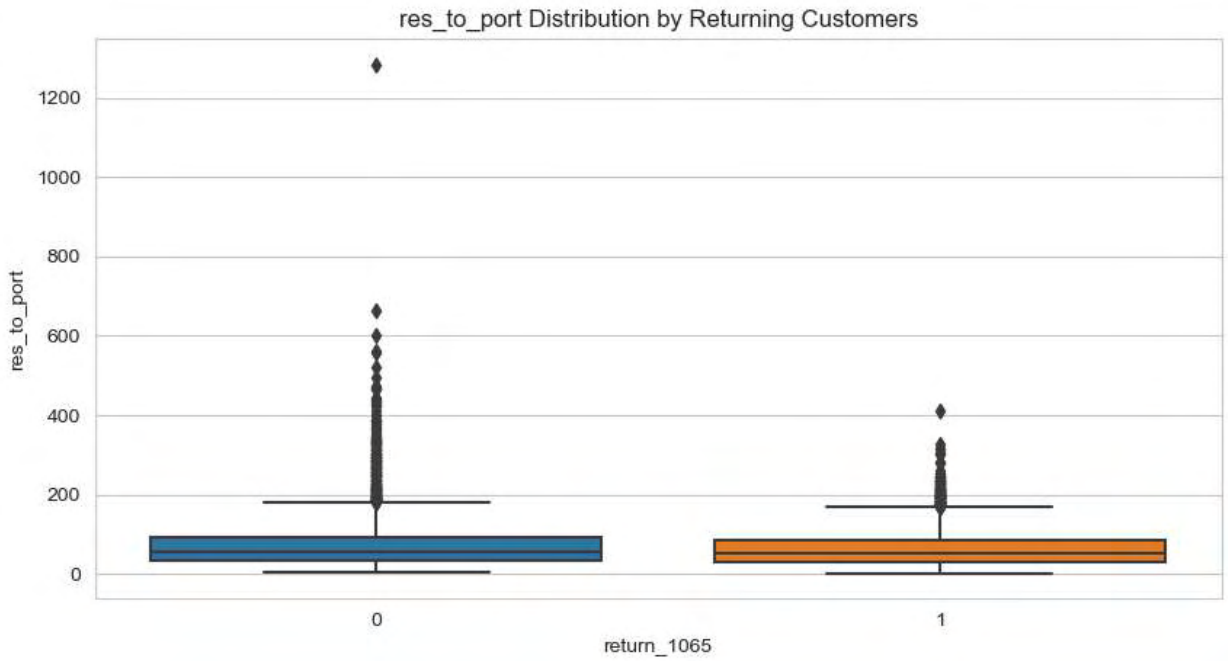
Out[ ]:
```
Index(['discount_original', 'coupon_received', 'cabin_type', 'season_of_year',
       'ticket_spend', 'cruise_theme', 'incidental_spending', 'park_package',
       'embark_port', 'res_to_port', 'number_children', 'oldest_child_age',
       'dis_plus_sub', 'return_1065', 'log_est_inc'],
      dtype='object')
```

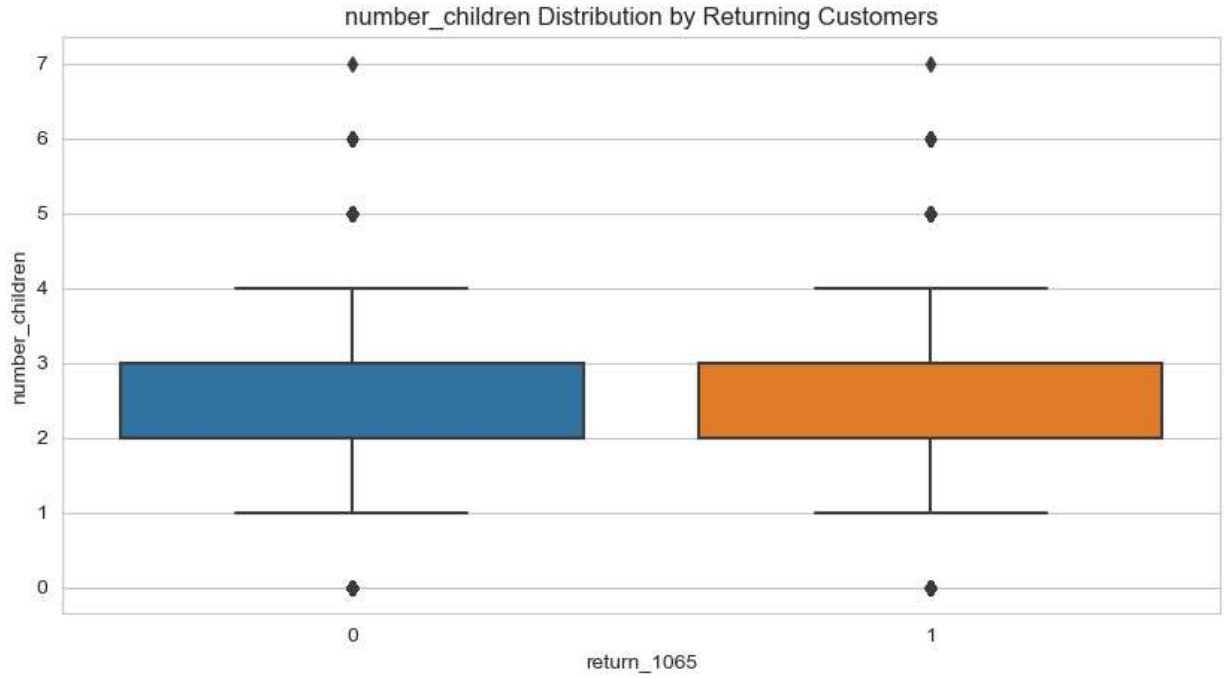
In [ ]:
```
numeric_vars = ['log_est_inc','ticket_spend', 'incidental_spending',
                'res_to_port', 'number_children', 'oldest_child_age']

for var in numeric_vars:
    plt.figure(figsize=(10,5))
    sns.boxplot(x='return_1065', y=var, data=cruise_clean)
    plt.title(f'{var} Distribution by Returning Customers')
    plt.show()
```

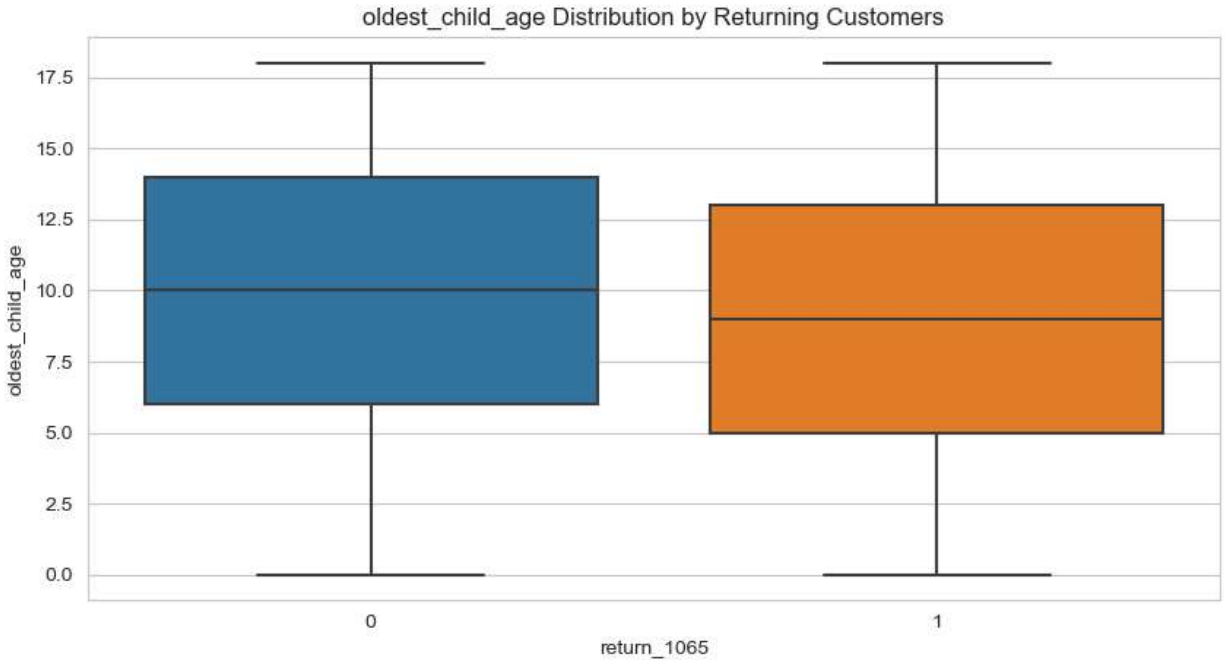



ticket_spend Distribution by Returning Customers
ticket_spend
3500
3000
2500
2000
1500
1000
0
1
return_1065


incidental_spending Distribution by Returning Customers
incidental_spending
70000
60000
50000
40000
30000
20000
10000
0
0
1
return_1065


res_to_port Distribution by Returning Customers
res_to_port
1200
1000
800
600
400
200
0
0
1
return_1065

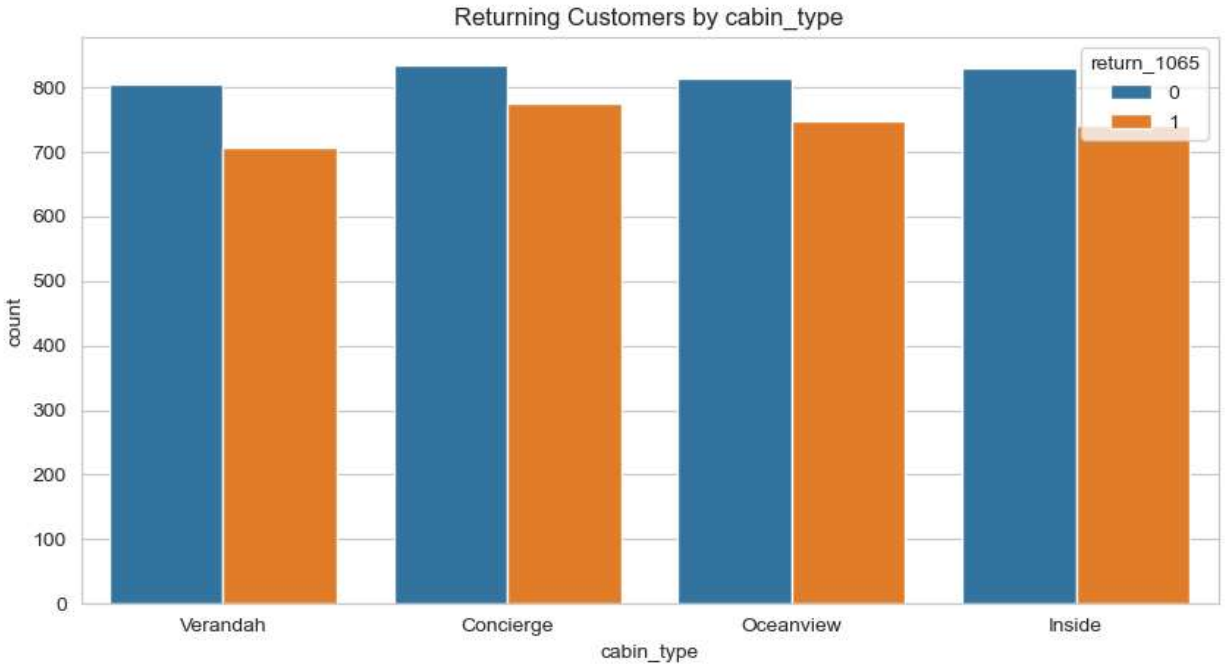

number_children Distribution by Returning Customers
number_children
7
6
5
4
3
2
1
0
0
1
return_1065

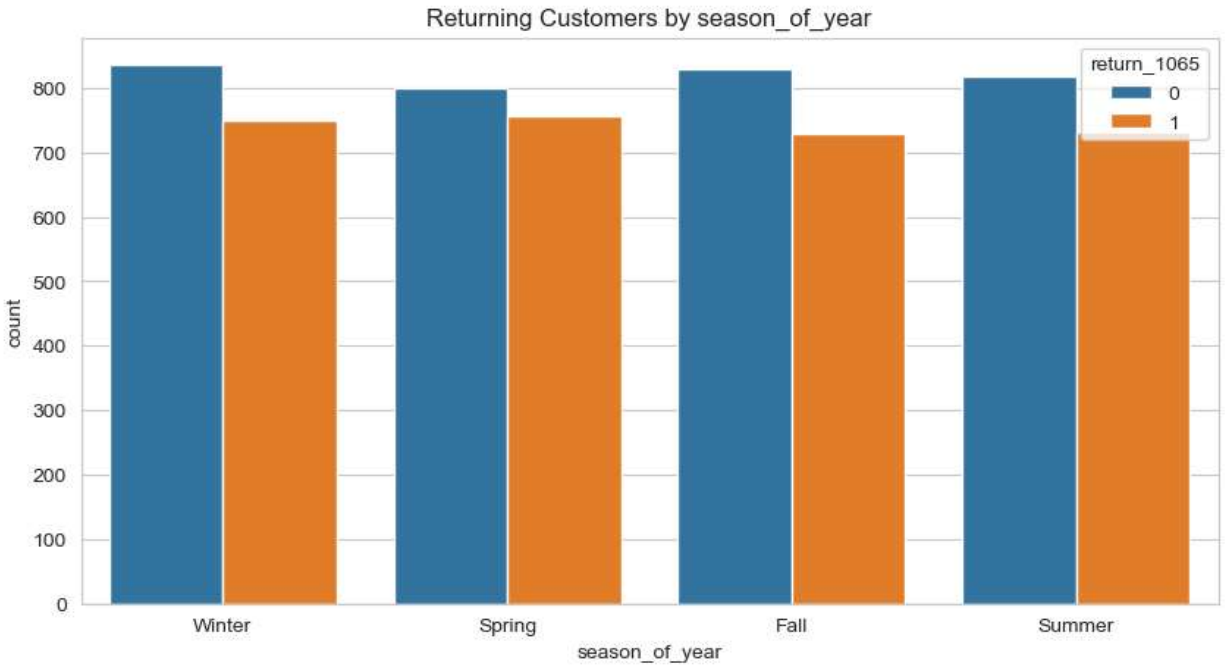


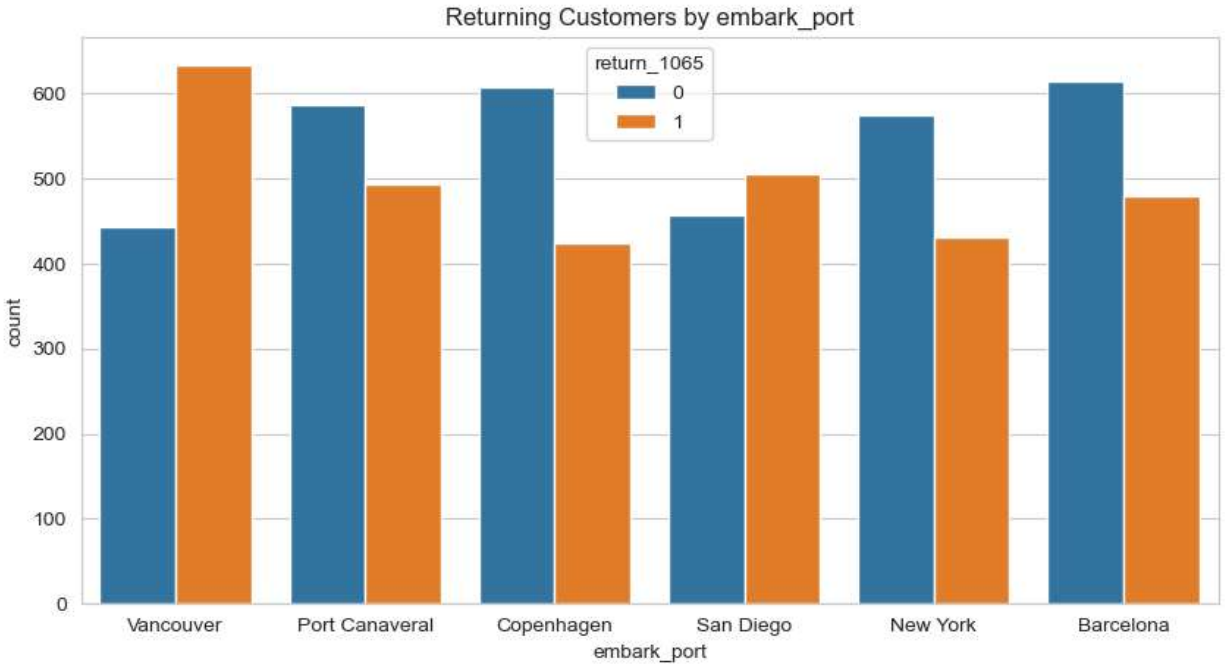
From the box plot, we can see that incidental_spending is also positively skewed, but we'll leave it alone. This is because the data description mentions that some cruise travelers spend a lot of money on gambling, drinking, shopping, and even purchasing subsequent vacations on their cruises, which explains why there are many large outliers.

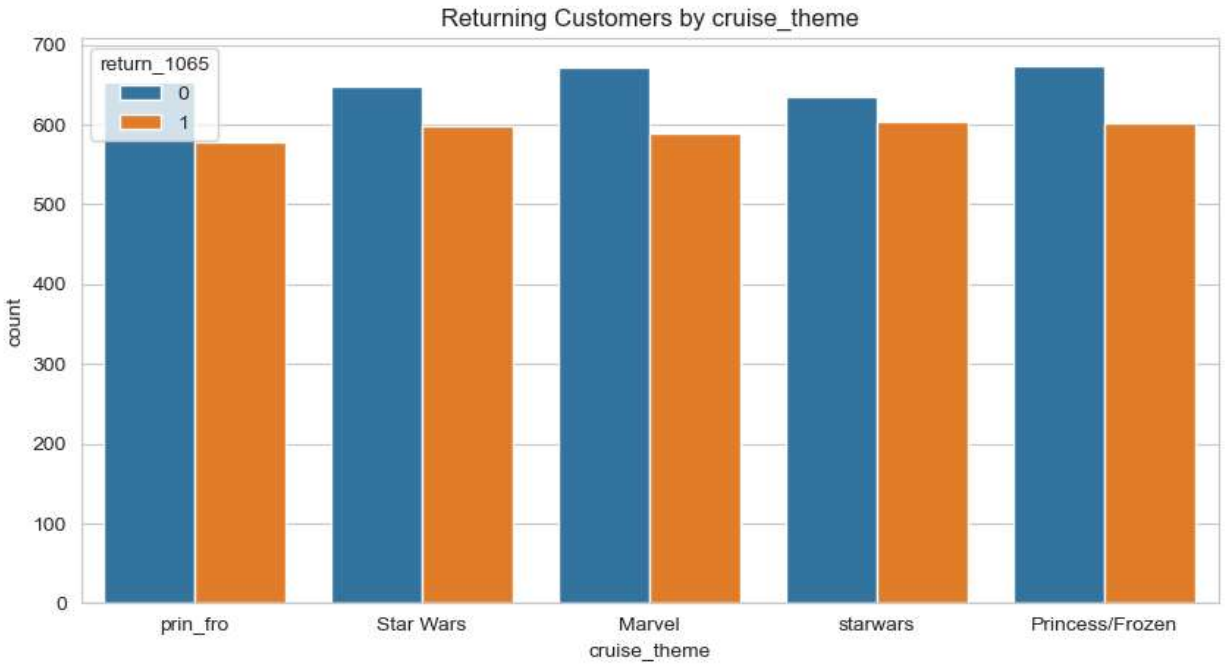
Visualizations of Categorical Variables

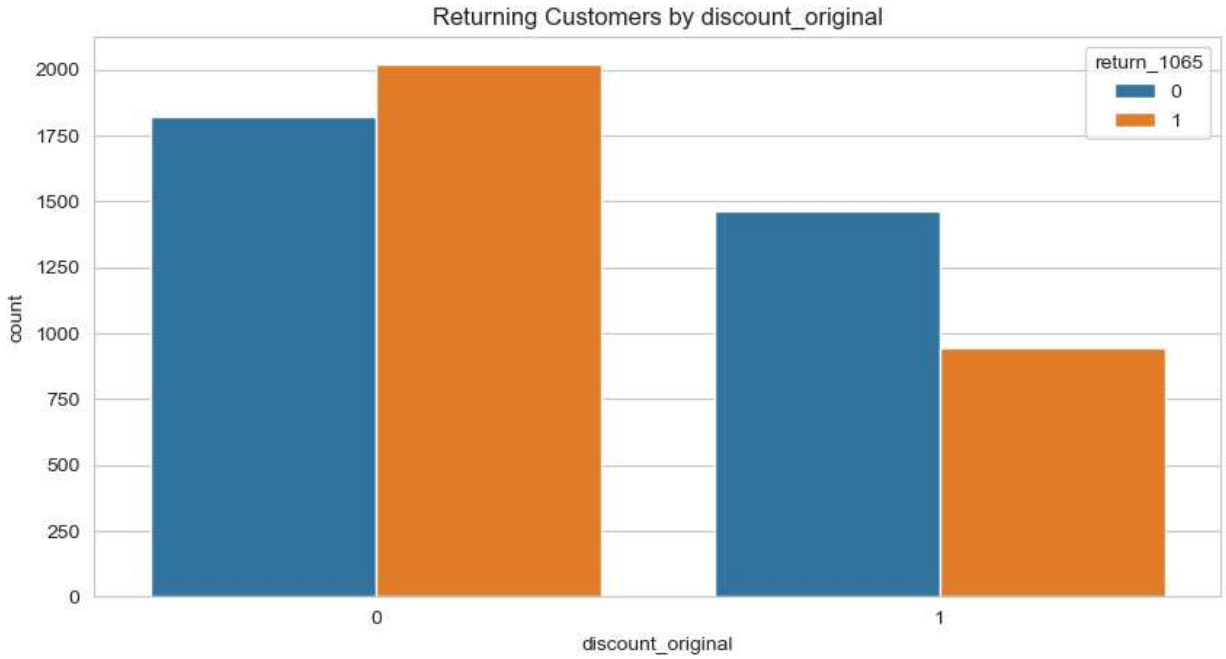
In [ ]:
```
categorical_vars = ['cabin_type', 'season_of_year', 'embark_port', 'cruise_theme',
                    'discount_original', 'coupon_received', 'park_package', 'dis_plus_

for var in categorical_vars:
    plt.figure(figsize=(10,5))
    sns.countplot(x=var, hue='return_1065', data=cruise_clean)
    plt.title(f'Returning Customers by {var}')
    plt.show()
```


Returning Customers by cabin_type
return_1065
0
1
count
0
100
200
300
400
500
600
700
800
Verandah
Concierge
Oceanview
Inside
cabin_type

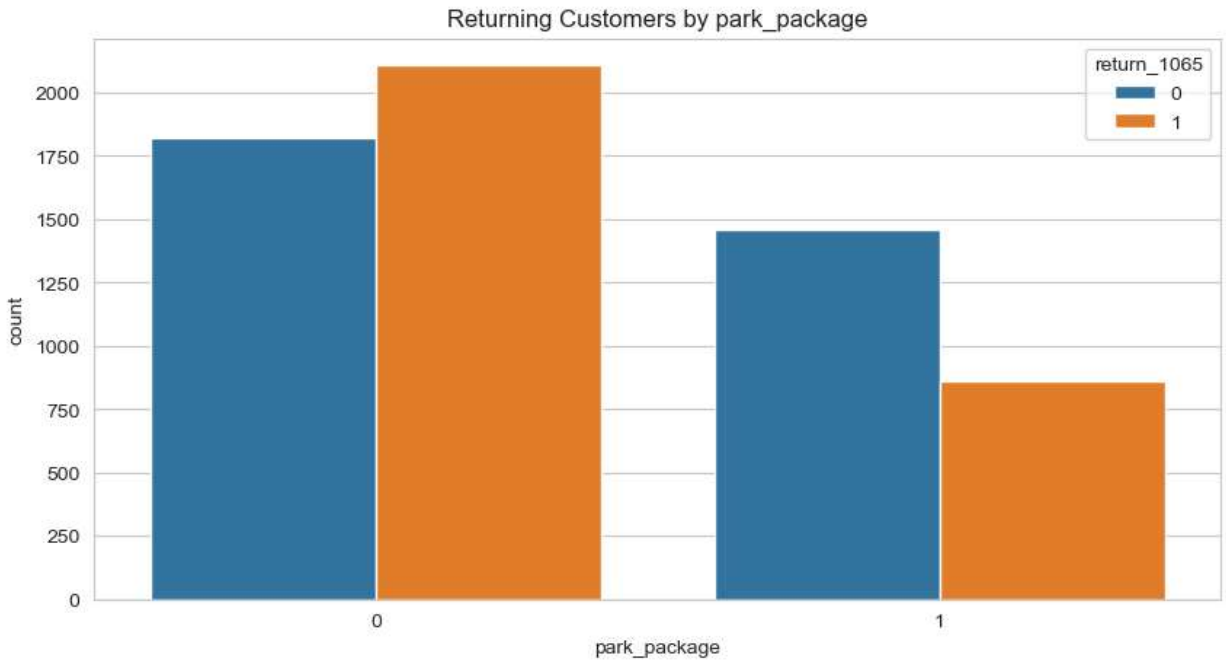

Returning Customers by season_of_year
return_1065
0
1
count
0
100
200
300
400
500
600
700
800
Winter
Spring
Fall
Summer
season_of_year

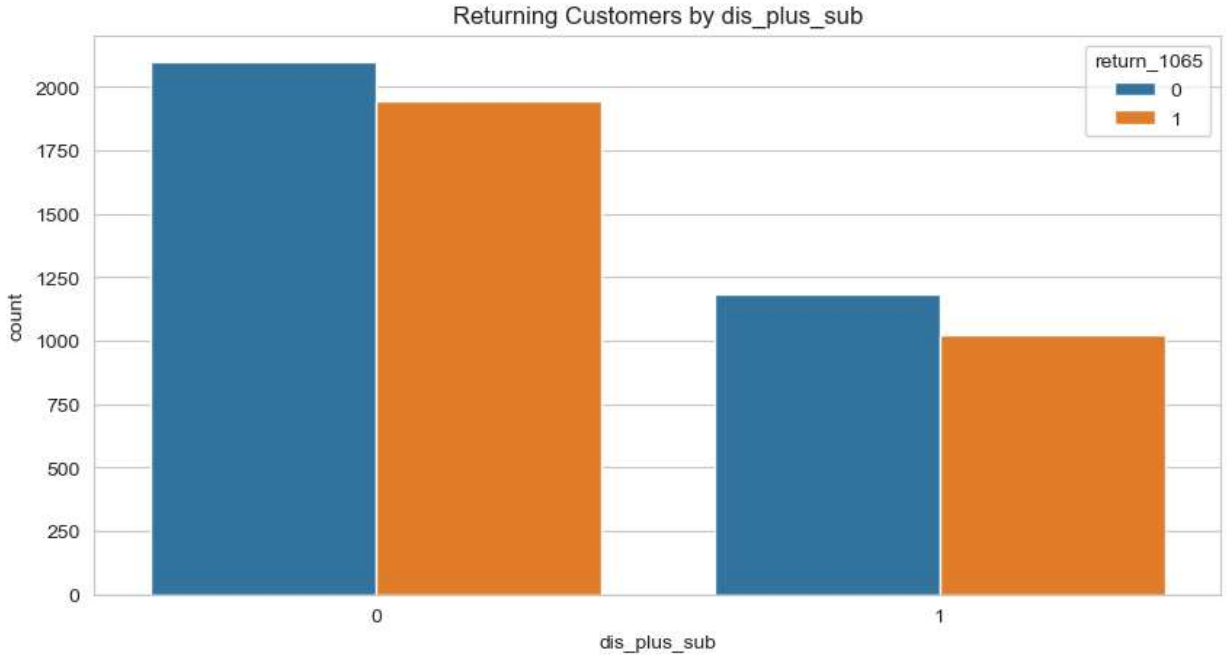

Returning Customers by embark_port
return_1065
0
1
count
0
100
200
300
400
500
600
Vancouver
Port Canaveral
Copenhagen
San Diego
New York
Barcelona
embark_port


Returning Customers by cruise_theme
return_1065
0
1
count
0
100
200
300
400
500
600
700
prin_fro
Star Wars
Marvel
starwars
Princess/Frozen
cruise_theme


Returning Customers by discount_original
return_1065
0
1
count
2000
1750
1500
1250
1000
750
500
250
0
0
1
discount_original


Returning Customers by coupon_received
return_1065
0
1
count
2500
2000
1500
1000
500
0
0
1
coupon_received





The above barplots give us some insights.

Customers showed a preference for cruises departing from Vancouver, as evidenced by the high repeat booking rate (return_1065 value of 1). In contrast, cruises departing from Barcelona have a significantly lower repeat booking rate, suggesting that Barcelona is less favored by customers.

Customers who received a discount on their initial cruise have a lower number of "1's" in return_1065, suggesting that these customers who received a discount are less likely to book a subsequent cruise within 3 years than those who did not receive a discount. This may be because discounts may attract one-time customers who are not necessarily interested in the cruise experience itself. If customers receive a discounted experience, they may perceive the

value of the cruise differently than customers who pay full price. Once the original price is restored, they become reluctant to make a purchase.

As can be seen from the bar charts of coupons and park packages received, fewer families who received coupons or booked with discounted park packages had both repeat and non-repeat customers. As a result, cruise lines may need to reconsider their discount and coupon strategies. While discounts and coupons can increase immediate sales, they do not necessarily lead to long-term customer loyalty or repeat business.

Similarly, a Disney Plus subscription does not mean that the number of people returning to cruise will increase. One might think that families who subscribe to a streaming service to invest in the Disney brand would also show loyalty to Disney cruises. But that's not the case, and streaming subscriptions may cater to those who are less of a cruise ship fan. The data suggests that being associated with only one Disney product does not necessarily guarantee engagement across all segments.

Data Prep - Categorical Data

In [ ]:
```
cruise_dummies = pd.get_dummies(cruise_clean, drop_first = False, columns = categorica
cruise_dummies.head()
```

Out[ ]:

	ticket_spend	incidental_spending	res_to_port	number_children	oldest_child_age	return_1065	log_
0	2649	2898.754343	558	3	6.0	0	9.
1	2941	2898.754343	280	3	8.0	0	13.
2	2609	11750.829364	351	3	3.0	0	15.
3	2367	13945.343512	314	3	7.0	0	11.
4	3021	2898.754343	342	4	10.0	0	9.

5 rows × 34 columns

In [ ]:
```
cruise_dummies.columns
```

Out[ ]:
```
Index(['ticket_spend', 'incidental_spending', 'res_to_port', 'number_children',
       'oldest_child_age', 'return_1065', 'log_est_inc',
       'cabin_type_Concierge', 'cabin_type_Inside', 'cabin_type_Oceanview',
       'cabin_type_Verandah', 'season_of_year_Fall', 'season_of_year_Spring',
       'season_of_year_Summer', 'season_of_year_Winter',
       'embark_port_Barcelona', 'embark_port_Copenhagen',
       'embark_port_New York', 'embark_port_Port Canaveral',
       'embark_port_San Diego', 'embark_port_Vancouver', 'cruise_theme_Marvel',
       'cruise_theme_Princess/Frozen', 'cruise_theme_Star Wars',
       'cruise_theme_prin_fro', 'cruise_theme_starwars', 'discount_original_0',
       'discount_original_1', 'coupon_received_0', 'coupon_received_1',
       'park_package_0', 'park_package_1', 'dis_plus_sub_0', 'dis_plus_sub_1'],
      dtype='object')
```

Random Forest

In [ ]:
```
from sklearn.model_selection import train_test_split
X = cruise_dummies.drop('return_1065', axis=1)
y = cruise_dummies['return_1065'] # predictor
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y, test_size=0.30,
                                                    random_state=654)
print(f'X_train : {X_train.shape}')
print(f'y_train : {y_train.shape}')
print(f'X_test : {X_test.shape}')
print(f'y_test : {y_test.shape}')
```

```
X_train : (4375, 33)
y_train : (4375,)
X_test : (1875, 33)
y_test : (1875,)
```

In [ ]:
```
rf = RandomForestClassifier(n_estimators=20,random_state=654)
rf.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier(n_estimators=20, random_state=654)
```

In [ ]:
```
rf_predictions = rf.predict(X_test)
print(classification_report(y_test, rf_predictions))
```

```
              precision    recall  f1-score   support

           0       0.65      0.68      0.66      1013
           1       0.60      0.58      0.59       862

    accuracy                           0.63      1875
   macro avg       0.63      0.63      0.63      1875
weighted avg       0.63      0.63      0.63      1875
```

Now, we use hyperparameter tuning to improve the performance of the random forest model.

In [ ]:
```
# define the hyperparameters
param_grid = {
    'n_estimators': [20,50,80,100],
    'max_features': ['auto', 'sqrt'],
    'max_depth' : [2,3,4,5],
    'criterion' :['gini', 'entropy']
}


rf_fi = RandomForestClassifier()

gscv= GridSearchCV(estimator=rf_fi, param_grid=param_grid,
                           cv=5, refit=True, scoring='accuracy')

gscv.fit(X_train, y_train)
```

Out[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 3, 4, 5],
                         'max_features': ['auto', 'sqrt'],
                         'n_estimators': [20, 50, 80, 100]},
             scoring='accuracy')
```

In [ ]:
```
best_params = gscv.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt',
'n_estimators': 100}
```

In [ ]:
```
# rebuild the model using the best hyperparameters
rf_best = RandomForestClassifier(**best_params, random_state=654)
rf_best.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier(max_depth=5, max_features='sqrt', random_state=654)
```

In [ ]:
```
# evaluate the performance
rf_best_predictions = rf_best.predict(X_test)
print(classification_report(y_test, rf_best_predictions))
```

```
              precision    recall  f1-score   support

           0       0.69      0.65      0.67      1013
           1       0.62      0.66      0.64       862

    accuracy                           0.66      1875
   macro avg       0.66      0.66      0.66      1875
weighted avg       0.66      0.66      0.66      1875
```

In [ ]:
```
train_predictions = rf_best.predict(X_train)
test_predictions = rf_best.predict(X_test)

train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)

print(f"Training set accuracy: {train_accuracy}")
print(f"Test set accuracy: {test_accuracy}")
```

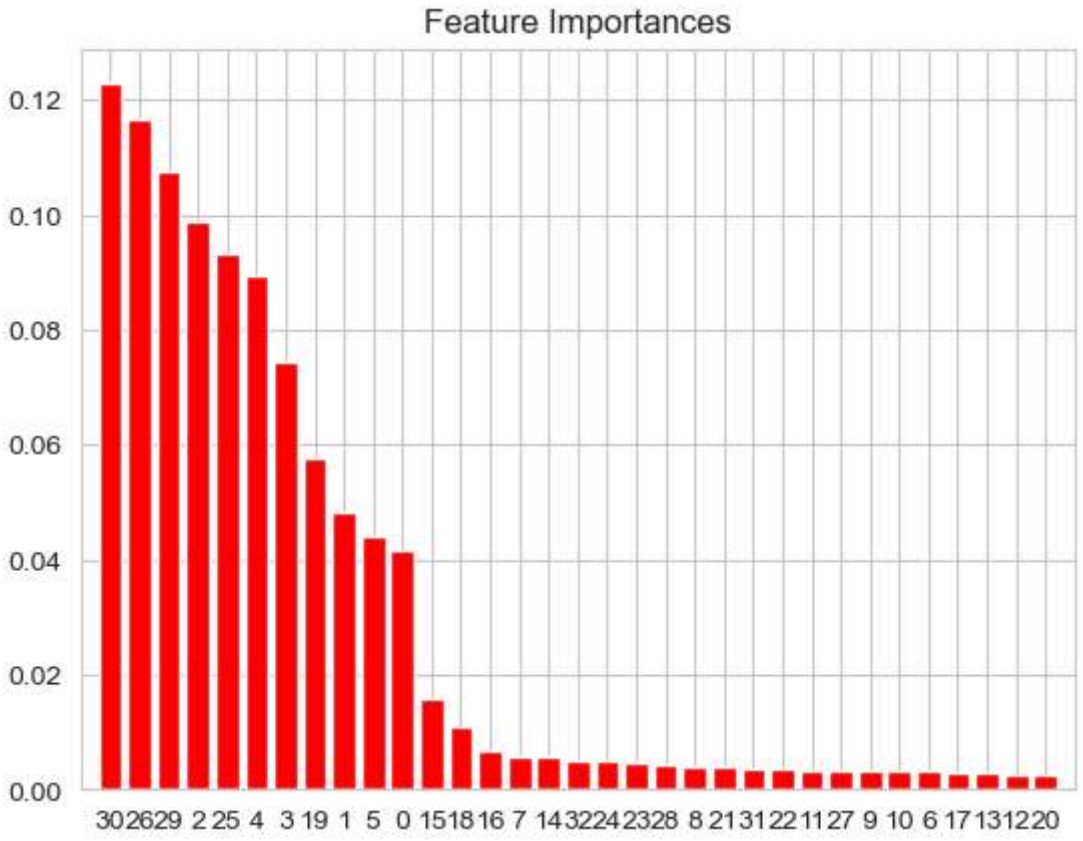
```
Training set accuracy: 0.6978285714285715
Test set accuracy: 0.6565333333333333
```

The random forest model generalizing well as both accuracies are relatively close.

In [ ]:
```
importances = rf_best.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_best.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
# printing feature ranking
print("Feature Ranking:")
for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

plt.figure()
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", align="center")
```

```
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

```
Feature Ranking:
1. feature 30 (0.122765)
2. feature 26 (0.116581)
3. feature 29 (0.107361)
4. feature 2 (0.098813)
5. feature 25 (0.093299)
6. feature 4 (0.089470)
7. feature 3 (0.074269)
8. feature 19 (0.057585)
9. feature 1 (0.048394)
10. feature 5 (0.044084)
11. feature 0 (0.041704)
12. feature 15 (0.015883)
13. feature 18 (0.011043)
14. feature 16 (0.006685)
15. feature 7 (0.005835)
16. feature 14 (0.005628)
17. feature 32 (0.004868)
18. feature 24 (0.004832)
19. feature 23 (0.004678)
20. feature 28 (0.004195)
21. feature 8 (0.003892)
22. feature 21 (0.003774)
23. feature 31 (0.003723)
24. feature 22 (0.003685)
25. feature 11 (0.003376)
26. feature 27 (0.003293)
27. feature 9 (0.003184)
28. feature 10 (0.003182)
29. feature 6 (0.003147)
30. feature 17 (0.002993)
31. feature 13 (0.002827)
32. feature 12 (0.002556)
33. feature 20 (0.002393)
```



In [ ]:
```
important_features = pd.Series(data=rf_best.feature_importances_, index=X_train.columr
important_features.sort_values(ascending=False, inplace=True)
important_features
```

Out[ ]:
```
park_package_1                0.122765
discount_original_1           0.116581
park_package_0                0.107361
res_to_port                   0.098813
discount_original_0           0.093299
oldest_child_age              0.089470
number_children               0.074269
embark_port_Vancouver         0.057585
incidental_spending           0.048394
log_est_inc                   0.044084
ticket_spend                  0.041704
embark_port_Copenhagen        0.015883
embark_port_San Diego         0.011043
embark_port_New York          0.006685
cabin_type_Inside             0.005835
embark_port_Barcelona         0.005628
dis_plus_sub_1                0.004868
cruise_theme_starwars         0.004832
cruise_theme_prin_fro         0.004678
coupon_received_1             0.004195
cabin_type_Oceanview          0.003892
cruise_theme_Princess/Frozen  0.003774
dis_plus_sub_0                0.003723
cruise_theme_Star Wars        0.003685
season_of_year_Spring         0.003376
coupon_received_0             0.003293
cabin_type_Verandah           0.003184
season_of_year_Fall           0.003182
cabin_type_Concierge          0.003147
embark_port_Port Canaveral    0.002993
season_of_year_Winter         0.002827
season_of_year_Summer         0.002556
cruise_theme_Marvel           0.002393
dtype: float64
```

Assume a cruise customer called Alice, we want to use random forest model to predict if she will take the cruise again within 3 years.

In [ ]:
```
cruise_dummies.columns
```

Out[ ]:
```
Index(['ticket_spend', 'incidental_spending', 'res_to_port', 'number_children',
       'oldest_child_age', 'return_1065', 'log_est_inc',
       'cabin_type_Concierge', 'cabin_type_Inside', 'cabin_type_Oceanview',
       'cabin_type_Verandah', 'season_of_year_Fall', 'season_of_year_Spring',
       'season_of_year_Summer', 'season_of_year_Winter',
       'embark_port_Barcelona', 'embark_port_Copenhagen',
       'embark_port_New York', 'embark_port_Port Canaveral',
       'embark_port_San Diego', 'embark_port_Vancouver', 'cruise_theme_Marvel',
       'cruise_theme_Princess/Frozen', 'cruise_theme_Star Wars',
       'cruise_theme_prin_fro', 'cruise_theme_starwars', 'discount_original_0',
       'discount_original_1', 'coupon_received_0', 'coupon_received_1',
       'park_package_0', 'park_package_1', 'dis_plus_sub_0', 'dis_plus_sub_1'],
      dtype='object')
```

In [ ]:
```
alice = pd.DataFrame({
    'ticket_spend': [3000],
    'incidental_spending': [1500],
    'res_to_port': [50],
    'number_children': [2],
```

```
    'oldest_child_age': [5],
    'log_est_inc': [10],
    'cabin_type_Concierge': [0],
    'cabin_type_Inside': [0],
    'cabin_type_Oceanview': [0],
    'cabin_type_Verandah': [1],
    'season_of_year_Fall': [0],
    'season_of_year_Spring': [1],
    'season_of_year_Summer': [0],
    'season_of_year_Winter': [0],
    'embark_port_Barcelona': [0],
    'embark_port_Copenhagen': [0],
    'embark_port_New York': [1],
    'embark_port_Port Canaveral': [0],
    'embark_port_San Diego': [0],
    'embark_port_Vancouver': [0],
    'cruise_theme_Marvel': [0],
    'cruise_theme_Princess/Frozen': [0],
    'cruise_theme_Star Wars': [1],
    'cruise_theme_prin_fro': [0],
    'cruise_theme_starwars': [0],
    'discount_original_0': [0],
    'discount_original_1': [1],
    'coupon_received_0': [0],
    'coupon_received_1': [1],
    'park_package_0': [0],
    'park_package_1': [1],
    'dis_plus_sub_0': [0],
    'dis_plus_sub_1': [1]
})

predicted_return = rf_best.predict(alice)
print("Predicted return_1065:", predicted_return[0])
```

```
Predicted return_1065: 0
```

In [ ]:
```
predicted_prob = rf_best.predict_proba(alice)
print("Probability of return:", predicted_prob[0][1])
```

```
Probability of return: 0.21346008621699308
```

Logistic Regression

The features are ranked in descending order of importance. The top-ranked features can be considered key drivers or predictors of model results. 'park_package_1' is the most important feature and 'cruise_theme_Marvel' is the least important one. Now, we will do the feature selection. Remove any numeric features that are not in the top 10 importance, and keep the entire variable (all levels) if any level of that variable was in the top 10 list.

In [ ]:
```
columns_to_drop = [
    "ticket_spend", "cabin_type_Inside", "dis_plus_sub_1", "cruise_theme_starwars",
    "cruise_theme_prin_fro", "coupon_received_1", "cabin_type_Oceanview",
    "cruise_theme_Princess/Frozen", "dis_plus_sub_0", "cruise_theme_Star Wars",
    "season_of_year_Spring", "coupon_received_0", "cabin_type_Verandah",
    "season_of_year_Fall", "cabin_type_Concierge", "season_of_year_Winter",
    "season_of_year_Summer", "cruise_theme_Marvel"
```

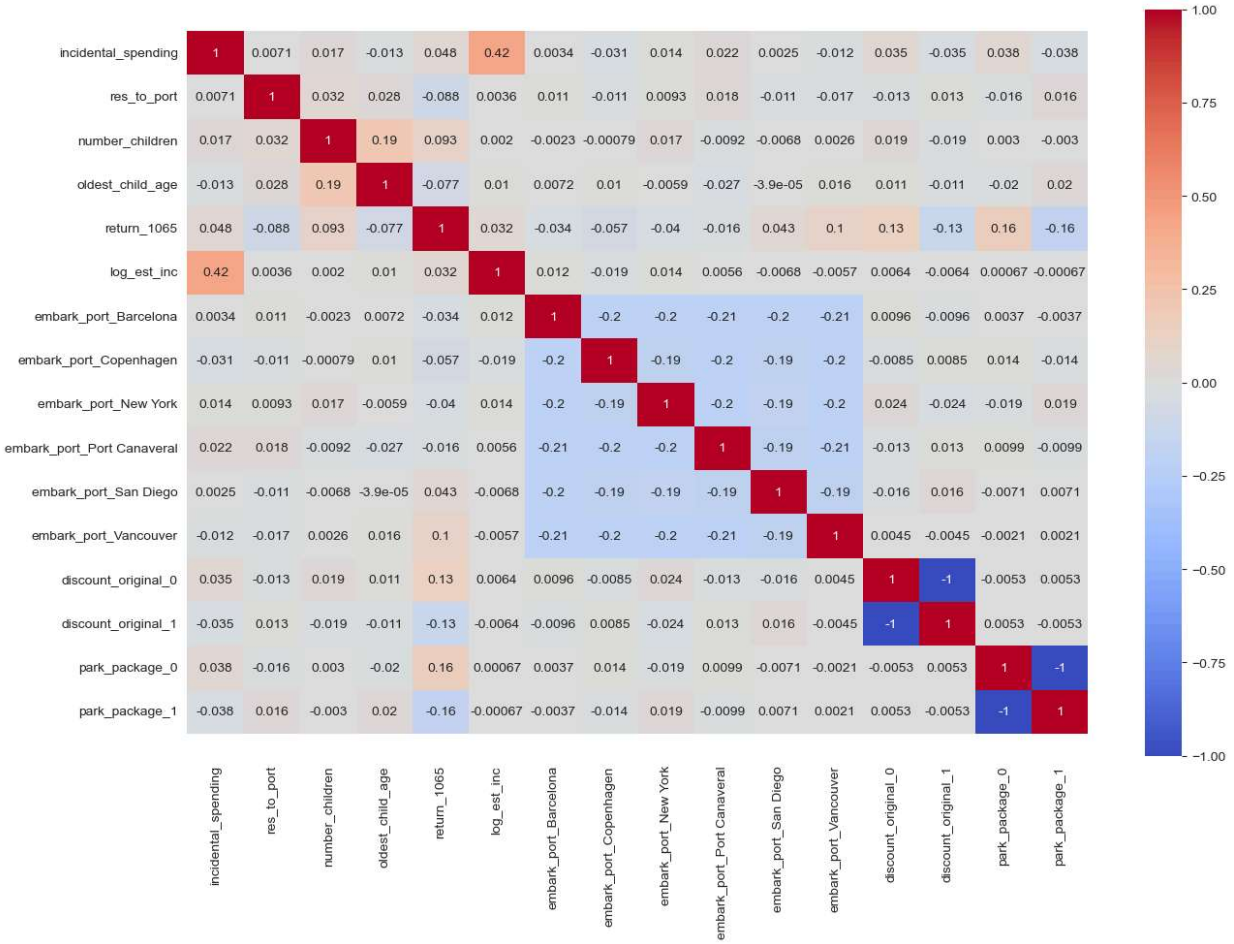
```
]

cruise_dummies_drop = cruise_dummies.drop(columns_to_drop,1)
cruise_dummies_drop.shape
```

```
/var/folders/_8/7ywrh1p94c52x8wdv9nlp14c0000gn/T/ipykernel_21343/3827744512.py:10: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  cruise_dummies_drop = cruise_dummies.drop(columns_to_drop,1)
```

Out[ ]:
```
(6250, 16)
```

Check if there are high correlations between features.

In [ ]:
```
cruise_corr = cruise_dummies_drop.corr()
plt.figure(figsize=(15, 10))
ax = sns.heatmap(cruise_corr, annot=True, cmap="coolwarm")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5);
```



In [ ]:
```
# Delete 'discount_original_0' and 'park_package_0', since they are perfectly negative
# 'discount_original_1' and 'park_package_1'
cruise_dummies_drop = cruise_dummies_drop.drop (columns = ['discount_original_0','park
```

In [ ]:
```
cruise_lr = cruise_dummies_drop
X = cruise_lr.drop(columns = ['return_1065'])
y = cruise_lr ['return_1065'] # predictor

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

```
print(f'X_train : {X_train.shape}')
print(f'y_train : {y_train.shape}')
print(f'X_test : {X_test.shape}')
print(f'y_test : {y_test.shape}')
```

```
X_train : (4375, 13)
y_train : (4375,)
X_test : (1875, 13)
y_test : (1875,)
```

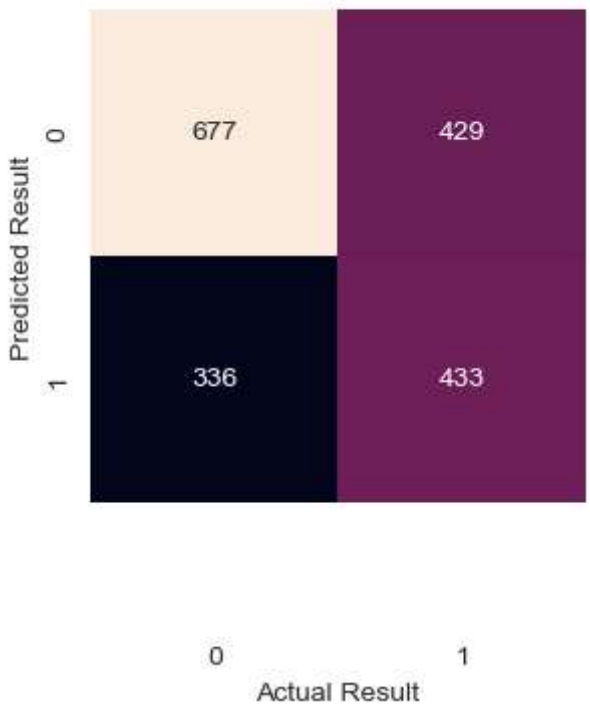
In [ ]:
```
# Build a Logistic Regression model
logmodel = LogisticRegression(max_iter=1000)
logmodel.fit(X_train, y_train)
predictions = logmodel.predict(X_test)
predictions
```

Out[ ]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [ ]:
```
probabilities = logmodel.predict_proba(X_test)
probabilities
```

Out[ ]:
```
array([[0.60175975, 0.39824025],
       [0.50019597, 0.49980403],
       [0.54559583, 0.45440417],
       ...,
       [0.59265699, 0.40734301],
       [0.60765945, 0.39234055],
       [0.5555549 , 0.4444451 ]])
```

In [ ]:
```
# Create a confusion matrix
mat = confusion_matrix(predictions, y_test)
sns.heatmap(mat, square=True, annot=True, fmt='g', cbar=False)
plt.xlabel("Actual Result")
plt.ylabel("Predicted Result")
a, b = plt.ylim()
a += 0.5
b -= 0.5
plt.ylim(a, b)
plt.show()
```



The logistic regression model's accuracy rate is 0.60 = (677+433)/1875.

In [ ]:
```
train_predictions = logmodel.predict(X_train)
test_predictions = logmodel.predict(X_test)

train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)

print(f"Training set accuracy: {train_accuracy}")
print(f"Test set accuracy: {test_accuracy}")
```

```
Training set accuracy: 0.5952
Test set accuracy: 0.592
```

The logistic regression model generalizing well as both accuracies are relatively close.

In [ ]:
```
logmodel.intercept_
```

Out[ ]:
```
array([0.00621739])
```

In [ ]:
```
pd.DataFrame(data=logmodel.coef_.transpose(),index=X_train.columns,columns=['Coef'])
```

Out[ ]:

	Coef
incidental_spending	0.000012
res_to_port	-0.003717
number_children	0.289436
oldest_child_age	-0.049230
log_est_inc	-0.005574
embark_port_Barcelona	-0.029137
embark_port_Copenhagen	-0.042192
embark_port_New York	-0.031170
embark_port_Port Canaveral	-0.019131
embark_port_San Diego	0.036906
embark_port_Vancouver	0.090940
discount_original_1	-0.137059
park_package_1	-0.157630

As we can see from the coefficient results, certain factors can greatly influence the likelihood that a family will take another Disney cruise. Families who spend more incidentally on a cruise are slightly more likely to cruise again, while families who are farther away from their departure port are slightly less likely to book again. Families with more children are more likely to cruise again, but this tendency decreases as the age of the oldest child increases. The role of port of embarkation is also evident; specifically, families embarking from Vancouver or San Diego are more inclined to cruise again than those embarking from ports such as Barcelona, Copenhagen, New York and Canaveral. Additionally, families who took advantage of discounts or used park packages on their first booking appeared less likely to cruise again, suggesting that a full-price cruise experience may result in higher satisfaction. Alternatively, once people take advantage of a discount, they downplay the perceived value of the brand or service.

Then, we use logistic regression model to predict if Alice will take the cruise again within 3 years.

In [ ]:
```
cruise_lr.columns
```

Out[ ]:
```
Index(['incidental_spending', 'res_to_port', 'number_children',
       'oldest_child_age', 'return_1065', 'log_est_inc',
       'embark_port_Barcelona', 'embark_port_Copenhagen',
       'embark_port_New York', 'embark_port_Port Canaveral',
       'embark_port_San Diego', 'embark_port_Vancouver', 'discount_original_1',
       'park_package_1'],
      dtype='object')
```

In [ ]:
```
alice2 = pd.DataFrame({
    'incidental_spending': [1500],
    'res_to_port': [50],
    'number_children': [2],
```

```
    'oldest_child_age': [5],
    'log_est_inc': [10],
    'embark_port_Barcelona': [0],
    'embark_port_Copenhagen': [0],
    'embark_port_New York': [1],
    'embark_port_Port Canaveral': [0],
    'embark_port_San Diego': [0],
    'embark_port_Vancouver': [0],
    'discount_original_1': [1],
    'park_package_1': [1]
})

predicted_return = logmodel.predict(alice2)
print("Predicted return_1065:", predicted_return[0])
```

```
Predicted return_1065: 0
```

In [ ]:

```
predicted_prob = logmodel.predict_proba(alice2)
print("Probability of return:", predicted_prob[0][1])
```

```
Probability of return: 0.44756323115964786
```

The Random Forest Model and Logistic Regression Model provide Disney with powerful tools to strengthen its cruise business through data-driven insights. The combined application of these two models provides a more comprehensive understanding of customer behavior. With for the random forest model and the logistic regression model both greater than 0.6, the predictions are relatively reliable. Disney can predict with relative accuracy whether a customer will return within three years. This predictive ability allows for effective allocation of marketing resources and optimization of strategies, resulting in increased customer loyalty and expansion of cruise business.

The Random Forest model identifies key determinants of customer return, while logistic regression quantifies the impact of these factors. By examining important characteristics, Disney can identify specific areas of the cruise experience that have the deepest impact on customer loyalty. For example, if guests have poor revisit data at certain ports of embarkation, Disney can drill down to understand why and correct any potential problems. Similarly, insight into the impact of initial discounts on repeat business can lead to targeted marketing strategies. Additionally, by understanding the key parameters that influence repeat bookings, Disney can tailor unique packages or offers to specific customer segments. For example, if it is found that families with young children are more likely to book again, then packages can be developed specifically for this demographic. Conversely, understanding the deterrents allows Disney to address pain points and ensure that they don't reoccur, thereby preserving the brand's reputation.

With trained models, Disney can input current customer data and predict outcomes. This prediction can guide marketing efforts and help Disney deliver the right product to the right audience. Using predictive modeling, Disney can segment audiences based on the likelihood of taking another cruise. For those who are not expected to take another cruise, feedback can be solicited to better understand their experience and address any issues. Not only does this provide valuable insights to improve service, it also shows customers that their feedback is valued, which increases the likelihood of them returning. Additionally, as market dynamics

continue to change, these models can be continually trained with new data to ensure they remain relevant and accurate.

In [ ]:
```
%cd /Users/claudiasmac/Desktop/BU ABA/AD 654/datasets
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
```

```
/Users/claudiasmac/Desktop/BU ABA/AD 654/datasets
```

In [ ]:
```
popcorn = pd.read_csv("popcorn_buckets.csv")
popcorn.head()
```

Out[ ]:

	location	bucket	Q1_sales_USD
0	Disneyland Paris	Cinderella	139413.1
1	Disneyland Paris	Star_Wars	138890.4
2	Disneyland Paris	Mermaid	117696.4
3	Disneyland Paris	MickeySuit	176821.9
4	Disneyland Paris	MickeyEars	168442.1

In [ ]:
```
popcorn.shape
```

Out[ ]:
```
(36, 3)
```

In [ ]:
```
popcorn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   location      36 non-null     object 
 1   bucket        36 non-null     object 
 2   Q1_sales_USD  36 non-null     float64
dtypes: float64(1), object(2)
memory usage: 992.0+ bytes
```

In [ ]:
```
# unique buckets
buckets = popcorn['bucket'].unique()
print(buckets)
```

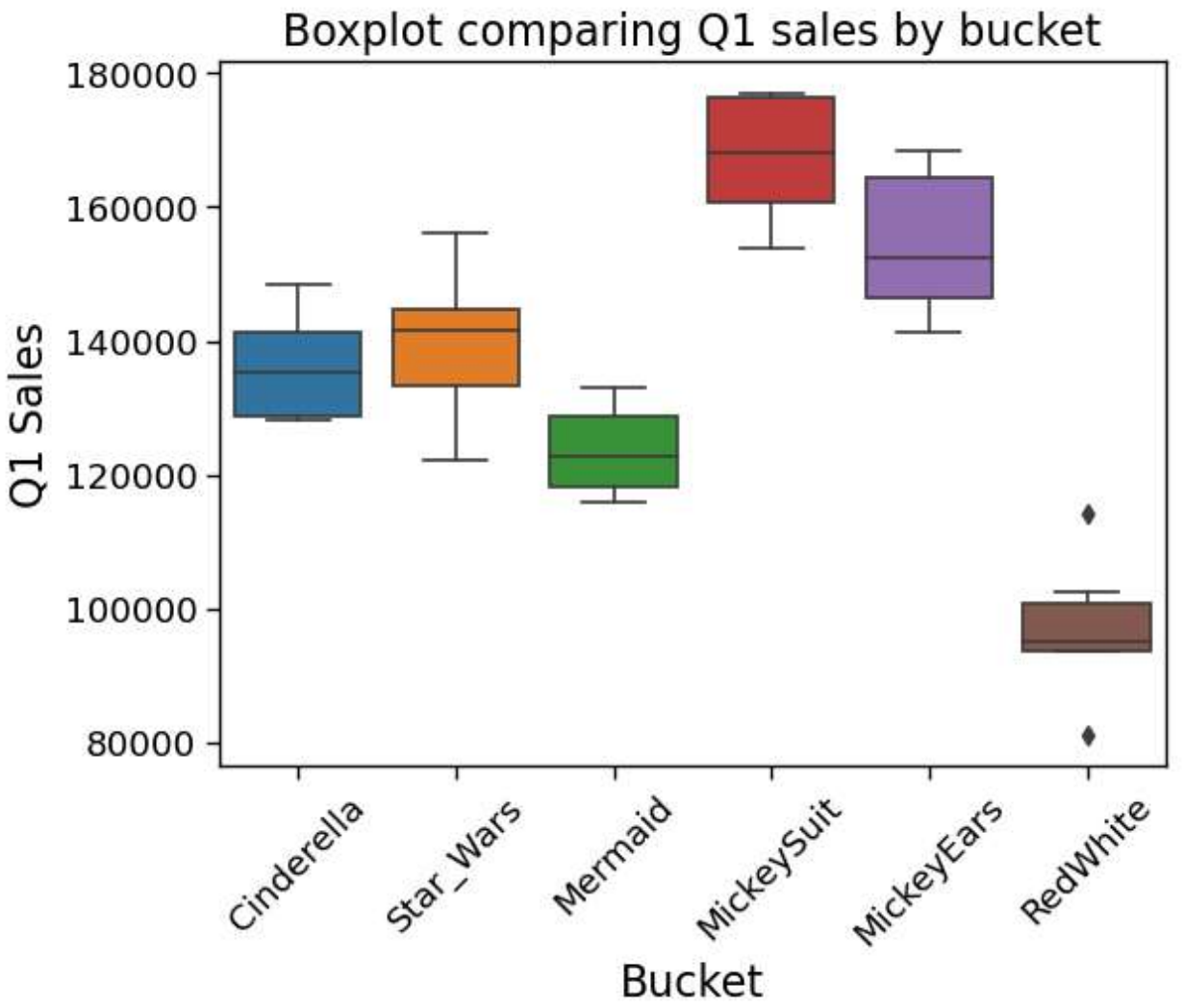
```
['Cinderella' 'Star_Wars' 'Mermaid' 'MickeySuit' 'MickeyEars' 'RedWhite']
```

In [ ]:
```
bygroup = popcorn.groupby(['bucket'])['Q1_sales_USD']
bygroup.describe()
```

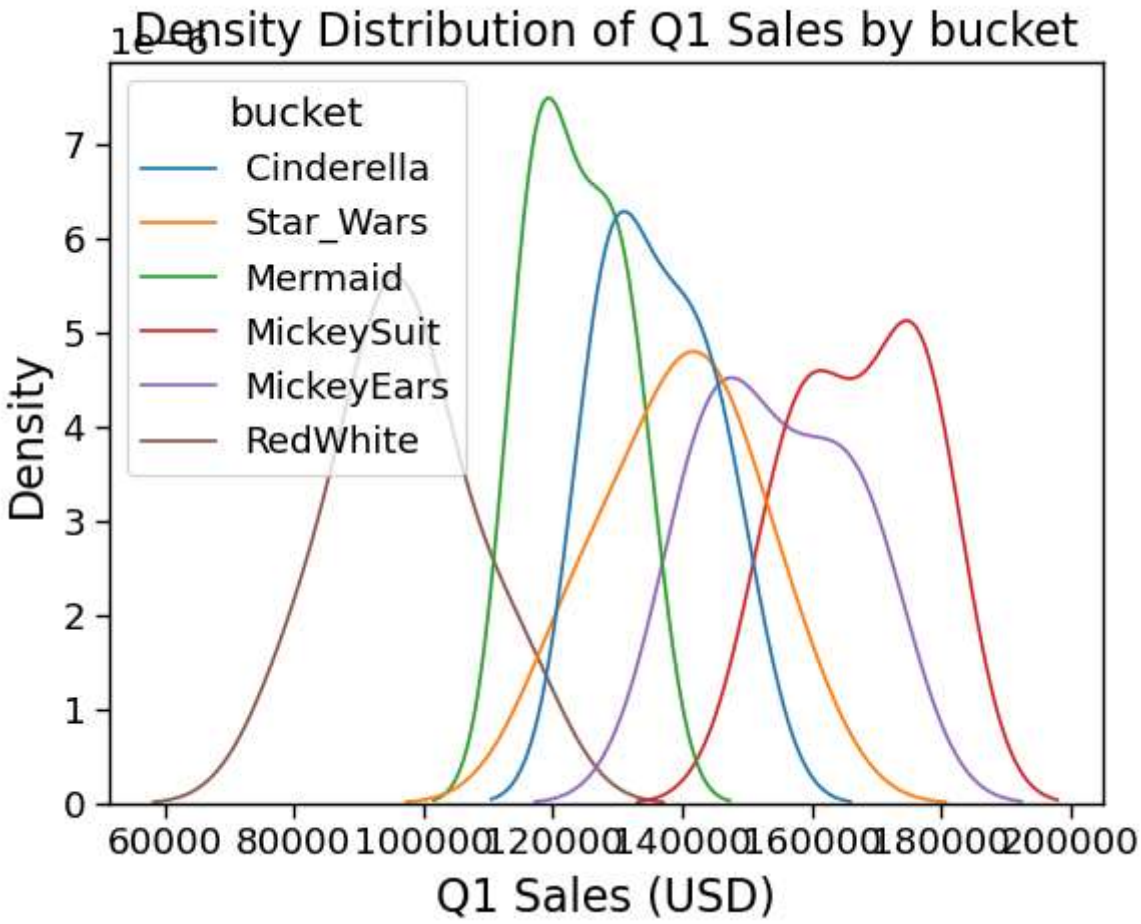
Out[ ]:

bucket	count	mean	std	min	25%	50%	75%	max
Cinderella	6.0	136139.850000	8362.372118	128031.7	128860.350	135215.75	141216.100	148417.8
Mermaid	6.0	123628.983333	6882.250179	116015.3	118180.575	122721.40	128754.350	132883.5
MickeyEars	6.0	154566.350000	11395.606430	141166.3	146513.325	152454.00	164374.950	168442.7
MickeySuit	6.0	167465.233333	9942.367653	153955.9	160831.200	168215.60	176274.700	177073.5
RedWhite	6.0	96923.400000	10895.539817	81159.8	93744.325	95140.45	100796.275	114139.2
Star_Wars	6.0	139564.566667	11778.816778	122046.4	133260.525	141632.35	144616.625	155995.0

In [ ]:
```
%matplotlib inline
sns.set_context("paper", font_scale=1.5, rc={"font.size":16,"axes.titlesize":16,"axes.
plt.title('Boxplot comparing Q1 sales by bucket')
sns.boxplot(x='bucket', y='Q1_sales_USD', data=popcorn)
plt.xlabel('Bucket')
plt.ylabel('Q1 Sales')
plt.xticks(rotation=45);
```



In [ ]:
```
# Create a density plot for all buckets
sns.kdeplot(data=popcorn, x="Q1_sales_USD", hue="bucket")

# Add labels and title
plt.title('Density Distribution of Q1 Sales by bucket')
plt.ylabel('Density')
plt.xlabel('Q1 Sales (USD)')
plt.show()
```



In [ ]:
```
# Form pairs of buckets
bucket_pairs = [(buckets[i], buckets[j]) for i in range(len(buckets)) for j in range(i
print(bucket_pairs)
```

```
[('Cinderella', 'Star_Wars'), ('Cinderella', 'Mermaid'), ('Cinderella', 'MickeySui
t'), ('Cinderella', 'MickeyEars'), ('Cinderella', 'RedWhite'), ('Star_Wars', 'Mermai
d'), ('Star_Wars', 'MickeySuit'), ('Star_Wars', 'MickeyEars'), ('Star_Wars', 'RedWhit
e'), ('Mermaid', 'MickeySuit'), ('Mermaid', 'MickeyEars'), ('Mermaid', 'RedWhite'),
('MickeySuit', 'MickeyEars'), ('MickeySuit', 'RedWhite'), ('MickeyEars', 'RedWhite')]
```

Null Hypothesis: There is no significant difference in Q1 sales between bucket1 and bucket2.
Alternative Hypothesis: There is a significant difference in Q1 sales between bucket1 and bucket2.

Note: bucket1 is the first bucket in each bucket pair and bucket2 is the second.

In [ ]:
```
significant_diff_pairs = []

# For each bucket pair, run the t-test
```

```
for bucket1, bucket2 in bucket_pairs:
    sales1 = popcorn[popcorn['bucket'] == bucket1]['Q1_sales_USD']
    sales2 = popcorn[popcorn['bucket'] == bucket2]['Q1_sales_USD']

    (tstat, p_value) = stats.ttest_ind(sales1, sales2, equal_var = False)

    if p_value < 0.05:
        significant_diff_pairs.append((bucket1, bucket2, p_value))

significant_diff_pairs
```

Out[ ]:
```
[('Cinderella', 'Mermaid', 0.01848018889193976),
 ('Cinderella', 'MickeySuit', 0.00016808092548139614),
 ('Cinderella', 'MickeyEars', 0.0106869869778289),
 ('Cinderella', 'RedWhite', 5.196188366704971e-05),
 ('Star_Wars', 'Mermaid', 0.020953882020095415),
 ('Star_Wars', 'MickeySuit', 0.0013553943725733311),
 ('Star_Wars', 'MickeyEars', 0.0488533470599294),
 ('Star_Wars', 'RedWhite', 7.004165914880581e-05),
 ('Mermaid', 'MickeySuit', 1.0283424886094439e-05),
 ('Mermaid', 'MickeyEars', 0.00041528571189356715),
 ('Mermaid', 'RedWhite', 0.0008120282520849337),
 ('MickeySuit', 'RedWhite', 3.953144518547701e-07),
 ('MickeyEars', 'RedWhite', 4.3870743322442576e-06)]
```

In [ ]:
```
# Bonferroni correction
alpha = 0.05
corrected_alpha = alpha / len(bucket_pairs)

significant_diff_pairs2 = []

# For each bucket pair, run the t-test
for bucket1, bucket2 in bucket_pairs:
    sales1 = popcorn[popcorn['bucket'] == bucket1]['Q1_sales_USD']
    sales2 = popcorn[popcorn['bucket'] == bucket2]['Q1_sales_USD']

    (tstat, p_value) = stats.ttest_ind(sales1, sales2, equal_var = False)

    if p_value < corrected_alpha:  # using the adjusted alpha
        significant_diff_pairs2.append((bucket1, bucket2, p_value))

significant_diff_pairs2
```

Out[ ]:
```
[('Cinderella', 'MickeySuit', 0.00016808092548139614),
 ('Cinderella', 'RedWhite', 5.196188366704971e-05),
 ('Star_Wars', 'MickeySuit', 0.0013553943725733311),
 ('Star_Wars', 'RedWhite', 7.004165914880581e-05),
 ('Mermaid', 'MickeySuit', 1.0283424886094439e-05),
 ('Mermaid', 'MickeyEars', 0.00041528571189356715),
 ('Mermaid', 'RedWhite', 0.0008120282520849337),
 ('MickeySuit', 'RedWhite', 3.953144518547701e-07),
 ('MickeyEars', 'RedWhite', 4.3870743322442576e-06)]
```

Firstly, we did data exploration and assumed that each bucket at each location is independent and follows a normal distribution. We created a total of 15 unique pairs, each with two different buckets. They were then subjected to a paired t-test to compare Q1 sales. The essence of the paired t-test is to determine if there is a statistically significant difference between the means of the two groups. We set the significance level (α) at 0.5 to identify the two groups whose sales were significantly different. However, performing multiple t-tests increases the likelihood of a

Type 1 error, so we used the Bonferroni correction to offset this. After the adjustment, we had the output of 'the significant_diff_pairs2'. Based on the result, we see that 'MickeySuit' bucket consistently showed significant differences when compared with 'Cinderella', 'Star_Wars', 'Mermaid', and 'RedWhite'. It might imply that the 'MickeySuit' bucket is either very popular or not popular compared to these others. Comparing this result to our previous box plot, it is clear that "MickeySuit" is the most popular. Besides, 'RedWhite' also consistently showed significant differences when paired with several other buckets, which indicated 'RedWhite' sales might be very distinct from the others. Leveraging the insights from the boxplot,'RedWhite' is underperforming compared to others. For Disney's strategic move forward, it might be worth pushing promotions around 'MickeySuit'. Also, they should do more research to understand the reason of low sales of 'RedWhite' or consider phasing out 'RedWhite'. Disney could also consider bundling 'MickeySuit' and 'RedWhite' as a promotional deal to stimulate the sales of 'RedWhite'.