Link: https://public.tableau.com/app/profile/kexin.xi/viz/654finalQ1/1_1#1

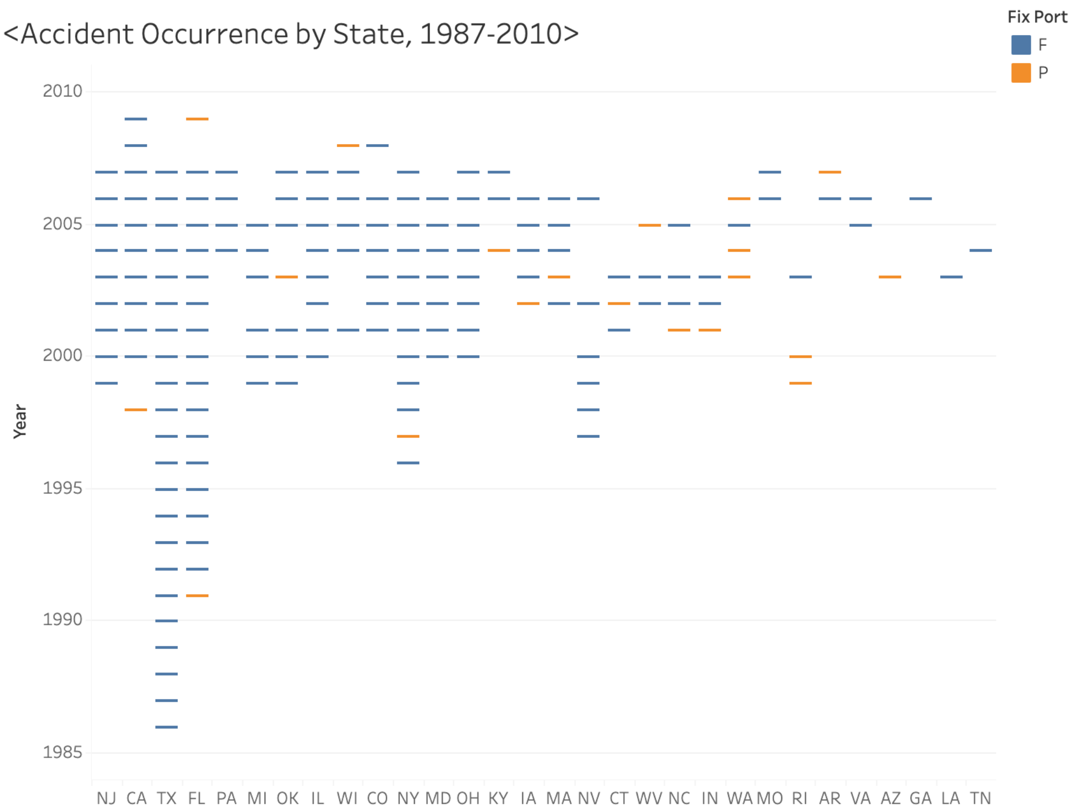
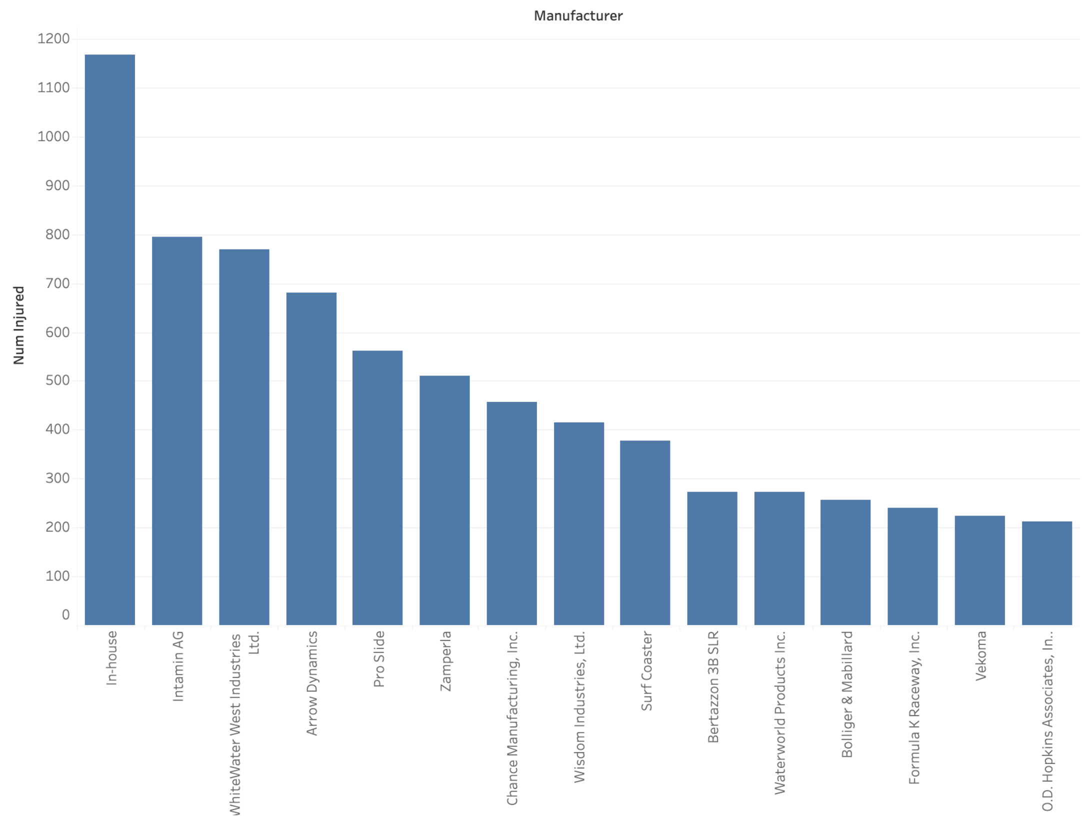


Figure 1: This Gantt bar chart shows the occurrence of park accidents by state in the United States between 1985 and 2010. From left to right by the number of accidents, so we can see that although the first accident occurred in Texas in 1986, the state with the highest number of accidents was the state of NJ. Over time, 1999 to 2007 produced a period of high accident frequency in each state, with a decrease in accidents after 2007, presumably due to the enactment of policies or improvements in facilities that occurred or simply insufficient data collection.

In addition, I distinguish the type of equipment by color, blue indicates fixed facilities: such as roller coasters and carousels, and yellow indicates portable facilities: such as Ferris wheels, giant slides and bouncy castles. Most of the accidents originate from fixed facilities, showing that portable equipment is much safer than fixed equipment.

<Histogram of the injured numbers by Manufacturer(Top15)>



<Pie chart of the injured percentage by Industry Sectors>

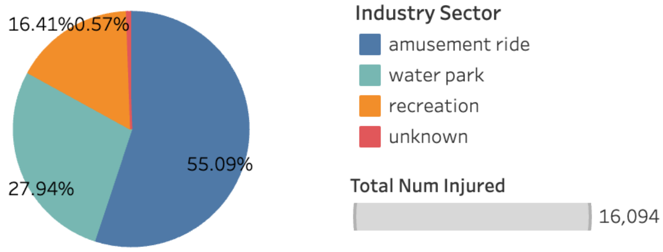
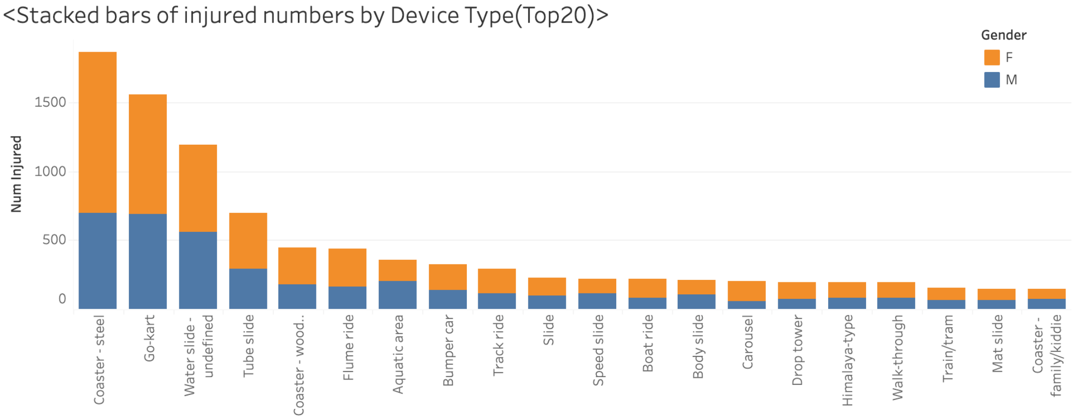


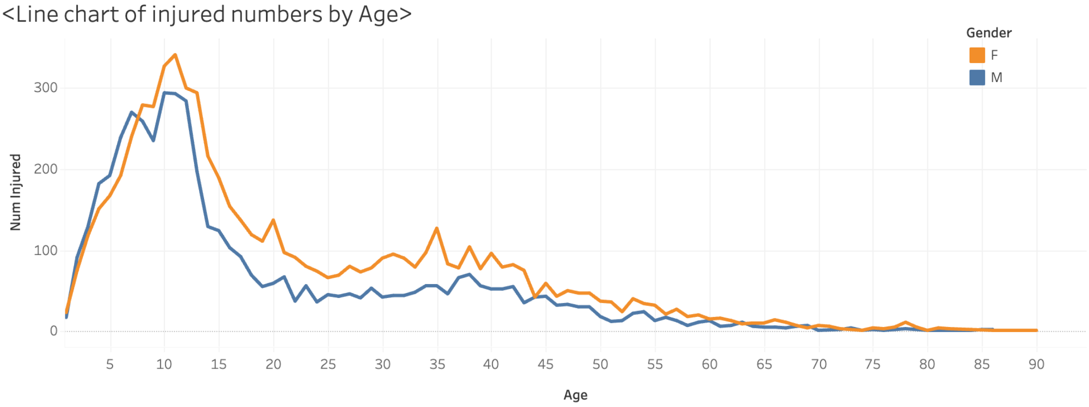
Figure 2-3: Figure 2 is a histogram of the number of injuries for different manufacturers, it can be seen that the manufacturer with the lowest safety level is In-house, and in addition for these manufacturers that cause the highest volume of accidents, managers are advised to choose carefully.

Figure 3 shows the number of accidents in different industry sectors, in which 55% of the accidents occurred in the areas of amusement rides, and the second place is water parks with 28% of accidents, based on this, managers can improve risk control in the relevant regions, such as raising the maintenance rate of amusement rides and increasing the number of lifeguards in water parks.

<Stacked bars of injured numbers by Device Type(Top20)>



<Line chart of injured numbers by Age>



Figures 4-5: Table 4 shows the accident occurrence of different types of devices. Among these, roller coaster-steel, go-kart and water slide are the top three in terms of accident occurrence, and their number is much higher than other device types.

Chart 5 shows the age and gender distribution of the injured. In terms of age, children around 10 years old have the highest injury rate, and girls are generally more vulnerable than boys. In addition, there is a small peak of 30-40 years old, so it is recommended that the park set up reminder notices to pay attention to children's safety.

Limitation of dataset：

- Excessive text content prevents in-depth data analysis.
- The data are too old, with the latest year being 2010, which may lead to errors in the analysis results and expectations.
- There are a lot of duplicate variables, such as 'device_category', 'device_type', 'tradename_or_generic', which only have the difference of dividing coarse and fine, thereby increasing the complexity of dividing data.

Shaoguang Yang AD654 Final Project

- Summary Stats
- Professor Page
- April 30, 2023

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import groupby
```

```
accident = pd.read_csv("park_accidents.csv")
```

Data Exploration and Summary Statistics

```
accident.columns
```

```
Index(['acc_id', 'acc_date', 'acc_state', 'acc_city', 'fix_port', 'source',
       'bus_type', 'industry_sector', 'device_category', 'device_type',
       'tradename_or_generic', 'manufacturer', 'num_injured', 'age_youngest',
       'gender', 'acc_desc', 'injury_desc', 'report', 'category',
'mechanical',
       'op_error', 'employee', 'notes', 'year'],
      dtype='object')
```

```
accident.dtypes
```

```
acc_id                   int64
acc_date                object
acc_state               object
acc_city                object
fix_port                object
source                  object
bus_type                object
industry_sector         object
device_category         object
device_type             object
tradename_or_generic    object
manufacturer            object
num_injured              int64
age_youngest           float64
gender                  object
acc_desc                object
injury_desc             object
report                  object
category                object
mechanical                bool
op_error                  bool
employee                  bool
notes                   object
year                     int64
dtype: object
```

```
#summary statistics on the entire dataset
accident.describe()
```

	acc_id	num_injured	age_youngest	year
count	1.488400e+04	14884.000000	14884.000000	14884.000000
mean	9.097222e+05	1.081295	16.858976	2001.885783
std	7.476700e+03	2.360132	16.542130	3.481648
min	8.973520e+05	0.000000	0.000000	1986.000000
25%	9.048528e+05	1.000000	4.000000	2000.000000
50%	9.102445e+05	1.000000	12.000000	2002.000000
75%	9.164742e+05	1.000000	27.000000	2005.000000
max	1.009106e+06	99.000000	110.000000	2009.000000

There's only a few numerical varibles in this dataset. By calling the describe function, the summary statistics could clearly be seen in the above chart.

```
#summary statistics 1 using groupby
manufacturer_num_injured = accident.groupby("manufacturer")["num_injured"].sum()
manufacturer_num_injured.sort_values(ascending=False)
```

```
manufacturer
0                                  2936
In-house                           1167
Intamin AG                          795
WhiteWater West Industries Ltd.     769
Arrow Dynamics                      681
                                   ...
Prime Play                            1
Falgas Commerical S.L.                1
Dino Jump Int.                        1
Vertical Reality                      0
Heintz Fahtze                         0
Name: num_injured, Length: 279, dtype: int64
```

The number of injuries of rides from different manufacturers are listed in a descending order. The higher up on the list, the more "dangerous" a manufacturer is. For example, there have been 795 people injured on the rides manufactured by Intamin AG. This company is more "dangerous" than the company that is one place below which is WhiteWater West that has caused 769 injuries in total. Customers' safety is the priority. Therefore, park management should avoid buying from or operating the rides manufactured by the companies at the top portion of this list.

```
#summary statistics 2 using groupby
industry_sector_injuries = accident.groupby("industry_sector")["num_injured"].sum()
industry_sector_injuries.sort_values(ascending=False)
```

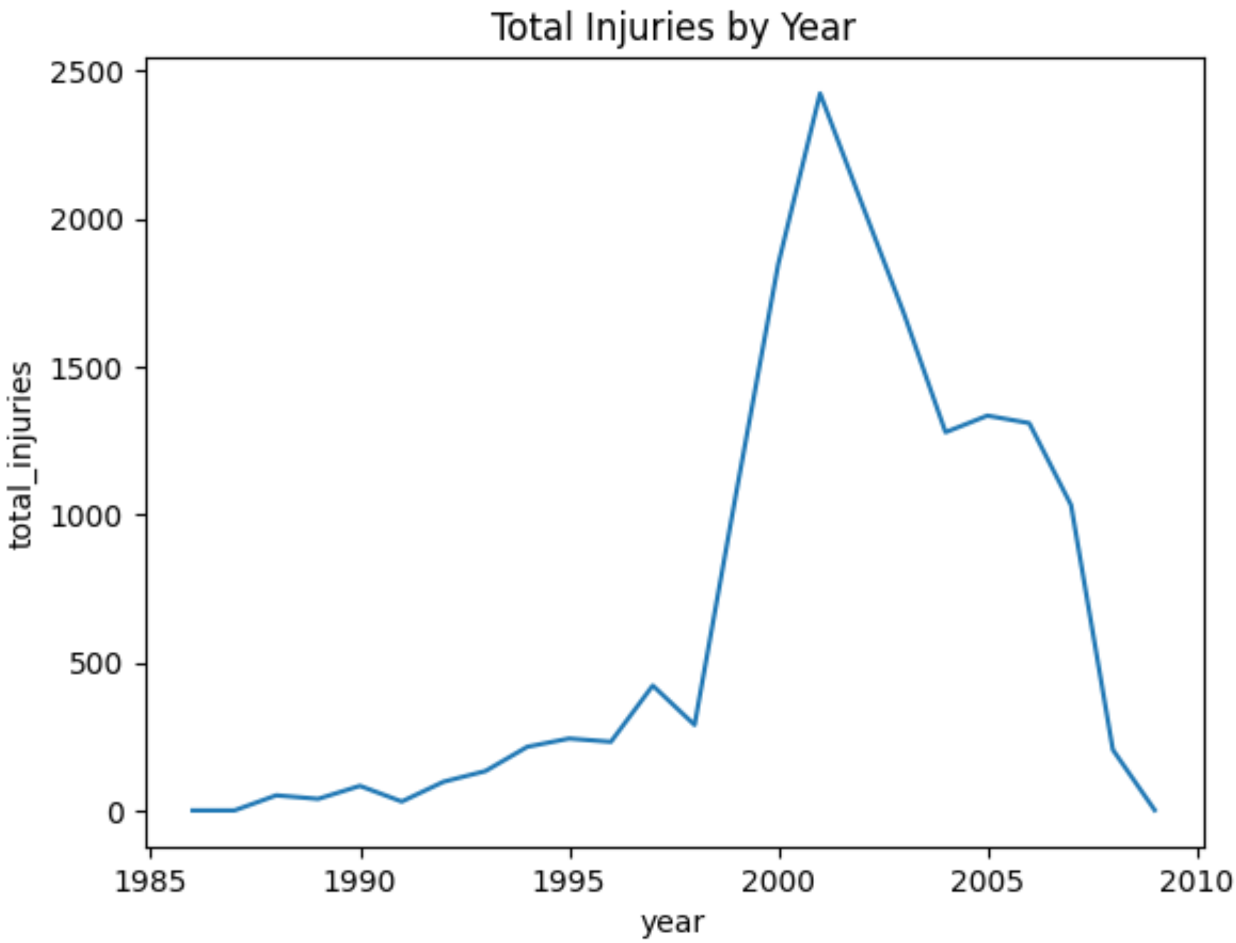
```
industry_sector
amusement ride    8866
water park        4496
recreation        2641
unknown             91
Name: num_injured, dtype: int64
```

This sums the number of injuries that occured at different industry sectors and displayed in a descending order. The higher up on the list, the more "dangerous" the corresponding industry sector is. Amusement ride is where the most number of injuries occured, corresponding to 8866 injuries in total. Therefore, park management should pay very close attention to the daily operation of amusement rides. If they see any red flags, they should shut down the rides immediately.

```
#summary statistics 3 using groupby
city_year_injuries = accident.groupby(["year"])["num_injured"].sum()
city_year_injuries
```

```
year
1986       1
1987       1
1988      52
1989      40
1990      84
1991      31
1992      98
1993     134
1994     216
1995     244
1996     233
1997     423
1998     290
1999    1078
2000    1847
2001    2424
2002    2051
2003    1682
2004    1279
2005    1335
2006    1310
2007    1033
2008     206
2009       2
Name: num_injured, dtype: int64
```

```
plt.plot(city_year_injuries.index, city_year_injuries.values)
plt.title("Total Injuries by Year")
plt.xlabel("year")
plt.ylabel("total_injuries")
plt.show()
```



This counts the number of injuries by year and display the result in chronological order. The general trend is that the number of injuries kept on increasing until the peak occured at 2001. Then, the number of injuries kept on decreasing throughout the years. This trend is also shown by the plot generated above. I think the park management handled the whole security issue well in recent years. But that's definitely not the sign that the park management could stop paying attention to rides security.

```
#summary statistics 4 using groupby
ride_injury = accident.groupby(["tradename_or_generic"])["num_injured"].count()
ride_injury.sort_values(ascending=False)
```

```
tradename_or_generic
go kart                    1656
waterslide                 1355
tube slide                  662
wooden coaster              515
flume ride                  471
                           ...
Space Shot                    1
Jet Star                      1
flipping platform ride        1
Loop-o-Plane                  1
Screamin' Swing               1
Name: num_injured, Length: 443, dtype: int64
```

This provides a more detailed breakdown of the specific rides where injuries occured. This basically tells us the "dangerousness" of any ride. The higher up on the chart, the more "dangerous" the ride is. There have been in total 1656 accidents that happened on "go kart" over the years, which makes it the most dangerous ride in the park. An ironic thing is that "screamin' swing" is actually not that dangerous because it's at the bottom of the list. Based on this chart, park management will know which ride should always be kept an eye on, and which rides are relatively safe.

```
#summary statistics 5 using groupby
injury_stats = accident.groupby("device_type")["category"].value_counts()
injury_stats.sort_values(ascending=False)
```

```
device_type              category
Go-kart                  Collision: patron-controlled vehicles
1063
Coaster - steel          Body pain (normal motion)
644
Water slide - undefined  Impact: hit something in participatory attraction
566
Go-kart                  Collision: go-kart or bumper car hit stationary
object     432
Coaster - steel          Impact: hit something within ride vehicle
390

...
Mat slide                Injured by foreign object
1
                         Impact: vaginal or rectal injury
1
                         Impact: hit wall or barrier at end of slide runout
1
                         Burn (includes friction burn)
1
Whip                     Impact: hit something within ride vehicle
1
Name: category, Length: 1055, dtype: int64
```

This links the device type with the category of the injury. In other words, this shows that each device could cause what type of injuries, and list the frequency of past injuries in a decreasing order. Based on this, the park management will know which security feature to implement for each ride. For example, the most frequent injury is body pain for steel roller coaster riders. Therefore, park management could consider to use better and thicker cushioned seats and handles on steel roller coaster rides so that less people would feel body pain after the ride.

Conclusion and Recommendation

In the above analysis, I mainly explored interesting relationships surrounding "num_injured" because I believe that the ultimate goal is to control the number of injuries. I tried to understand the number of injuries across years, and happily found that we have bypassed the peak and the injury number is steadily decreasing. I also showed which manufacturer to avoid based on the number of accosiated injuries. I also showed the number of injuries for each ride and suggest the park management to pay close attention to the rides that are high up on the chart. By exploring the relationship between the ride and the most common types of associated injuries, I suggest possible ways of improvements. Therefore, based on my analysis, park management know where problems lie and how to solve those problems accordingly.

several important findings:

- Go-kart is the most dangerous ride.
- Amusement ride is the most accident-prone industry sector.
- Manufacturing ride facitlities in-house is really dangerous and riders' safety would be compromised. Switch to safer alternatives.

limitations of the dataset:

- only a few numerical columns
- many boolean columns, need to switch to other datatype before doing any subsequent analysis on those columns.
- "notes" and "report" have many invalid entries, cannot be used in subsequent analysis.
- data are outdated.
- Possibly need more advanced techniques (like text mining) to do more in-depth analysis on "acc_desc" (accident description) and "injury desc" (injury description.)
- There are a lot of categorical columns in the dataset. A lot of them have many unique values, creating an extra layer of complexity if we want to dummify them.

Shaoguang Yang AD654 Final Project

- Segmentation & Targeting
- Professor Page
- April 30th, 2023

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
from sklearn import metrics
```

Part 1: Data Preparation

```
ski = pd.read_csv("ski_hotels.csv")
```

```
ski
```

	Unnamed: 0	country	resort	hotel	price (£)	distance_from_lift_(m)	alt
0	0	italy	bardonecchia	residence-tabor	550	unknown	
1	1	italy	bardonecchia	residence-villa-frejus	561	unknown	
2	2	bulgaria	bansko	hotel-mura	566	1100	
3	3	bulgaria	borovets	hotel-samokov	574	75	
4	4	bulgaria	bansko	hotel-lion--bansko	596	800	
...	...	...	...	...	...	...	
402	402	france	val-thorens	hotel-fitz-roy	2216	unknown	
403	403	austria	ischgl	hotel-fliana	2258	unknown	
404	404	austria	ischgl	hotel-elisabeth	2420	unknown	
405	405	austria	ischgl	hotel-trofana-royal	2484	unknown	
406	406	austria	hinterglemm	hotel-alpine-palace	2517	20	

407 rows × 24 columns

```
ski = ski.drop("Unnamed: 0",axis=1)
```

We don't need ID in our clustering model because ID is not an intrinsic characteristic of the each hotel. It is a number that we assign to each hotel out of labeling needs and convenience. A hotel that has an ID 5 is not intrinsically different from a hotel that has an ID 117. ID number is not a meaningful indicator of hotel quality. Based on the ID number alone, we wouldn't be able to tell the difference between each hotels. Another reason is that it doesn't make sense to use vendorID to calculate the distance. Therefore, this number is not relevant and not informative to our clustering model.

```
ski.columns
```

```
Index(['country', 'resort', 'hotel', 'price (£)', 'distance_from_lift_(m)',
       'altitude (m)', 'totalPiste (km)', 'totalLifts', 'gondolas',
       'chairlifts', 'draglifts', 'blues', 'reds', 'blacks', 'totalRuns',
       'link', 'sleeps', 'decSnowLow2020(cm)', 'decSnowHigh2020(cm)',
       'janSnowLow2020(cm)', 'janSnowHigh2020(cm)', 'febSnowLow2020(cm)',
       'febSnowHigh2020(cm)'],
      dtype='object')
```

```
ski.describe()
```

	price (£)	altitude (m)	totalPiste (km)	totalLifts	gondolas	chairlifts	dra
count	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	40
mean	1095.027027	1358.781327	220.270270	60.395577	9.108108	24.766585	2
std	342.841268	508.322847	164.592139	39.025295	8.398517	16.968010	1
min	550.000000	180.000000	0.000000	0.000000	0.000000	0.000000	
25%	839.000000	900.000000	110.000000	30.000000	3.000000	12.000000	1
50%	1021.000000	1441.000000	185.000000	55.000000	8.000000	22.000000	2
75%	1270.500000	1800.000000	282.000000	82.000000	11.000000	35.000000	3
max	2517.000000	2300.000000	1220.000000	206.000000	43.000000	83.000000	11

This function gives us very detailed summary statistics of each varaibles in the dataset. In particular, count, mean, standard deviation, minimum value, 25% threshold, median, 75% threshold, and maximum value for all the columns are made explicit. This can give us the big picture of the whole dataset. By doing this, we could have a general understanding of the values of different column, which is crucial for any further analysis.

```
ski = ski.rename(columns={ski.columns[3]:"price"})
```

I have trouble typing the "british pound" symbol, so I renamed the "price" column.

```
categorical = ski.select_dtypes(include=["object"])
categorical.columns
```

```
Index(['country', 'resort', 'hotel', 'distance_from_lift_(m)', 'link',
       'sleeps', 'decSnowLow2020(cm)', 'decSnowHigh2020(cm)',
       'janSnowLow2020(cm)', 'janSnowHigh2020(cm)', 'febSnowLow2020(cm)',
       'febSnowHigh2020(cm)'],
      dtype='object')
```

```
numerical = ski.select_dtypes(include=["int","float"])
numerical.columns
```

```
Index(['price', 'altitude (m)', 'totalPiste (km)', 'totalLifts', 'gondolas',
       'chairlifts', 'draglifts', 'blues', 'reds', 'blacks', 'totalRuns'],
      dtype='object')
```

```
ski.isna().sum()
```

```
country                  0
resort                   0
hotel                    0
price                    0
distance_from_lift_(m)   0
altitude (m)             0
totalPiste (km)          0
totalLifts               0
gondolas                 0
chairlifts               0
draglifts                0
blues                    0
reds                     0
blacks                   0
totalRuns                0
link                     0
sleeps                   0
decSnowLow2020(cm)       0
decSnowHigh2020(cm)      0
janSnowLow2020(cm)       0
janSnowHigh2020(cm)      0
febSnowLow2020(cm)       0
febSnowHigh2020(cm)      0
dtype: int64
```

There's no NA values in our dataset. We are good to go.

```
unknown_percentage_1 = ski["distance_from_lift_(m)"].value_counts()["unknown"]/len(
unknown_percentage_1
```

```
0.47174447174447176
```

```
ski = ski.drop("distance_from_lift_(m)",axis=1)
```

I noticed that there are too much missing values in the column named "distance_from_lift_(m)." The percetage of missingness is dangerously close to our commonly-used threshold of 50%. When dealing with a column that has this high degree of missingness, imputation could be quite dangerous. Therefore, I will need to remove this column from our dataset.

```
unknown_percentage_2 = ski["sleeps"].value_counts()["unknown"]/len(ski)
unknown_percentage_2
```

```
0.23587223587223588
```

I felt like "sleeps" also have a lot of "unknown"s. But based on the above check, 22.81% of "sleeps" are "unknown". I think that's an acceptable number and will NOT remove it from the dataset.

```
ski = ski.drop("link",axis=1)
```

We won't need any extra external information in our clustering analysis.

```
ski[ski["hotel"]=="hotel-genziana"]
```

	country	resort	hotel	price	altitude (m)	totalPiste (km)	totalLifts	gondolas
328	italy	ortisei-st-ulrich	hotel-genziana	1349	1236	176	0	0

1 rows × 21 columns

```
zero_counts = (numerical==0).sum(axis=1)
zero_percentage = zero_counts/numerical.shape[1]
ski = ski.drop(ski[zero_percentage>0.4].index,axis=0)
len(ski)
```

```
400
```

There's a lot of rows that have too many 0s for numerical columns. For example, row 308 (hotel-wirlerholf) and row 328 (hotel-genziana) and many more. They have too many missing numerical values that they are not helpful in building the model. Therefore, I removed every row that are missing more than 50% of numerical values. After removing these rows, we are left with 400 rows in the dataframe.

```
ski[ski["hotel"]=="hotel-euroski"]
```

	country	resort	hotel	price	altitude (m)	totalPiste (km)	totalLifts	gondolas	c
46	andorra	soldeu	hotel-euroski	752	1800	210	67	4	

1 rows × 21 columns

```
unknown_counts = categorical.apply(lambda row: row.str.count("unknown").sum(),axis=
unknown_percentage = unknown_counts/categorical.shape[1]
ski = ski.drop(ski[unknown_percentage>0.4].index, axis=0)
len(ski)
```

```
<ipython-input-20-587accc05b93>:3: UserWarning: Boolean Series key will be re
  ski = ski.drop(ski[unknown_percentage>0.4].index, axis=0)
346
```

There's a lot of rows that have many "unknown"s for nearly all of the categorical columns. For example, row 46 (hotel-euroski) and row 47 (soldeu-maistre) and many more. These rows have too many "unknown"s towards the last few columns. Therefore, I need to drop all the rows that have more than 50% of categorical values being "unknown." After removing those rows, we are left with 346 rows in our dataframe.

```
ski
```

	country	resort	hotel	price	altitude (m)	totalPiste (km)	totalLifts	gon
0	italy	bardonecchia	residence-tabor	550	1312	140	23	
1	italy	bardonecchia	residence-villa-frejus	561	1312	140	23	
2	bulgaria	bansko	hotel-mura	566	935	70	24	
3	bulgaria	borovets	hotel-samokov	574	1390	58	18	
4	bulgaria	bansko	hotel-lion--bansko	596	935	70	24	
...	...	...	...	...	...	...	...	
402	france	val-thorens	hotel-fitz-roy	2216	2300	600	183	
403	austria	ischgl	hotel-fliana	2258	1400	230	48	
404	austria	ischgl	hotel-elisabeth	2420	1400	230	48	
405	austria	ischgl	hotel-trofana-royal	2484	1400	230	48	
406	austria	hinterglemm	hotel-alpine-palace	2517	1003	200	54	

346 rows × 21 columns

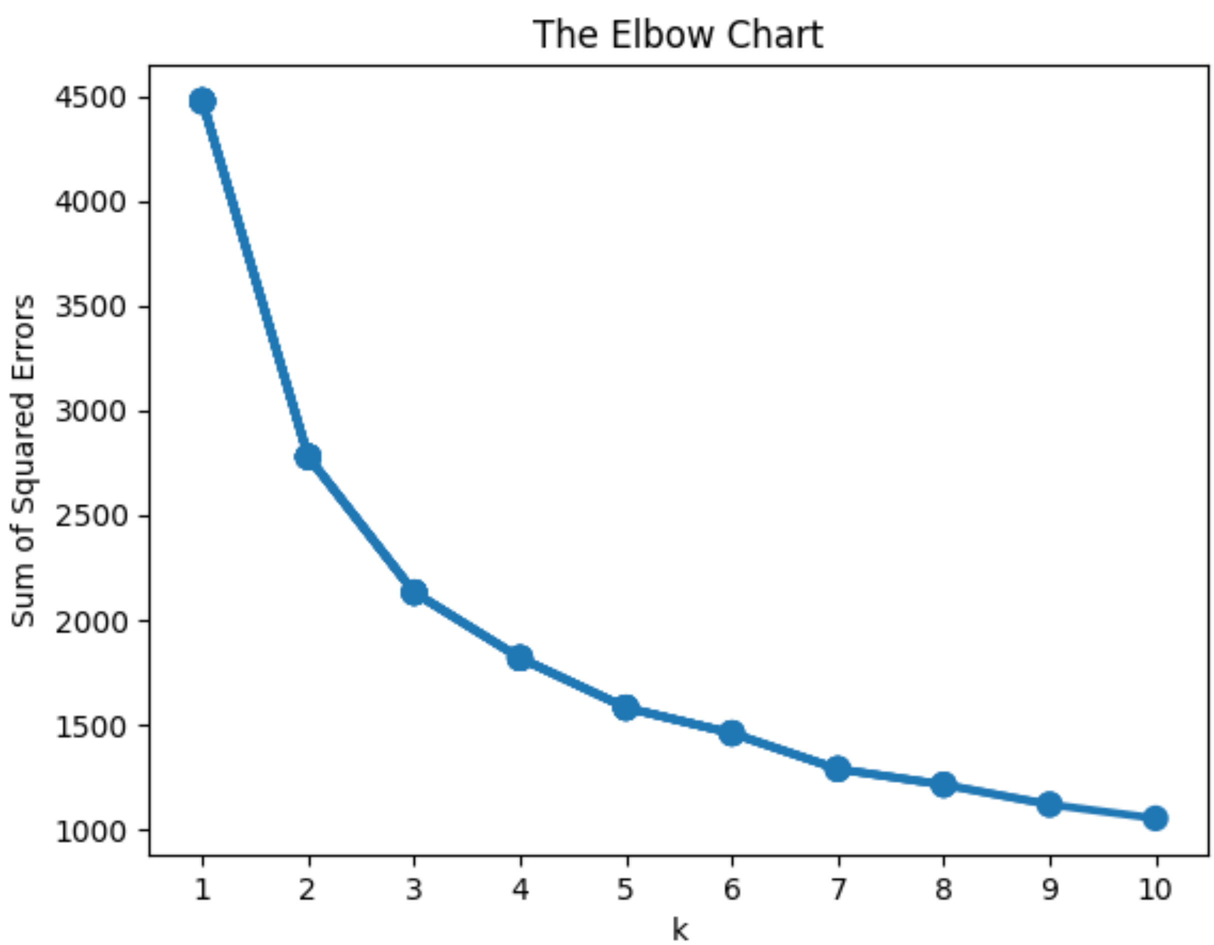
```
scaler = StandardScaler()
numerical_normalized = pd.DataFrame(scaler.fit_transform(numerical),columns=numeric
```

We need to standardize all the numerical variables before we build the clustering model. Notice how variables like "price" and "attitude" could take much higher values than other variables. Therefore, scaling would be needed, otherwise the model result would be dominated by variables that take larger values and therefore be biased. By standardizing, all the numerical variables have "equal say" in the model and we are good to go.

Part 2: K-Means Clustering

```
sse={}
for k in range (1,11):
  kmeans = KMeans(n_clusters=k, random_state=53097367)
  kmeans.fit(numerical_normalized)
  sse[k]=kmeans.inertia_
  plt.title("The Elbow Chart")
  plt.xlabel("k")
  plt.ylabel("Sum of Squared Errors")
  sns.pointplot(x=list(sse.keys()),y=list(sse.values()));
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
```



Based on the elbow chart, I would go with having 3 clusters. We see a sharp decrease in SSE from k=2 to k=3. Therefore, k=3 is better than k=2. Notice how there's an "elbow" (a sharp turning point) at k=3.

I can foresee that other people might argue that k=4 is a good idea. However, notice how k=3, k=4, and k=5 is almost on a straight line, suggesting that when k increase from (3 to 4) and (4 to 5), the SSE decreases as a constant rate. This decrease is not impressive at all compare to the sharp decrease in SSE from k=2 to k=3. Therefore, when somebody argue that k=4 is a better idea, somebody else could follow the same logic and say that k=5 is equally good. That argument could go on and on. Therefore, to avoid such argument, I would go proceed with having 3 clusters.

```
kmeans = KMeans(n_clusters=3, random_state= 53097367)
kmeans.fit(numerical_normalized)
cluster_labels = kmeans.labels_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
```

I pick that seed number because it's my BU ID number.

```
kmeans3 = numerical_normalized.assign(Cluster=cluster_labels)
grouped_ski= kmeans3.groupby("Cluster")[numerical_normalized.columns]
pd.set_option("display.max_columns",None)
grouped_ski.describe()
```

	price							
	count	mean	std	min	25%	50%	75%	max
Cluster								
0	199.0	-0.201454	0.960628	-1.591692	-0.908321	-0.426456	0.274438	4.152718
1	46.0	0.661390	0.977445	-1.092305	0.118197	0.516830	1.187790	3.273680
2	162.0	0.059663	0.973297	-1.442752	-0.594379	-0.150479	0.542384	4.056345

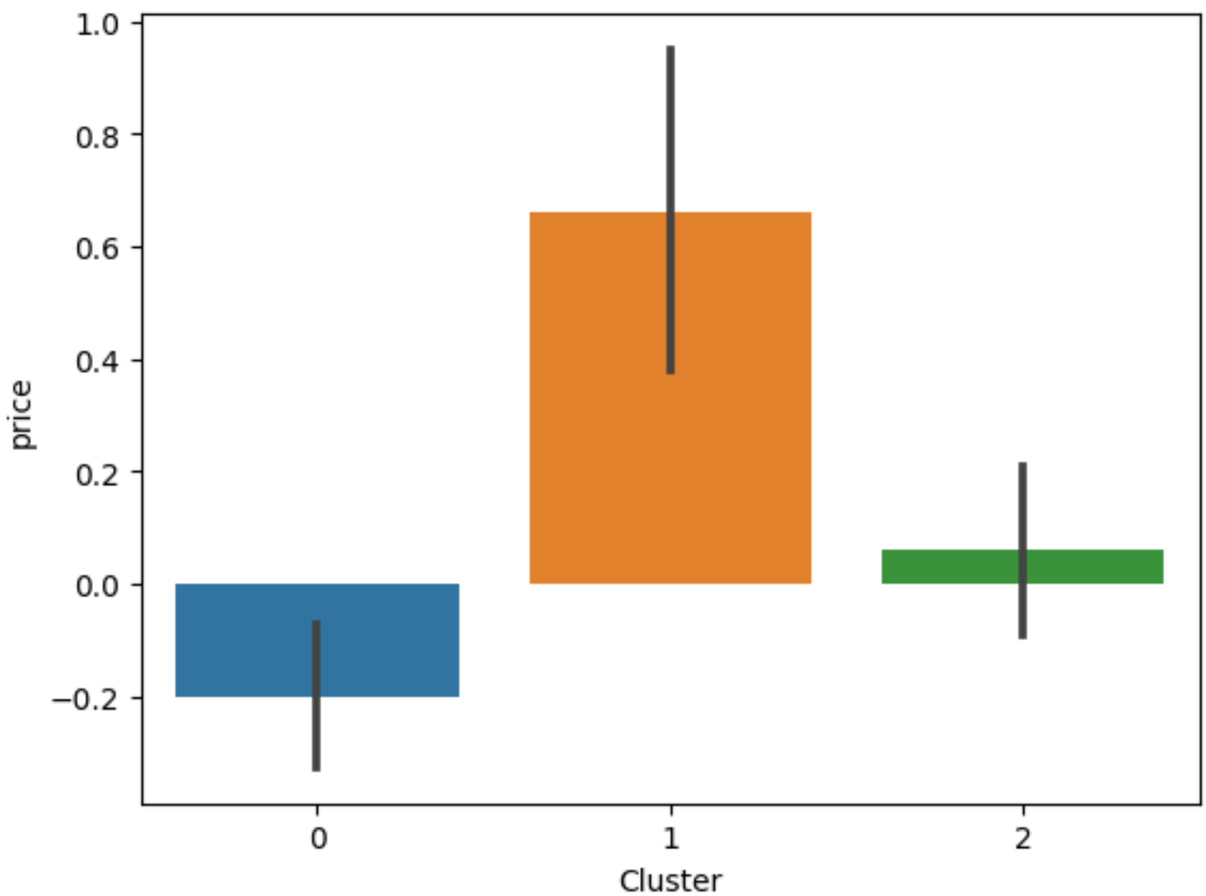
```
kmeans3["Cluster"].value_counts()
```

```
0    199
2    162
1     46
Name: Cluster, dtype: int64
```

Part 3: Visualization

```
#Graph 1
sns.barplot(x="Cluster", y="price", data=kmeans3)
```

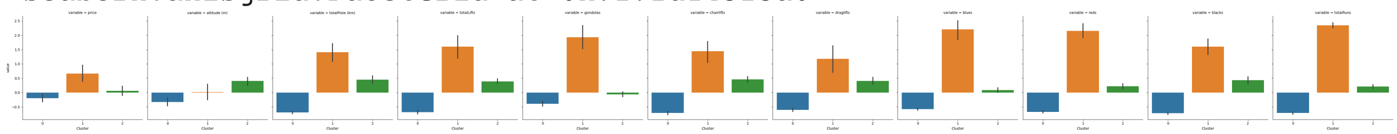
```
<Axes: xlabel='Cluster', ylabel='price'>
```



This barplot provides a side by side comparison for values of "price" across all three clusters. It is indisputable that the price information is one of the most important characteristics of any hotel. It suggests that on average, cluster 1 hotels are the most expensive among three clusters and cluster 3 hotels are the cheapest.

```
#Graph Series
data_melted = pd.melt(kmeans3, id_vars=["Cluster"],var_name="variable",value_name='
sns.catplot(x="Cluster",y="value",col="variable",data=data_melted,kind="bar")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f72d143f3a0>
```



IMPORTANT: PLEASE CLICK ON THE GRAPH TO ENLARGE THEM!!!!!!!!!!!!!!

- Small versions good for capturing the general trend.
- Enlarged versions good for the actual analysis.

This gives us the big picture. For the attributes that would positively contribute to price, Cluster 1 seems to be always above average. Examples include the number of lifts, the number of tracks of various levels, total Piste, etc. Cluster 0 seems to be below average all the time.

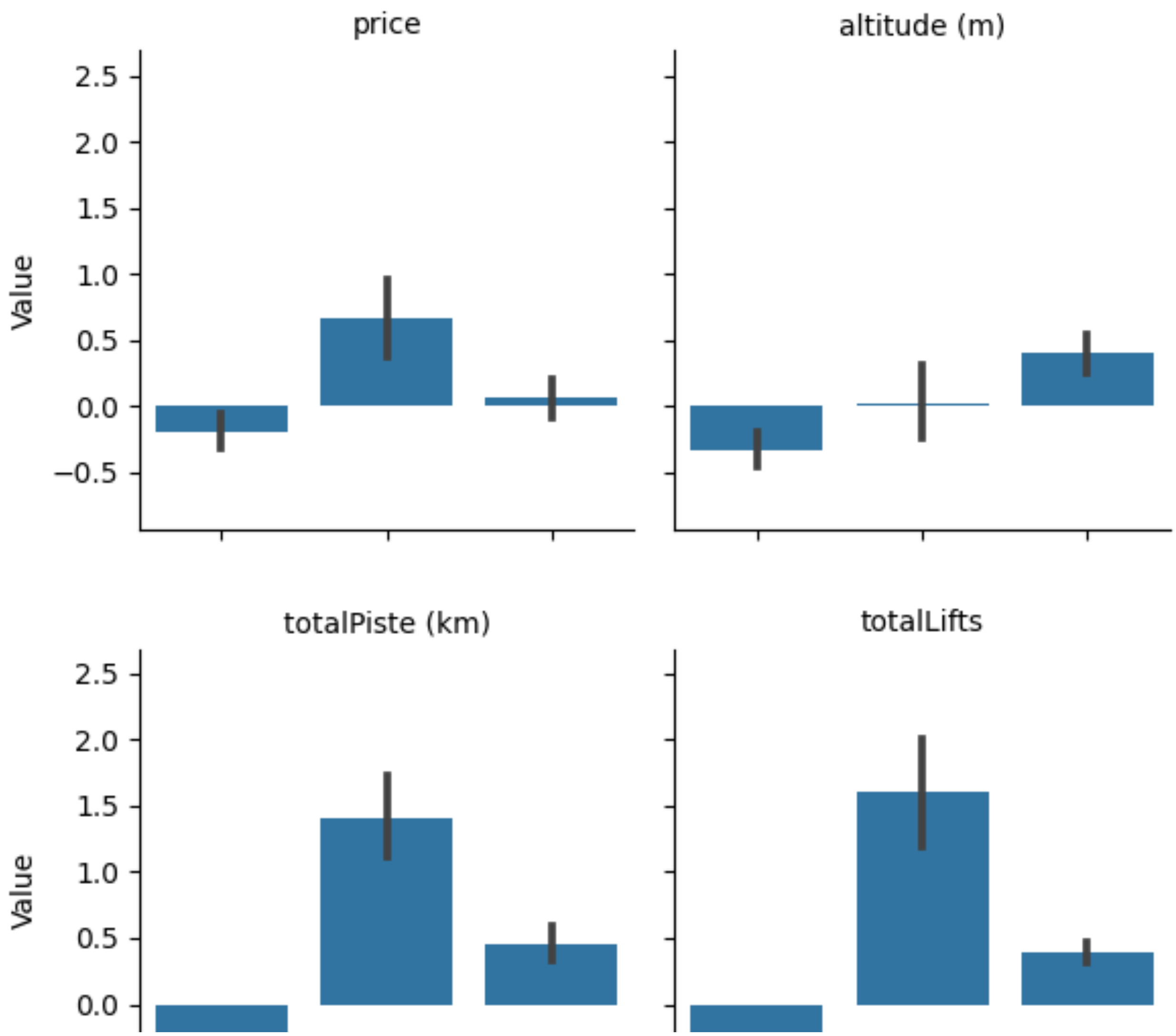
One special feature is "altitude." Cluster 1 seems to be in between most of the time. One metric that doens't follow this rule is the "altitude." In terms of where the hotels are located, it seems that cluster 0 hotels on average are at low altitude wheres cluster 1 hotels on average are located at high altitude. Cluster 1 seems to be at the sweat spot: not too high and not too low. People definitely find the location of cluster 1 hotels to be really convenient. Not high above the mountain, not low in the valley, just a nice place at the appropriate altitude that's really convenient to drive to.

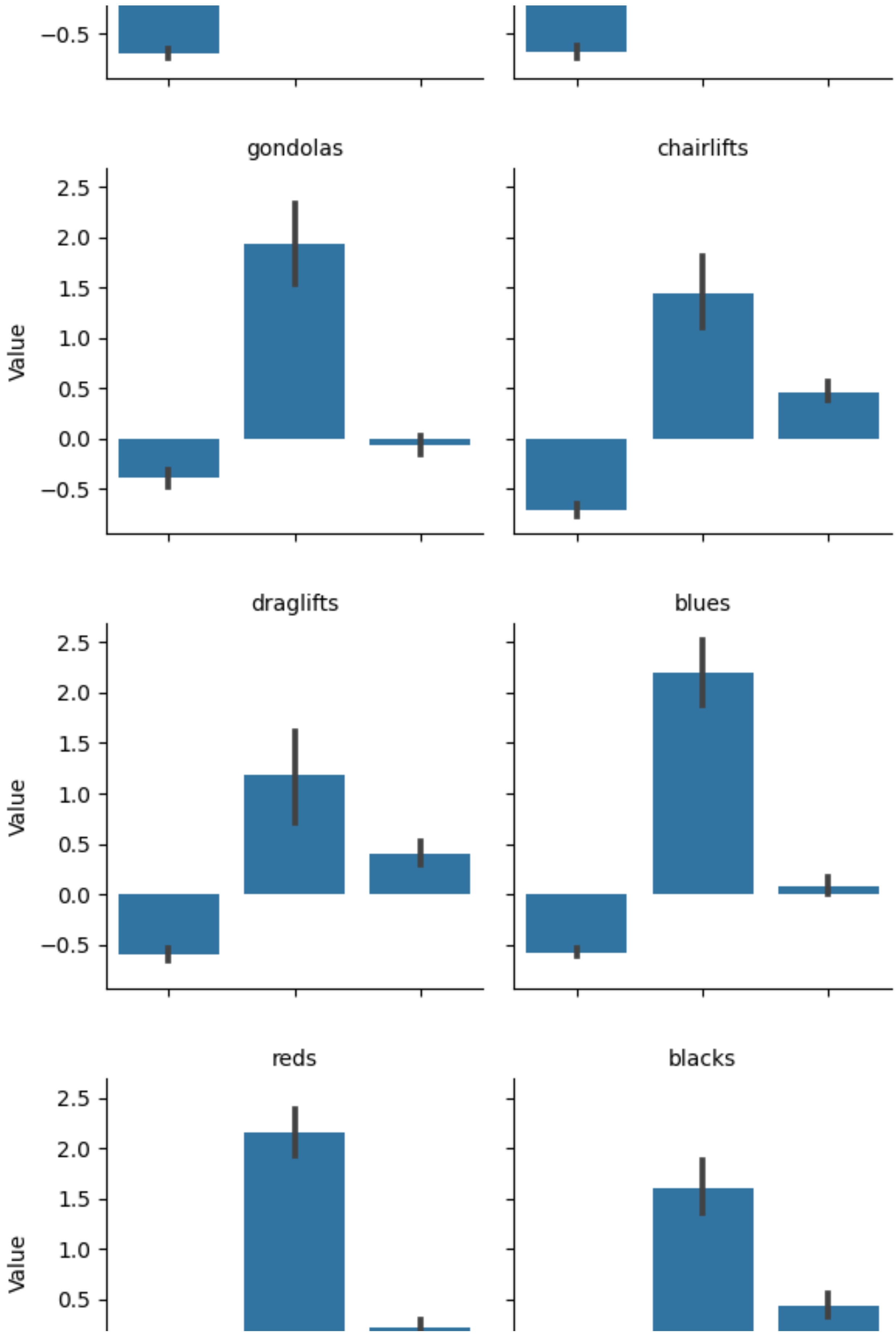
This intuitively justify the price points for all three clusters.

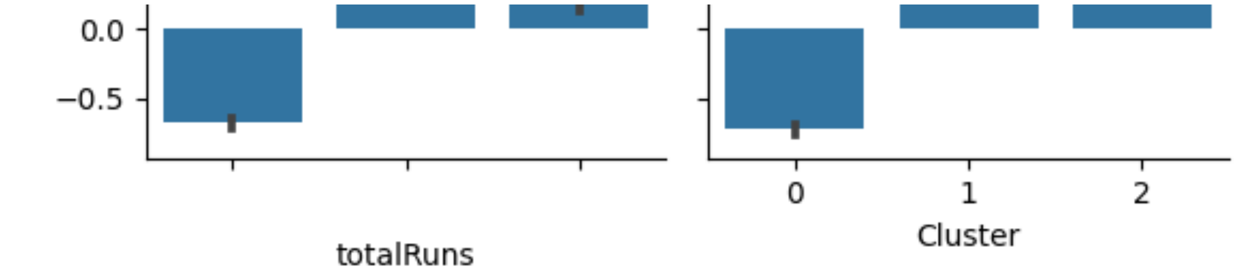
If you are viewing the pdf file, You will not be able to click to enlarge the above visualization. Therefore, I generated the below visualization as the second best option. The leftmost bars always represents cluster 0. The middle bar always represents cluster 1. The rightmost bar always represents cluster 2.

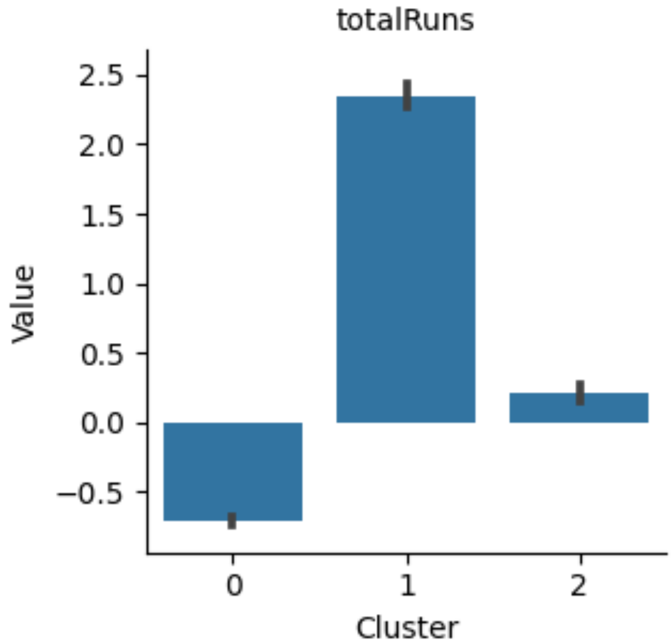
```
data_melted=pd.melt(kmeans3,id_vars=["Cluster"],var_name="variable",value_name="val
g=sns.FacetGrid(data_melted,col="variable",col_wrap=2)
g.map(sns.barplot,"Cluster","value",hue_order=[0,1,2])
g.set_titles("{col_name}")
g.set_xlabels("Cluster")
g.set_ylabels("Value")
sns.despine()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:712: UserWarning:
  warnings.warn(warning)
```

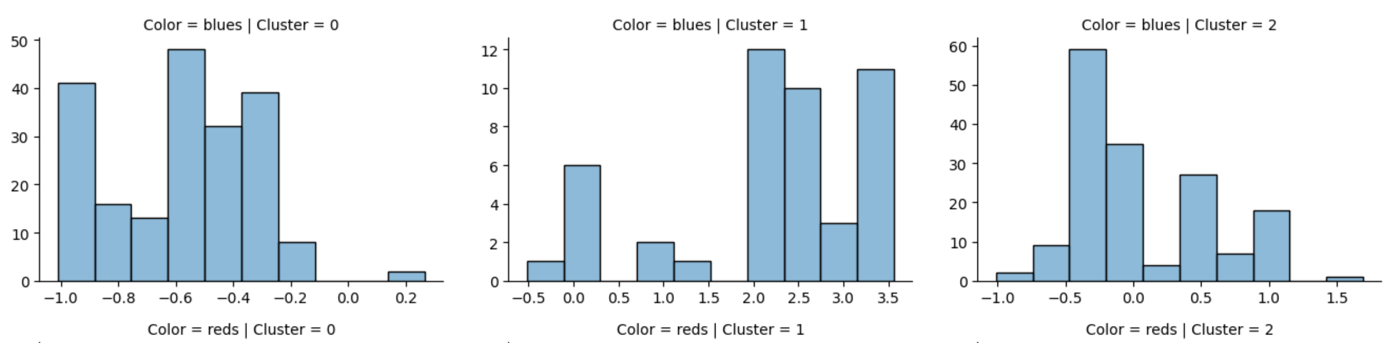


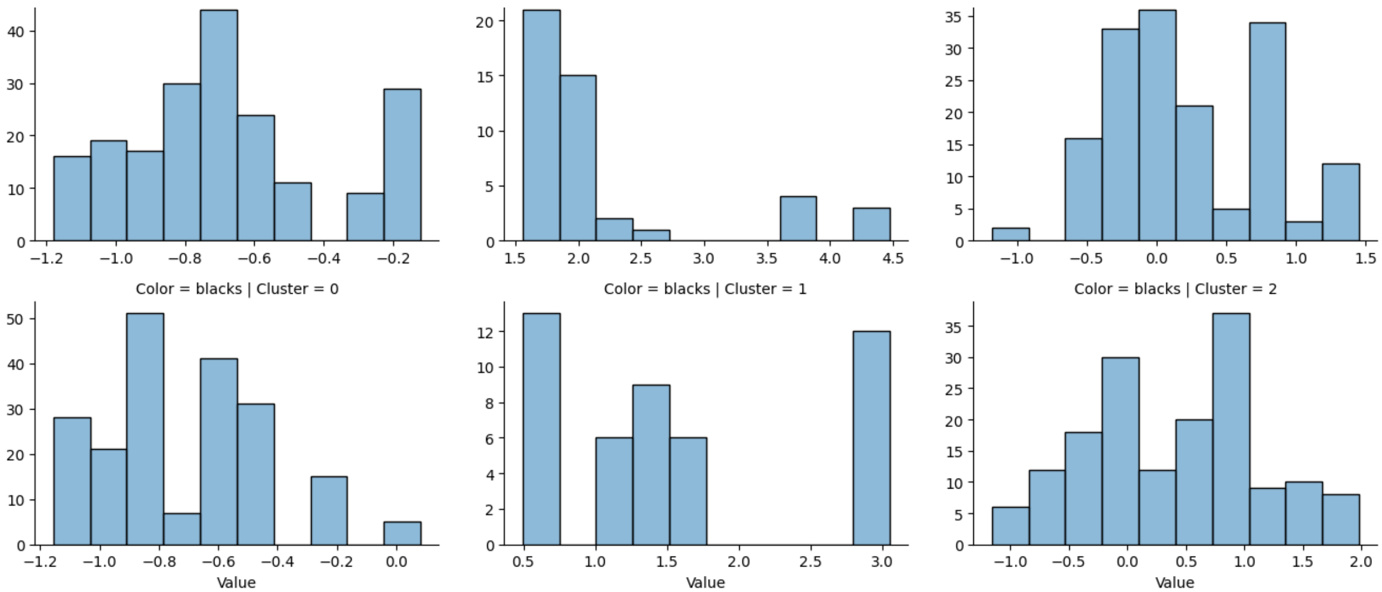

−0.5
gondolas
chairlifts
draglifts
blues
reds
blacks
Value
2.5
2.0
1.5
1.0
0.5
0.0
−0.5





```
#Graph 2
df_long = pd.melt(kmeans3,id_vars=["Cluster"],value_vars=["blues","reds","blacks"],
g = sns.FacetGrid(df_long,col="Cluster",row="Color",height=3, aspect=1.5, sharex=Fa
def plot_facet(x,**kwargs):
  sns.histplot(x,**kwargs)
g.map(plot_facet,"Value",alpha=0.5,bins=10)
g.set_axis_labels("Value","")
plt.show()
```





There's multiple piece of information in this graph. This graph is meant to be an insider graph, not to show to any exectutives, etc. Therefore, this graph will only be included in this report, but not the actual presentation.

This graph provides a side by side comparison of the number of each tracks for each cluster. We have 3 clusters (0,1,2) and 3 types of track (blue, red, black.) The first column represents Cluster 0. The second column represents Cluster 1. The thrid column represents Cluster 2. The first row represents the number of black tracks. The second row represents the number of red tracks. The third row represents the number of black tracks.

I prefer to read this graph horizontally. First of all, let's examine the number of blue tracks across clusters. Notice the distribution of data for each cluster and keep in mind that the data have already been normalized. First, pay attention to the x-axis.

It seems that cluster 0 hotels have few blue tracks in general, because the x-values range from -1.0 to 0.2. Notice how the data is right skewed, meaning more hotels clustered in the even lower end of that spectrum. For cluster 1 hotels, the x-range is (-0.5,3.5), which is in itself a much higher range compared to cluster 0. Furthermore, notice how the data is left skewed, suggesting more hotels closer to the 3.5 side of the spectrum. For cluster 2, notice how the x-value are within (-1.0, 1.5) and the data is right skewed, meaning more hotels are at the lower end of that spectrum. Cluster 0 has the lowest possible values for the number of blue tracks and most hotels are at the lower end of that already low enough spectrum. Cluster 3 has slightly higher x-range, and most hotels still cluster at the lower side of that spectrum. This means that cluster 2 in general have more blue tracks than cluster 0 hotels, but not by much. Cluster 1 however, have much higher x-range than the other 2 and most cluster 2 hotels are seen at the higher side of that spectrum, indicating that cluster 2 hotels in general have most blue tracks among all three clusters.

The same analysis could be done on the second and third row of the graphs. We could see that for red tracks, most cluster 2 hotels are at the lower end of a much higher spectrum, still suggesting that they on average have the most red tracks. The logic is that worst students at Harvard are still better than best students at some chicken colleges. By the x-range, we could also easily tell that cluster 2 hotels on average have more red tracks than average cluster 0 hotels. The same conclusion could be made on the number of black tracks.

Therefore, based on this graph, we could conclude that cluster 1 hotels on average have the most blue, red, and black tracks. Cluster 0 hotels on average have the least blue, red, and black tracks. Therefore, this finding is consistent with the fact that cluster 2 hotels are the most luxiourious of all and could charge the highest price.

Name the Clusters

Cluster 0: Affordable Valley Inn

The above analysis suggests that cluster 0 hotels have the least number of various lifts and shortest piste and least number of various tracks. These feed into the low price points of cluster 0 hotels. Therefore, cluster 0 hotels are prefect for those who don't mind sacrificing luxury accomodations in exchange for a more affordable price. However, these hotels are not appealing to more advanced skiers due to its limited amount of all sorts of equipment. Also notice that cluster 0 hotels on average locate at low altitude. These characteristics are all reflected in the naming of the cluster by using the word "affordable" "valley" and "Inn".

Cluster 1: Ski Majesty Grand

Based on the above analysis, it's pretty clear that cluster 3 hotels outwin other clusters in all attributes that positively contribute to the hotel quality. For example, cluster 1 have the most amount of various tracks, most amount of various lifts, etc. Therefore, their high price points are well justified. Cluster 1 hotels are perfect for skiers who priortize variety and options in their skiing experience. The luxiousness is reflected in the diction "majesty" and "grand."

Cluster 2: Ski Summit Lodge

Cluster 2 hotels are like the perfect middle ground. It's not as expensive as cluster 2 hotels and not as shabby as cluster 0 hotels. They are the average ones in the ski hotel world, nothing stand out, but nothing too bad. Can say nothing good and also nothing bad at them. One thing that's special about them is that cluster 2 hotels tend to locate at high altitude. Therefore, people need to have some sort of vehicle to take them to these hotels. The target customers of cluster 2 hotels are skiers who enjoy a middle ground in terms of pricing and amenities and have some sort of vehicle. In the name, the word "summit" captures the high altitude and the word "lodge" is consistent with the quality and the price point.

✓ 5s completed at 8:55 PM

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn import preprocessing


amenities = pd.read_csv("/Users/xikexin/Desktop/hotel_amenities.csv")
cost = pd.read_csv("/Users/xikexin/Desktop/amenity_costs.csv")
```

1. Import & Inspect the dataset

In [2]:
```
amenities.head()
```

Out[2]:

	WiFi_Network	breakfast	parking	gym	flex_check	shuttle_bus	air_pure	jacuzzi
0	Basic	NaN	Valet	NaN	No	No	No	No
1	Basic	NaN	Valet	NaN	No	No	No	No
2	Basic	NaN	Valet	NaN	No	No	No	No
3	Basic	NaN	Valet	NaN	No	No	No	No
4	Basic	NaN	Valet	NaN	No	No	No	No

In [3]:
```
amenities['avg_rating'].describe()
```

Out[3]:
```
count    6912.000000
mean        7.370858
std         1.656300
min         2.130000
25%         6.347500
50%         7.970000
75%         8.280000
max        10.000000
Name: avg_rating, dtype: float64
```

In [4]:
```
amenities2 = pd.get_dummies(amenities, drop_first = True,
                         columns = ['WiFi_Network', 'breakfast', 'parking',
amenities2 = amenities2.replace({True: 1, False: 0})
amenities2
```

Out[4]:

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

6912 rows × 14 columns

In [5]:
```
amenities2.columns
```

Out[5]:
```
Index(['avg_rating', 'WiFi_Network_Best in Class', 'WiFi_Network_Strong',
       'breakfast_Full Buffet', 'parking_Valet', 'gym_Basic', '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 [6]:
```
X = amenities2[['WiFi_Network_Best in Class', 'WiFi_Network_Strong',
       'breakfast_Full Buffet', 'parking_Valet', 'gym_Basic', 'gym_Super',
       'flex_check_Yes', 'shuttle_bus_Yes', 'air_pure_Yes', 'jacuzzi_Yes',
       'VIP_shop_Yes', 'pool_temp_80', 'pool_temp_84']]
X = sm.add_constant(X)
Y = amenities2.avg_rating
linearRegression = sm.OLS(Y, X).fit()
linearRegression.summary()
```

Out[6]:

OLS Regression Results

Dep. Variable:	avg_rating	**R-squared:**	0.272
Model:	OLS	**Adj. R-squared:**	0.271
Method:	Least Squares	**F-statistic:**	198.4
Date:	Sun, 07 May 2023	**Prob (F-statistic):**	0.00
Time:	22:04:39	**Log-Likelihood:**	-12197.
No. Observations:	6912	**AIC:**	2.442e+04
Df Residuals:	6898	**BIC:**	2.452e+04
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.3308	0.060	88.627	0.000	5.213	5.449
WiFi_Network_Best in Class	1.7268	0.042	41.438	0.000	1.645	1.809
WiFi_Network_Strong	1.1877	0.042	28.501	0.000	1.106	1.269
breakfast_Full Buffet	0.6140	0.036	17.012	0.000	0.543	0.685
parking_Valet	0.0937	0.034	2.753	0.006	0.027	0.160
gym_Basic	-0.0621	0.042	-1.490	0.136	-0.144	0.020
gym_Super	0.1286	0.042	3.086	0.002	0.047	0.210
flex_check_Yes	0.4782	0.034	14.055	0.000	0.412	0.545
shuttle_bus_Yes	0.4199	0.034	12.342	0.000	0.353	0.487
air_pure_Yes	0.0753	0.034	2.212	0.027	0.009	0.142
jacuzzi_Yes	0.1839	0.034	5.405	0.000	0.117	0.251
VIP_shop_Yes	0.2179	0.034	6.405	0.000	0.151	0.285
pool_temp_80	0.0747	0.042	1.794	0.073	-0.007	0.156
pool_temp_84	0.2638	0.042	6.331	0.000	0.182	0.345

Omnibus:	162.829	**Durbin-Watson:**	1.936
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	174.003
Skew:	-0.382	**Prob(JB):**	1.64e-38
Kurtosis:	3.144	**Cond. No.**	7.90

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

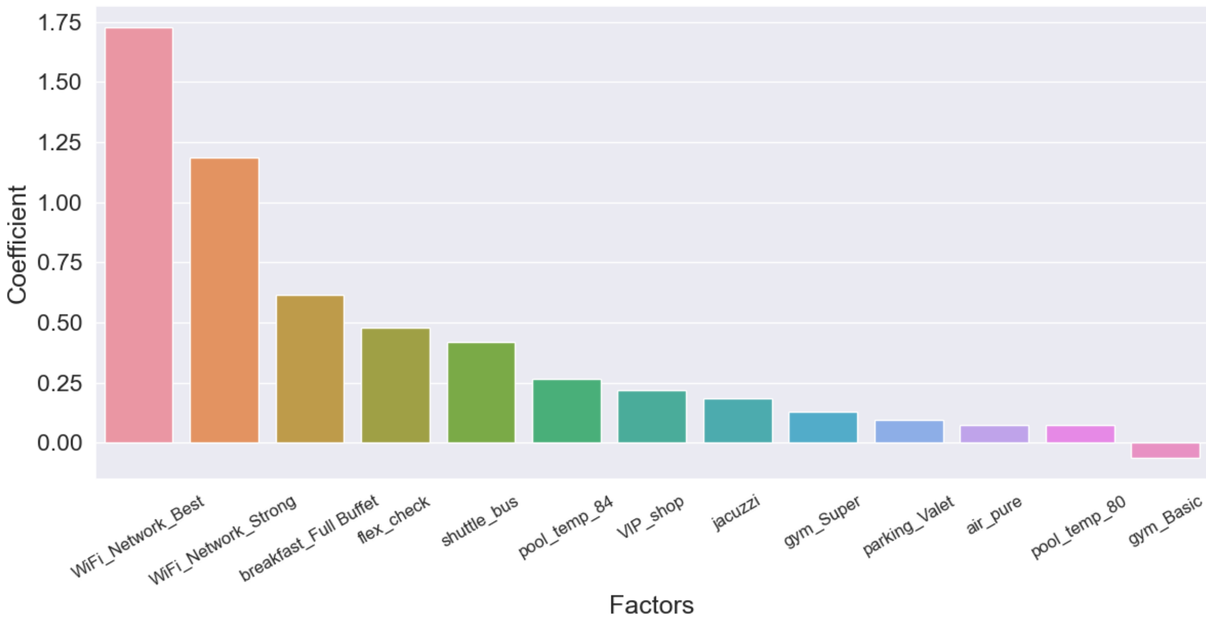
2. Visualization of the importance of factors

In [7]:
```
data = {'Factors': ['WiFi_Network_Best', 'WiFi_Network_Strong',
                    'breakfast_Full Buffet', 'flex_check', 'shuttle_bus','po
                    'VIP_shop', 'jacuzzi', 'gym_Super',
                    'parking_Valet', 'air_pure', 'pool_temp_80', 'gym_Basic'
        'Coefficient': [1.726814, 1.187700, 0.613961, 0.478220, 0.419939, 0.

coef_df = pd.DataFrame(data)
```

In [8]:
```
from sklearn.linear_model import LinearRegression
import warnings
from sklearn.exceptions import DataConversionWarning

#create a coefficient histogram of each factor
plt.figure(figsize=(14,6))
sns.set(rc={'figure.figsize':(30.7,8.27)})
sns.set(font_scale=1.5)
sns.barplot(x = 'Factors', y = 'Coefficient', ci = None, data = coef_df)
plt.xticks(rotation=30, fontsize=12)
```

```
/var/folders/tl/q4v56tn94kx3m91j9_7fp7340000gn/T/ipykernel_26781/1344825829.p
y:9: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x = 'Factors', y = 'Coefficient', ci = None, data = coef_df)
```

Out[8]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
 [Text(0, 0, 'WiFi_Network_Best'),
  Text(1, 0, 'WiFi_Network_Strong'),
  Text(2, 0, 'breakfast_Full Buffet'),
  Text(3, 0, 'flex_check'),
  Text(4, 0, 'shuttle_bus'),
  Text(5, 0, 'pool_temp_84'),
  Text(6, 0, 'VIP_shop'),
  Text(7, 0, 'jacuzzi'),
  Text(8, 0, 'gym_Super'),
  Text(9, 0, 'parking_Valet'),
  Text(10, 0, 'air_pure'),
  Text(11, 0, 'pool_temp_80'),
  Text(12, 0, 'gym_Basic')])
```



In [9]:

```
conjoint_attributes = ['WiFi_Network_Best in Class', 'WiFi_Network_Strong',
       'breakfast_Full Buffet', 'parking_Valet', 'gym_Basic', 'gym_Super',
       'flex_check_Yes', 'shuttle_bus_Yes', 'air_pure_Yes', 'jacuzzi_Yes',
       'VIP_shop_Yes', 'pool_temp_80', 'pool_temp_84']
level_name = []
part_worth = []
part_worth_range = []
end = 1
for item in conjoint_attributes:
    nlevels = len(list(set(amenities2[item])))
    level_name.append(list(set(amenities2[item])))
    begin = end
    end = begin + nlevels - 1
    new_part_worth = list(linearRegression.params[begin:end])
    new_part_worth.append((-1) * sum(new_part_worth))
    part_worth_range.append(max(new_part_worth) - min(new_part_worth))
    part_worth.append(new_part_worth)
    # end set to begin next iteration

attribute_importance = []
for item in part_worth_range:
    attribute_importance.append(round(100 * (item / sum(part_worth_range)),2

effect_name_dict = {u'WiFi_Network_Best in Class':u'WiFi_Network_Best in Cla
                    u'WiFi_Network_Strong':u'WiFi_Network_Strong',
                    u'breakfast_Full Buffet':u'breakfast_Full Buffet',
                    u'parking_Valet':u'parking_Valet',
                    u'gym_Basic':u'gym_Basic',
                    u'gym_Super':u'gym_Super',
                    u'flex_check_Yes':u'flex_check_Yes',
                    u'shuttle_bus_Yes':u'shuttle_bus_Yes',
                    u'air_pure_Yes':u'air_pure_Yes',
                    u'jacuzzi_Yes':u'jacuzzi_Yes',
                    u'VIP_shop_Yes':u'VIP_shop_Yes',
                    u'pool_temp_80':u'pool_temp_80',
                    u'pool_temp_84':u'pool_temp_84'}
```

```
#print out parthworth's for each level
estimates_of_choice = []
index = 0
for item in conjoint_attributes :
    print ("\n Attribute : " , effect_name_dict[item])
    print ("\n Importance : " , attribute_importance[index])
    print('    Level Part-Worths')
    for level in range(len(level_name[index])):
        print('        ',level_name[index][level], part_worth[index][level])
    index = index + 1
```

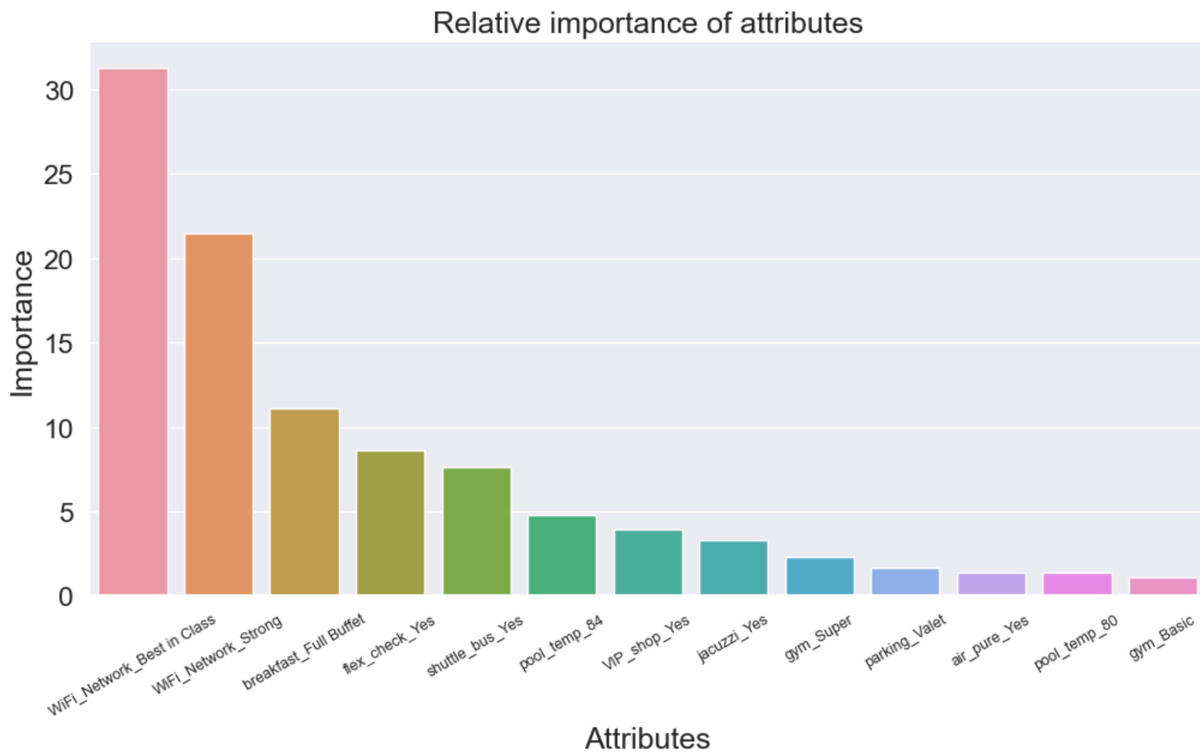
```
Attribute :  WiFi_Network_Best in Class

Importance :  31.25
   Level Part-Worths
       0 1.7268142361109553
       1 -1.7268142361109553

Attribute :  WiFi_Network_Strong

Importance :  21.49
   Level Part-Worths
       0 1.1876996527775177
       1 -1.1876996527775177

Attribute :  breakfast_Full Buffet

Importance :  11.11
   Level Part-Worths
       0 0.6139605034721922
       1 -0.6139605034721922

Attribute :  parking_Valet

Importance :  1.7
   Level Part-Worths
       0 0.09367766203704118
       1 -0.09367766203704118

Attribute :  gym_Basic

Importance :  1.12
   Level Part-Worths
       0 -0.062071759259264725
       1 0.062071759259264725

Attribute :  gym_Super

Importance :  2.33
   Level Part-Worths
       0 0.12860532407406808
       1 -0.12860532407406808

Attribute :  flex_check_Yes

Importance :  8.65
   Level Part-Worths
       0 0.47822048611110646
       1 -0.47822048611110646

Attribute :  shuttle_bus_Yes

Importance :  7.6
   Level Part-Worths
       0 0.41993923611111306
       1 -0.41993923611111306
```

```
Attribute :  air_pure_Yes

Importance :  1.36
   Level Part-Worths
       0 0.07525752314818634
       1 -0.07525752314818634

Attribute :  jacuzzi_Yes

Importance :  3.33
   Level Part-Worths
       0 0.18390914351849885
       1 -0.18390914351849885

Attribute :  VIP_shop_Yes

Importance :  3.94
   Level Part-Worths
       0 0.21792534722222845
       1 -0.21792534722222845

Attribute :  pool_temp_80

Importance :  1.35
   Level Part-Worths
       0 0.07474392361111058
       1 -0.07474392361111058

Attribute :  pool_temp_84

Importance :  4.77
   Level Part-Worths
       0 0.2638064236111158
       1 -0.2638064236111158
```

In [10]:

```
#Histogram of the relative importance of the factor attributes
plt.figure(figsize=(12,6))
sns.barplot(x=conjoint_attributes,y=attribute_importance, order=sorted(conjo
plt.title('Relative importance of attributes')
plt.xlabel('Attributes')
plt.ylabel('Importance')
plt.xticks(rotation=30, fontsize=9)
```

Out[10]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
 [Text(0, 0, 'WiFi_Network_Best in Class'),
  Text(1, 0, 'WiFi_Network_Strong'),
  Text(2, 0, 'breakfast_Full Buffet'),
  Text(3, 0, 'flex_check_Yes'),
  Text(4, 0, 'shuttle_bus_Yes'),
  Text(5, 0, 'pool_temp_84'),
  Text(6, 0, 'VIP_shop_Yes'),
  Text(7, 0, 'jacuzzi_Yes'),
  Text(8, 0, 'gym_Super'),
  Text(9, 0, 'parking_Valet'),
  Text(10, 0, 'air_pure_Yes'),
  Text(11, 0, 'pool_temp_80'),
  Text(12, 0, 'gym_Basic')])
```



3. Consider the cost

Amenity	Level	Cost	Coefficient
WiFi_Network	Basic	11.75	
WiFi_Network	Strong	16.25	1.726814
WiFi_Network	Best in Class	19.15	1.1877
breakfast	None	0	
breakfast	Continental	13.25	
breakfast	Full Buffet	22.45	0.613961
parking	Valet	60	0.093678
parking	Open Lot	15	
gym	None	0	
gym	Basic	10	-0.062072
gym	Advanced	35	
gym	Super	65	0.128605
flex_check	No	0	
flex_check	Yes	12	0.47822
shuttle_bus	No	0	
shuttle_bus	Yes	75	0.419939
air_pure	No	0	
air_pure	Yes	12.85	0.075258
jacuzzi	No	0	
jacuzzi	Yes	40	0.183909
VIP_shop	No	0	
VIP_shop	Yes	12	0.217925
pool temp	76	15	
pool temp	80	35	0.074744
pool temp	84	45	0.263806

a:	Best Choice	363.45
b:	-air_pure	-12.85
	-parking_valet	-45
		305.6
c:	-gym_super	-65
	+gym_advanced	35
	-jacuzzi	-40
	+air_pure	12.85
		248.45

After conjoint analysis, we get the contribution of each amenity to the rating of the hotel and now introduced a cost limit of $250 to determine the final choice of amenities for the hotel. Firstly, we set the best choice as the reference, when each amenity is rated we choose the best one, which costs 363.45 dollars per room, per night. Next, according to the coefficient histogram, we change the amenities that have the least impact on the rating in descending order. After we cancel air_pure and change the parking valet to parking open lot, the cost is 305.6 dollars per room, per night. After we continue to change gym_super to gym_advanced cancel jacuzzi and add the air_pure, the cost is 248.45 dollars per room, per night, which meets our cost requirement. Therefore, the final amenities level of our proposed hotel is as follows:

Amenity	Level	Cost
WiFi_Netwo	Best in Class	19.15
breakfast	Full Buffet	22.45
parking	Open Lot	15
gym	Advanced	35
flex_check	Yes	12
shuttle_bus	Yes	75
air_pure	Yes	12.85
VIP_shop	Yes	12
pool temp	84	45
	Total cost	248.45

4. Use code to find the best amenities combination

In [11]:

```
# Merge the amenities and costs datasets
merged_amenities = amenities.copy()
for _, row in cost.iterrows():
    amenity = row["Amenity"].replace(" ", "_")
    level = row["Level"]
    estimated_cost = row["Estimated Incremental Cost,\nPer Visitor/Per Night

    if amenity == "pool_temp":
        level = int(level)  # Convert the pool_temp level to an integer

    merged_amenities.loc[merged_amenities[amenity] == level, amenity + "_cos

# Calculate the total cost per amenity combination
cost_columns = [col for col in merged_amenities.columns if "_cost" in col]
merged_amenities["total_cost"] = merged_amenities[cost_columns].sum(axis=1)

# Filter the merged dataset based on the cost constraint ($250 per room per
filtered_amenities = merged_amenities[merged_amenities["total_cost"] <= 250]

# Analyze the average ratings of the filtered dataset
filtered_amenities = filtered_amenities.sort_values(by="avg_rating", ascendi

# Recommend the amenity combination with the highest average rating
recommended_amenities = filtered_amenities.head()
recommended_amenities[["WiFi_Network", "breakfast", "parking", "gym", "flex_
```

Out[11]:

	WiFi_Network	breakfast	parking	gym	flex_check	shuttle_bus	air_pure	ja
4596	Strong	Full Buffet	Open Lot	Super	Yes	Yes	Yes	
6255	Best in Class	Full Buffet	Valet	Basic	No	No	Yes	
2726	Strong	NaN	Open Lot	NaN	No	Yes	Yes	
6298	Best in Class	Full Buffet	Valet	Basic	Yes	No	No	
1624	Basic	Full Buffet	Valet	NaN	Yes	Yes	Yes	

In [12]:

```
# Create a custom LinearRegression class to suppress warnings
class CustomLinearRegression(LinearRegression):
    def _validate_data(self, *args, **kwargs):
        with warnings.catch_warnings():
            warnings.filterwarnings("ignore", category=UserWarning)
            return super()._validate_data(*args, **kwargs)


# Create a dictionary to map amenity levels to their costs
cost_dict = {}
for index, row in cost.iterrows():
    amenity_name = row["Amenity"]
    if amenity_name == "pool temp":
        amenity_name = "pool_temp"
    if amenity_name not in cost_dict:
        cost_dict[amenity_name] = {}

    level = row["Level"]
    if amenity_name == "pool_temp":
        level = int(level)

    cost_dict[amenity_name][level] = row["Estimated Incremental Cost,\nPer V

# Add a 'total_cost' column to the amenities DataFrame
def calculate_total_cost(row):
    total_cost = 0
    for amenity in cost_dict:
        total_cost += cost_dict[amenity][row[amenity]]
    return total_cost

amenities["total_cost"] = amenities.apply(calculate_total_cost, axis=1)

# Merge amenities and cost DataFrames
merged_amenities = amenities.copy()

# Create dummy variables for categorical amenities
amenities_dummies = pd.get_dummies(merged_amenities, columns=['WiFi_Network'
amenities_dummies['pool_temp'] = merged_amenities['pool_temp']
```

```
# Define the independent and dependent variables
X = amenities_dummies.drop(['avg_rating', 'total_cost'], axis=1)
y = amenities_dummies['avg_rating']

# Set valid feature names for the input data (X)
X = X.set_axis([f"f{x}" for x in range(X.shape[1])], axis=1, copy=False)

# Fit the linear regression model
reg = CustomLinearRegression().fit(X, y)

best_rating = float("-inf")
best_cost = float("inf")
best_combination = None

for index, row in merged_amenities.iterrows():
    x = X.iloc[index, :]
    estimated_rating = reg.predict([x])[0]
    cost = row['total_cost']

    if estimated_rating > best_rating and cost <= 250:   # >>>>> Adjust budg
        best_rating = estimated_rating
        best_cost = cost
        best_combination = row

print("Recommended Amenity Combination:")
print("--------------------------------")
print("WiFi_Network:", best_combination["WiFi_Network"])
print("breakfast:", best_combination["breakfast"])
print("parking:", best_combination["parking"])
print("gym:", best_combination["gym"])
print("flex_check:", best_combination["flex_check"])
print("shuttle_bus:", best_combination["shuttle_bus"])
print("air_pure:", best_combination["air_pure"])
print("jacuzzi:", best_combination["jacuzzi"])
print("VIP_shop:", best_combination["VIP_shop"])
print("pool_temp:", best_combination["pool_temp"])
print("\nEstimated Average Rating:", best_rating)
print("Total Cost Per Visitor/Per Night:", best_cost)
```

```
Recommended Amenity Combination:
--------------------------------
WiFi_Network: Best in Class
breakfast: Full Buffet
parking: Open Lot
gym: Advanced
flex_check: Yes
shuttle_bus: Yes
air_pure: Yes
jacuzzi: No
VIP_shop: Yes
pool_temp: 84

Estimated Average Rating: 9.14453125
Total Cost Per Visitor/Per Night: 248.45
```

5. Recommendation

Recommended Amenity Combination

Amenities	Value $150	Standard $200	Deluxe $250	Lobster King $300
WiFi-Network	Best in Class	Best in Class	Best in Class	Best in Class
Breakfast	Full Buffet	Full Buffet	Full Buffet	Full Buffet
Parking	Open Lot	Open Lot	Open Lot	Open Lot
Gym Type	Basic	None	Advanced	Advanced
Flexible Check in/out	Yes	Yes	Yes	Yes
Shuttle Bus Service	No	Yes	Yes	Yes
Air Purifier	Yes	No	Yes	Yes
Jacuzzi	No	No	No	Yes
VIP Shopping	Yes	Yes	Yes	Yes
Outdoor Pool Temperature	84	80	84	84
Estimated Average Rating:	**8.63**	**8.82**	**9.19**	**9.38**
Total Cost Per Visitor/Per Night:	**148.45**	**190.6**	**248.45**	**288.45**

Based on the above analysis, we suggest that lobster land managers can divide the room types into four classes: Value-$150, $Standard-$200,Deluxe-$250andLobsterLing-$300, and price the rooms according to the different costs. First of all, all rooms offer the three most important amenities of the best level, best Wi-Fi, full buffet breakfast and open lot parking. Other differences are as follows, for example, in the value rooms, we choose to delete some higher cost and lower marginal effect amenities, such as jacuzzi and shuttle bus, but provide high-temperature pools, which can make the room more affordable. This approach limits the cost to 148.45, but still has an estimated rating of 8.63. The highest level lobster king comes with all the best amenities, so the cost is 288.45. Of course, if the manager chooses to firmly control the cost at $250, they can all choose the deluxe room.

AD654_Project_Forecasting_Zhen

May 7, 2023

1 Part 1: Forecast for Hilton

2 1.a. Import Dataset

[1]:
```
import pandas as pd

hlt = pd.read_csv("HLT_annual_financials.csv")
hlt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   name        55 non-null     object
 1   ttm         45 non-null     object
 2   12/31/2022  48 non-null     object
 3   12/31/2021  49 non-null     object
 4   12/31/2020  49 non-null     object
 5   12/31/2019  46 non-null     object
 6   12/31/2018  47 non-null     object
 7   12/31/2017  48 non-null     object
 8   12/31/2016  51 non-null     object
 9   12/31/2015  51 non-null     object
 10  12/31/2014  50 non-null     object
 11  12/31/2013  51 non-null     object
 12  12/31/2012  50 non-null     object
 13  12/31/2011  50 non-null     object
 14  12/31/2010  51 non-null     object
dtypes: object(15)
memory usage: 6.6+ KB
```

[2]:
```
hlt
```

[2]:
```
                                   name            ttm  \
0                          TotalRevenue  9,345,000,000
1                    \tOperatingRevenue  3,883,000,000
2                         CostOfRevenue  6,515,000,000
```

3	GrossProfit	2,830,000,000
4	OperatingExpense	607,000,000
5	\tSellingGeneralAndAdministration	382,000,000
6	\t\tGeneralAndAdministrativeExpense	382,000,000
7	\t\t\tOtherGandA	382,000,000
8	\tDepreciationAmortizationDepletionIncomeState…	155,000,000
9	\t\tDepreciationAndAmortizationInIncomeStatement	155,000,000
10	\tOtherOperatingExpenses	70,000,000
11	OperatingIncome	2,223,000,000
12	NetNonOperatingInterestIncomeExpense	-441,000,000
13	\tInterestIncomeNonOperating	NaN
14	\tInterestExpenseNonOperating	441,000,000
15	OtherIncomeExpense	-37,000,000
16	\tGainOnSaleOfSecurity	5,000,000
17	\tEarningsFromEquityInterest	NaN
18	\tSpecialIncomeCharges	0
19	\t\tRestructuringAndMergernAcquisition	0
20	\t\tImpairmentOfCapitalAssets	0
21	\t\tWriteOff	NaN
22	\t\tOtherSpecialCharges	NaN
23	\t\tGainOnSaleOfPPE	0
24	\tOtherNonOperatingIncomeExpenses	46,000,000
25	PretaxIncome	1,745,000,000
26	TaxProvision	490,000,000
27	NetIncomeCommonStockholders	1,249,000,000
28	\tNetIncome	1,249,000,000
29	\t\tNetIncomeIncludingNoncontrollingInterests	1,255,000,000
30	\t\t\tNetIncomeContinuousOperations	1,255,000,000
31	\t\t\tNetIncomeDiscontinuousOperations	NaN
32	\t\tMinorityInterests	-6,000,000
33	DilutedNIAvailtoComStockholders	1,249,000,000
34	BasicEPS	NaN
35	DilutedEPS	NaN
36	BasicAverageShares	NaN
37	DilutedAverageShares	NaN
38	TotalOperatingIncomeAsReported	2,223,000,000
39	TotalExpenses	7,122,000,000
40	NetIncomeFromContinuingAndDiscontinuedOperation	1,249,000,000
41	NormalizedIncome	1,242,527,220.63
42	InterestIncome	NaN
43	InterestExpense	441,000,000
44	NetInterestIncome	-441,000,000
45	EBIT	2,186,000,000
46	EBITDA	2,341,000,000
47	ReconciledCostOfRevenue	6,515,000,000
48	ReconciledDepreciation	155,000,000
49	NetIncomeFromContinuingOperationNetMinorityInt…	1,249,000,000

50	TotalUnusualItemsExcludingGoodwill	9,000,000
51	TotalUnusualItems	9,000,000
52	NormalizedEBITDA	2,332,000,000
53	TaxRateForCalcs	0.281
54	TaxEffectOfUnusualItems	2,527,220.63

	12/31/2022	12/31/2021	12/31/2020	12/31/2019	\
0	8,773,000,000	5,788,000,000	4,307,000,000	9,452,000,000	
1	3,634,000,000	2,365,000,000	1,527,000,000	3,665,000,000	
2	6,075,000,000	4,133,000,000	3,724,000,000	7,017,000,000	
3	2,698,000,000	1,655,000,000	583,000,000	2,435,000,000	
4	604,000,000	638,000,000	702,000,000	859,000,000	
5	382,000,000	405,000,000	311,000,000	441,000,000	
6	382,000,000	405,000,000	311,000,000	441,000,000	
7	382,000,000	405,000,000	311,000,000	441,000,000	
8	162,000,000	188,000,000	331,000,000	346,000,000	
9	162,000,000	188,000,000	331,000,000	346,000,000	
10	60,000,000	45,000,000	60,000,000	72,000,000	
11	2,094,000,000	1,017,000,000	-119,000,000	1,576,000,000	
12	-415,000,000	-397,000,000	-429,000,000	-414,000,000	
13	NaN	NaN	NaN	NaN	
14	415,000,000	397,000,000	429,000,000	414,000,000	
15	55,000,000	-60,000,000	-376,000,000	82,000,000	
16	5,000,000	-7,000,000	-27,000,000	-2,000,000	
17	NaN	NaN	NaN	NaN	
18	0	-76,000,000	-347,000,000	81,000,000	
19	0	0	41,000,000	NaN	
20	0	0	258,000,000	NaN	
21	NaN	NaN	NaN	NaN	
22	NaN	69,000,000	48,000,000	NaN	
23	0	-7,000,000	0	81,000,000	
24	50,000,000	23,000,000	-2,000,000	3,000,000	
25	1,734,000,000	560,000,000	-924,000,000	1,244,000,000	
26	477,000,000	153,000,000	-204,000,000	358,000,000	
27	1,255,000,000	410,000,000	-715,000,000	881,000,000	
28	1,255,000,000	410,000,000	-715,000,000	881,000,000	
29	1,257,000,000	407,000,000	-720,000,000	886,000,000	
30	1,257,000,000	407,000,000	-720,000,000	886,000,000	
31	NaN	NaN	NaN	NaN	
32	-2,000,000	3,000,000	5,000,000	-5,000,000	
33	1,255,000,000	410,000,000	-715,000,000	881,000,000	
34	4.56	1.47	-2.58	3.07	
35	4.53	1.46	-2.58	3.04	
36	275,000,000	279,000,000	277,000,000	287,000,000	
37	277,000,000	281,000,000	279,000,000	290,000,000	
38	2,094,000,000	1,010,000,000	-418,000,000	1,657,000,000	
39	6,679,000,000	4,771,000,000	4,426,000,000	7,876,000,000	

40	1,255,000,000	410,000,000	-715,000,000	881,000,000
41	1,251,375,432.526	470,323,214.286	-423,571,428.571	824,734,726.688
42	NaN	NaN	NaN	NaN
43	415,000,000	397,000,000	429,000,000	414,000,000
44	-415,000,000	-397,000,000	-429,000,000	-414,000,000
45	2,149,000,000	957,000,000	-495,000,000	1,658,000,000
46	NaN	NaN	NaN	NaN
47	6,075,000,000	4,133,000,000	3,724,000,000	7,017,000,000
48	162,000,000	188,000,000	331,000,000	346,000,000
49	1,255,000,000	410,000,000	-715,000,000	881,000,000
50	5,000,000	-83,000,000	-374,000,000	79,000,000
51	5,000,000	-83,000,000	-374,000,000	79,000,000
52	2,306,000,000	1,228,000,000	210,000,000	1,925,000,000
53	0.275	0.273	0.221	0.288
54	1,375,432.526	-22,676,785.714	-82,571,428.571	22,734,726.688

	12/31/2018	12/31/2017	12/31/2016	12/31/2015 \
0	8,906,000,000	9,140,000,000	11,663,000,000	11,272,000,000
1	3,570,000,000	3,390,000,000	7,217,000,000	7,142,000,000
2	6,655,000,000	6,931,000,000	8,494,000,000	8,195,000,000
3	2,251,000,000	2,209,000,000	3,169,000,000	3,077,000,000
4	819,000,000	837,000,000	1,302,000,000	1,303,000,000
5	443,000,000	434,000,000	616,000,000	611,000,000
6	443,000,000	434,000,000	616,000,000	611,000,000
7	443,000,000	434,000,000	616,000,000	611,000,000
8	325,000,000	347,000,000	686,000,000	692,000,000
9	325,000,000	347,000,000	686,000,000	692,000,000
10	51,000,000	56,000,000	4,446,000,000	4,130,000,000
11	1,432,000,000	1,372,000,000	1,867,000,000	1,774,000,000
12	-371,000,000	-408,000,000	-575,000,000	-556,000,000
13	NaN	NaN	12,000,000	19,000,000
14	371,000,000	408,000,000	587,000,000	575,000,000
15	17,000,000	-34,000,000	-37,000,000	278,000,000
16	-11,000,000	3,000,000	-13,000,000	-41,000,000
17	NaN	NaN	8,000,000	23,000,000
18	0	-60,000,000	-6,000,000	297,000,000
19	NaN	NaN	NaN	NaN
20	NaN	NaN	15,000,000	9,000,000
21	NaN	NaN	15,000,000	9,000,000
22	NaN	60,000,000	NaN	NaN
23	0	0	9,000,000	306,000,000
24	28,000,000	23,000,000	-26,000,000	-1,000,000
25	1,078,000,000	930,000,000	1,255,000,000	1,496,000,000
26	309,000,000	-334,000,000	891,000,000	80,000,000
27	764,000,000	1,259,000,000	348,000,000	1,404,000,000
28	764,000,000	1,259,000,000	348,000,000	1,404,000,000
29	769,000,000	1,264,000,000	364,000,000	1,416,000,000

30	769,000,000	1,264,000,000	364,000,000	1,416,000,000
31	0	0	NaN	NaN
32	-5,000,000	-5,000,000	-16,000,000	-12,000,000
33	764,000,000	1,259,000,000	348,000,000	1,404,000,000
34	2.53	3.34	1.06	4.26
35	2.5	3.32	1.05	4.26
36	302,000,000	324,000,000	329,032,574	328,666,338
37	305,000,000	327,000,000	330,032,673	329,666,337
38	1,432,000,000	1,372,000,000	1,861,000,000	2,071,000,000
39	7,474,000,000	7,768,000,000	9,796,000,000	9,498,000,000
40	764,000,000	1,259,000,000	348,000,000	1,404,000,000
41	771,846,938.776	1,293,200,000	359,400,000	1,161,689,839.572
42	NaN	NaN	12,000,000	19,000,000
43	371,000,000	408,000,000	587,000,000	575,000,000
44	-371,000,000	-408,000,000	-575,000,000	-556,000,000
45	1,449,000,000	1,338,000,000	1,842,000,000	2,071,000,000
46	NaN	NaN	NaN	NaN
47	6,655,000,000	6,931,000,000	8,494,000,000	8,195,000,000
48	325,000,000	347,000,000	686,000,000	692,000,000
49	764,000,000	1,259,000,000	348,000,000	1,404,000,000
50	-11,000,000	-57,000,000	-19,000,000	256,000,000
51	-11,000,000	-57,000,000	-19,000,000	256,000,000
52	1,785,000,000	1,742,000,000	2,547,000,000	2,507,000,000
53	0.287	0.4	0.4	0.053
54	-3,153,061.224	-22,800,000	-7,600,000	13,689,839.572

	12/31/2014	12/31/2013	12/31/2012	12/31/2011 \
0	10,502,000,000	9,735,000,000	9,276,000,000	8,783,000,000
1	6,811,000,000	6,330,000,000	6,152,000,000	5,856,000,000
2	7,710,000,000	7,282,000,000	7,112,000,000	6,808,000,000
3	2,792,000,000	2,453,000,000	2,164,000,000	1,975,000,000
4	1,119,000,000	1,351,000,000	1,010,000,000	980,000,000
5	491,000,000	748,000,000	460,000,000	416,000,000
6	491,000,000	748,000,000	460,000,000	416,000,000
7	491,000,000	748,000,000	460,000,000	416,000,000
8	628,000,000	603,000,000	550,000,000	564,000,000
9	628,000,000	603,000,000	550,000,000	564,000,000
10	3,691,000,000	3,405,000,000	3,124,000,000	2,927,000,000
11	1,673,000,000	1,102,000,000	1,154,000,000	995,000,000
12	-608,000,000	-611,000,000	-554,000,000	-632,000,000
13	10,000,000	9,000,000	15,000,000	11,000,000
14	618,000,000	620,000,000	569,000,000	643,000,000
15	82,000,000	207,000,000	-27,000,000	-167,000,000
16	26,000,000	-45,000,000	23,000,000	-21,000,000
17	19,000,000	16,000,000	-11,000,000	-145,000,000
18	0	229,000,000	-54,000,000	-20,000,000
19	NaN	NaN	NaN	NaN

20	0	0	54,000,000	20,000,000
21	0	0	54,000,000	20,000,000
22	NaN	-229,000,000	NaN	NaN
23	NaN	NaN	NaN	NaN
24	37,000,000	7,000,000	15,000,000	19,000,000
25	1,147,000,000	698,000,000	573,000,000	196,000,000
26	465,000,000	238,000,000	214,000,000	-59,000,000
27	673,000,000	415,000,000	352,000,000	253,000,000
28	673,000,000	415,000,000	352,000,000	253,000,000
29	682,000,000	460,000,000	359,000,000	255,000,000
30	682,000,000	460,000,000	359,000,000	255,000,000
31	NaN	NaN	NaN	NaN
32	-9,000,000	-45,000,000	-7,000,000	-2,000,000
33	673,000,000	415,000,000	352,000,000	253,000,000
34	2.04	1.35	1.14	0.81
35	2.04	1.35	1.14	0.81
36	328,333,005	307,407,100	306,999,693	306,999,693
37	328,666,338	307,407,100	306,999,693	306,999,693
38	1,673,000,000	1,102,000,000	1,100,000,000	975,000,000
39	8,829,000,000	8,633,000,000	8,122,000,000	7,788,000,000
40	673,000,000	415,000,000	352,000,000	253,000,000
41	656,100,000	293,739,255.014	371,422,338.569	279,650,000
42	10,000,000	9,000,000	15,000,000	11,000,000
43	618,000,000	620,000,000	569,000,000	643,000,000
44	-608,000,000	-611,000,000	-554,000,000	-632,000,000
45	1,765,000,000	1,318,000,000	1,142,000,000	839,000,000
46	NaN	NaN	NaN	NaN
47	7,710,000,000	7,282,000,000	7,112,000,000	6,808,000,000
48	628,000,000	603,000,000	550,000,000	564,000,000
49	673,000,000	415,000,000	352,000,000	253,000,000
50	26,000,000	184,000,000	-31,000,000	-41,000,000
51	26,000,000	184,000,000	-31,000,000	-41,000,000
52	2,367,000,000	1,737,000,000	1,723,000,000	1,444,000,000
53	0.35	0.341	0.373	0.35
54	9,100,000	62,739,255.014	-11,577,661.431	-14,350,000

	12/31/2010
0	8,068,000,000
1	5,431,000,000
2	6,280,000,000
3	1,788,000,000
4	1,211,000,000
5	637,000,000
6	637,000,000
7	637,000,000
8	574,000,000
9	574,000,000

10	2,637,000,000
11	577,000,000
12	-937,000,000
13	9,000,000
14	946,000,000
15	779,000,000
16	18,000,000
17	-12,000,000
18	765,000,000
19	NaN
20	24,000,000
21	24,000,000
22	-789,000,000
23	NaN
24	8,000,000
25	419,000,000
26	308,000,000
27	128,000,000
28	128,000,000
29	111,000,000
30	111,000,000
31	NaN
32	17,000,000
33	128,000,000
34	0.39
35	0.39
36	328,204,793
37	328,204,793
38	553,000,000
39	7,491,000,000
40	128,000,000
41	-380,950,000
42	9,000,000
43	946,000,000
44	-937,000,000
45	1,365,000,000
46	NaN
47	6,280,000,000
48	574,000,000
49	128,000,000
50	783,000,000
51	783,000,000
52	1,156,000,000
53	0.35
54	274,050,000

[3]:
```python
# Data pre-processing
# Extract net income row
net_income_hlt = hlt[hlt['name'] == '\tNetIncome']

# Drop name and ttm columns
net_income_hlt = net_income_hlt.drop(columns=['name', 'ttm'])

# Transpose the DataFrame and convert index to datetime
net_income_hlt = net_income_hlt.transpose()
net_income_hlt.index = pd.to_datetime(net_income_hlt.index)

# Convert values to numeric, replacing any non-numeric characters
net_income_hlt = net_income_hlt.replace('[\$,]', '', regex=True).astype(float)

# Rename the column to 'Net Income'
net_income_hlt.columns = ['Net Income']

net_income_hlt
```

[3]:
```
              Net Income
2022-12-31  1.255000e+09
2021-12-31  4.100000e+08
2020-12-31 -7.150000e+08
2019-12-31  8.810000e+08
2018-12-31  7.640000e+08
2017-12-31  1.259000e+09
2016-12-31  3.480000e+08
2015-12-31  1.404000e+09
2014-12-31  6.730000e+08
2013-12-31  4.150000e+08
2012-12-31  3.520000e+08
2011-12-31  2.530000e+08
2010-12-31  1.280000e+08
```

[4]:
```python
# Convert the net income values to millions and round to 2 decimal places
net_income_hlt['Net Income'] = net_income_hlt['Net Income'].apply(lambda x:␣
↪round(x / 1e6, 2))

# Rename the column to 'Net Income (Millions)'
net_income_hlt.columns = ['Net Income (Millions)']

net_income_hlt
```

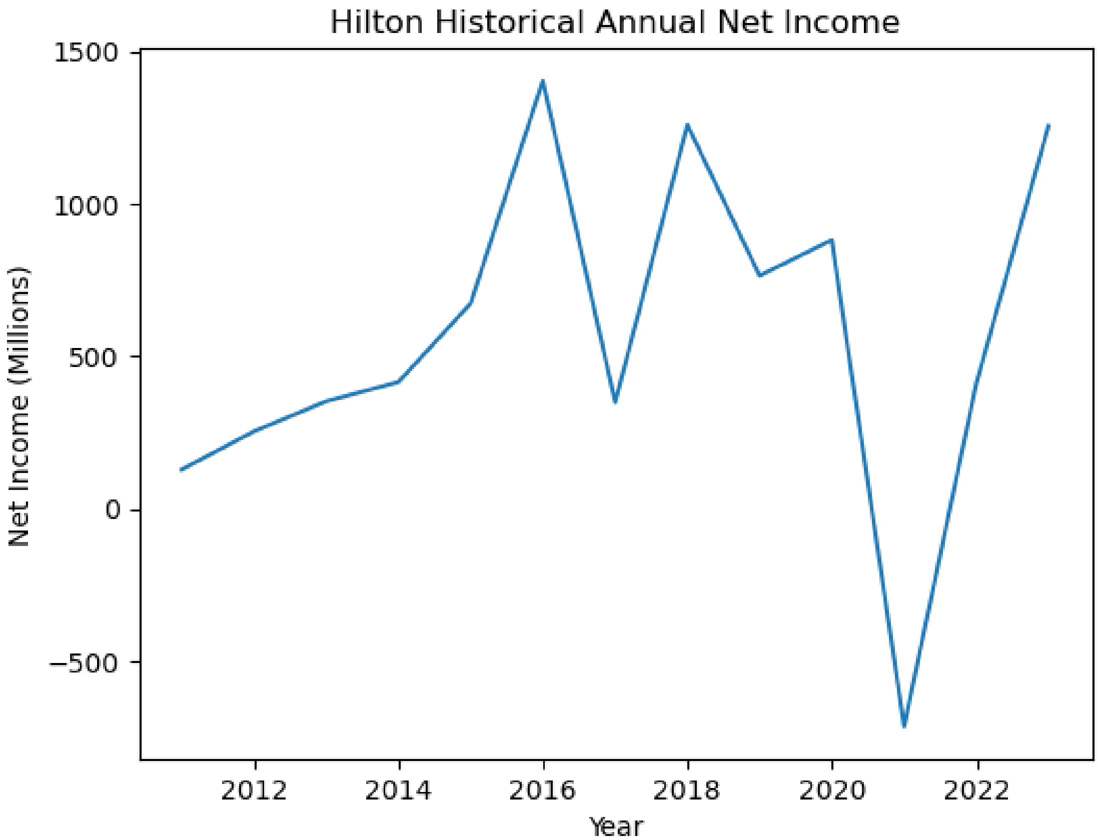
[4]:
```
            Net Income (Millions)
2022-12-31                 1255.0
2021-12-31                  410.0
2020-12-31                 -715.0
```

```
2019-12-31                    881.0
2018-12-31                    764.0
2017-12-31                   1259.0
2016-12-31                    348.0
2015-12-31                   1404.0
2014-12-31                    673.0
2013-12-31                    415.0
2012-12-31                    352.0
2011-12-31                    253.0
2010-12-31                    128.0
```

[5]:

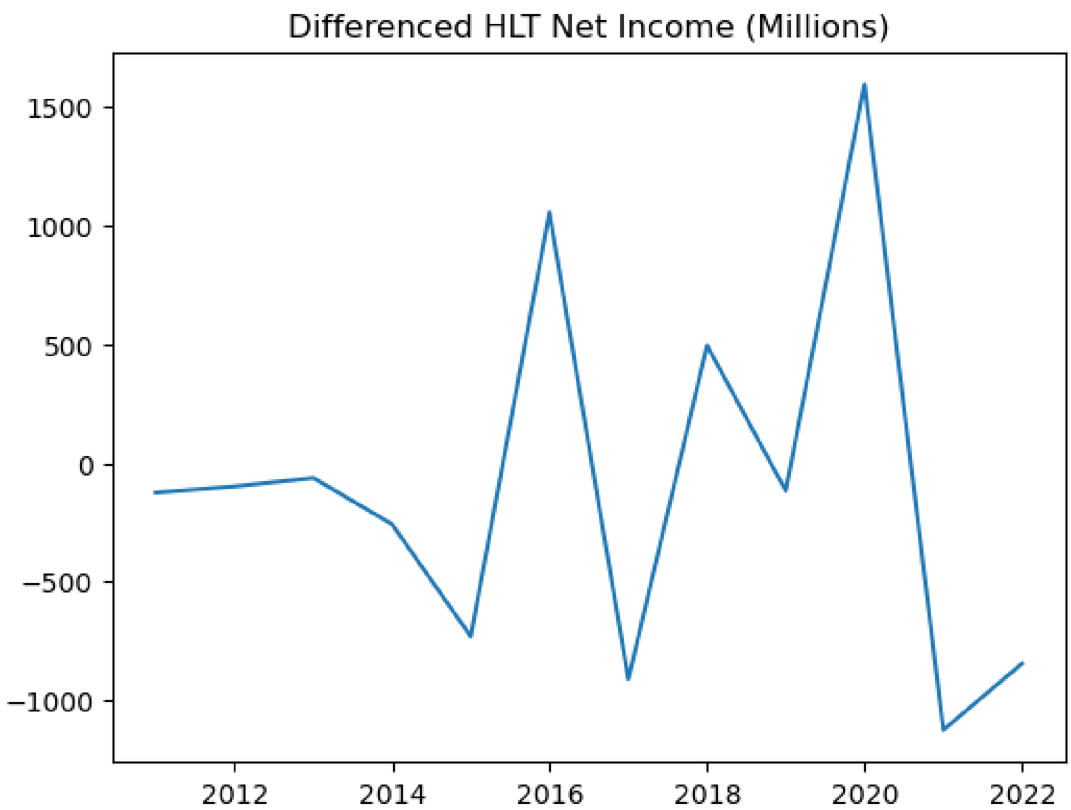
```python
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

# Perform Augmented Dickey-Fuller test to check stationarity
result = adfuller(net_income_hlt['Net Income (Millions)'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])

# Plot the time series
plt.plot(net_income_hlt['Net Income (Millions)'])
plt.title("Hilton Historical Annual Net Income")
plt.xlabel("Year")
plt.ylabel("Net Income (Millions)")
plt.show()
```

```
ADF Statistic: -2.215128
p-value: 0.200801
```



[6]:
```
# Differencing to make the series stationary
diff_net_income_hlt = net_income_hlt['Net Income (Millions)'].diff().dropna()

# Check stationarity again
result = adfuller(diff_net_income_hlt)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])

# Plot the differenced time series
plt.plot(diff_net_income_hlt)
plt.title("Differenced HLT Net Income (Millions)")
plt.show()
```

```
ADF Statistic: -1.091967
p-value: 0.718265
```



The ADF Statistic is -2.215128. It is compared to the critical values (e.g., 1%, 5%, and 10% levels) to determine if the null hypothesis can be rejected. In general, if the ADF Statistic is smaller (i.e., more negative) than the critical values, we reject the null hypothesis, meaning that the time series is stationary.

The p-value is 0.200801. A p-value less than or equal to a significance level (commonly 0.05) indicates that we reject the null hypothesis, and the time series is stationary. In this case, the p-value is greater than 0.05, which means that the null hypothesis cannot be rejected, and the time series is not stationary.

The plot of the time series also shows that the net income fluctuates over time, and there does not seem to be a constant mean or variance, which further supports the non-stationary nature of the time series.

3 1.b. ARIMA Forecast for HLT

[21]:
```
import pmdarima as pm

# Find optimal ARIMA parameters
model = pm.auto_arima(net_income_hlt['Net Income (Millions)'], seasonal=False,
    suppress_warnings=True)
```

```python
# Print optimal order
print(model.order)
```

```
(0, 0, 0)
```

ARIMA model with order (0, 0, 0) is equivalent to a simple mean model, which is not the best approach for forecasting net income.

[22]:
```python
from statsmodels.tsa.arima.model import ARIMA

# Fit the ARIMA model
arima_model = ARIMA(net_income_hlt['Net Income (Millions)'], order=model.order)
arima_model_fit = arima_model.fit()

# Print model summary
print(arima_model_fit.summary())
```

```
                               SARIMAX Results
================================================================================
=
Dep. Variable:     Net Income (Millions)   No. Observations:
13
Model:                             ARIMA   Log Likelihood
-100.384
Date:                   Sun, 07 May 2023   AIC
204.769
Time:                           20:52:50   BIC
205.899
Sample:                                0   HQIC
204.536
                                    - 13
Covariance Type:                     opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        571.3078    160.131      3.568      0.000     257.457     885.158
sigma2      2.985e+05   1.17e+05      2.542      0.011    6.83e+04    5.29e+05
================================================================================
===
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
0.53
Prob(Q):                              0.89   Prob(JB):
0.77
Heteroskedasticity (H):               0.17   Skew:
-0.48
Prob(H) (two-sided):                  0.11   Kurtosis:
3.22
================================================================================
===
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

```
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency -1A-DEC will be used.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: A date index has
been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency -1A-DEC will be used.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: A date index has
been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency -1A-DEC will be used.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: A date index has
been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
```

[23]:

```python
# Forecast the 2023 net income
forecast_2023 = arima_model_fit.forecast(steps=1)

# Print the 2023 net income forecast
print("2023 Net Income (Millions) forecast: ", forecast_2023.iloc[-1])
```

```
2023 Net Income (Millions) forecast:  571.3077597861073
```

```
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:834: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
```

ARIMA model is not suitable for this forecast, disregard this result.

4 1.c. Linear Regression Forecast for HLT (Chosen)

[10]:
```
from sklearn.linear_model import LinearRegression

# Create a new DataFrame with a column for years
net_income_hlt_with_years = net_income_hlt.reset_index()
net_income_hlt_with_years['Year'] = net_income_hlt_with_years['index'].dt.year

# Drop the index column
net_income_hlt_with_years = net_income_hlt_with_years.drop(columns=['index'])

# Fit a linear regression model
X = net_income_hlt_with_years[['Year']]
y = net_income_hlt_with_years['Net Income (Millions)']
reg = LinearRegression().fit(X, y)

# Predict the 2023 net income
forecast_2023 = reg.predict(np.array([[2023]]))

# Print the 2023 net income forecast
print("2023 Net Income (Millions) forecast for HLT: ", forecast_2023[0])
```

```
2023 Net Income (Millions) forecast for HLT:  752.6153846153902
```

```
C:\Users\watso\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

Let's use the linear regression forecast result here.

5 Part 2: Forecast for Hyatt

6 2.a. Import Dataset for H

[11]:
```
h = pd.read_csv("H_annual_financials.csv")
```

[12]:
```
# Data pre-processing
# Extract net income row
net_income_h = h[h['name'] == '\tNetIncome']

# Drop name and ttm columns
net_income_h = net_income_h.drop(columns=['name', 'ttm'])

# Transpose the DataFrame and convert index to datetime
net_income_h = net_income_h.transpose()
net_income_h.index = pd.to_datetime(net_income_h.index)
```

```
# Convert values to numeric, replacing any non-numeric characters
net_income_h = net_income_h.replace('[\$,]', '', regex=True).astype(float)

# Rename the column to 'Net Income'
net_income_h.columns = ['Net Income']

# Convert the net income values to millions and round to 2 decimal places
net_income_h['Net Income'] = net_income_h['Net Income'].apply(lambda x: round(x␣
↪/ 1e6, 2))

# Rename the column to 'Net Income (Millions)'
net_income_h.columns = ['Net Income (Millions)']

net_income_h
```

[12]:

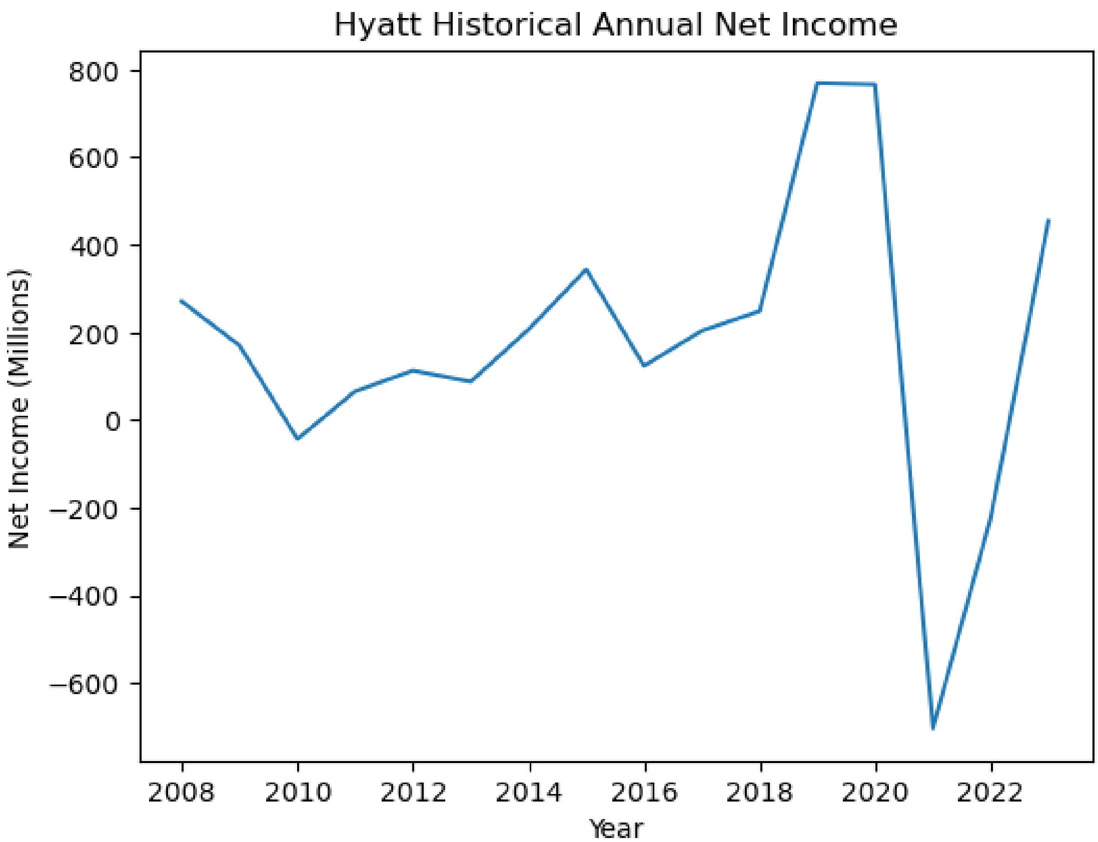
```
            Net Income (Millions)
2022-12-31                  455.0
2021-12-31                 -222.0
2020-12-31                 -703.0
2019-12-31                  766.0
2018-12-31                  769.0
2017-12-31                  249.0
2016-12-31                  204.0
2015-12-31                  124.0
2014-12-31                  344.0
2013-12-31                  207.0
2012-12-31                   88.0
2011-12-31                  113.0
2010-12-31                   66.0
2009-12-31                  -43.0
2008-12-31                  170.0
2007-12-31                  271.0
```

[13]:

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

# Perform Augmented Dickey-Fuller test to check stationarity
result = adfuller(net_income_h['Net Income (Millions)'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])

# Plot the time series
plt.plot(net_income_h['Net Income (Millions)'])
plt.title("Hyatt Historical Annual Net Income")
plt.xlabel("Year")
plt.ylabel("Net Income (Millions)")
```

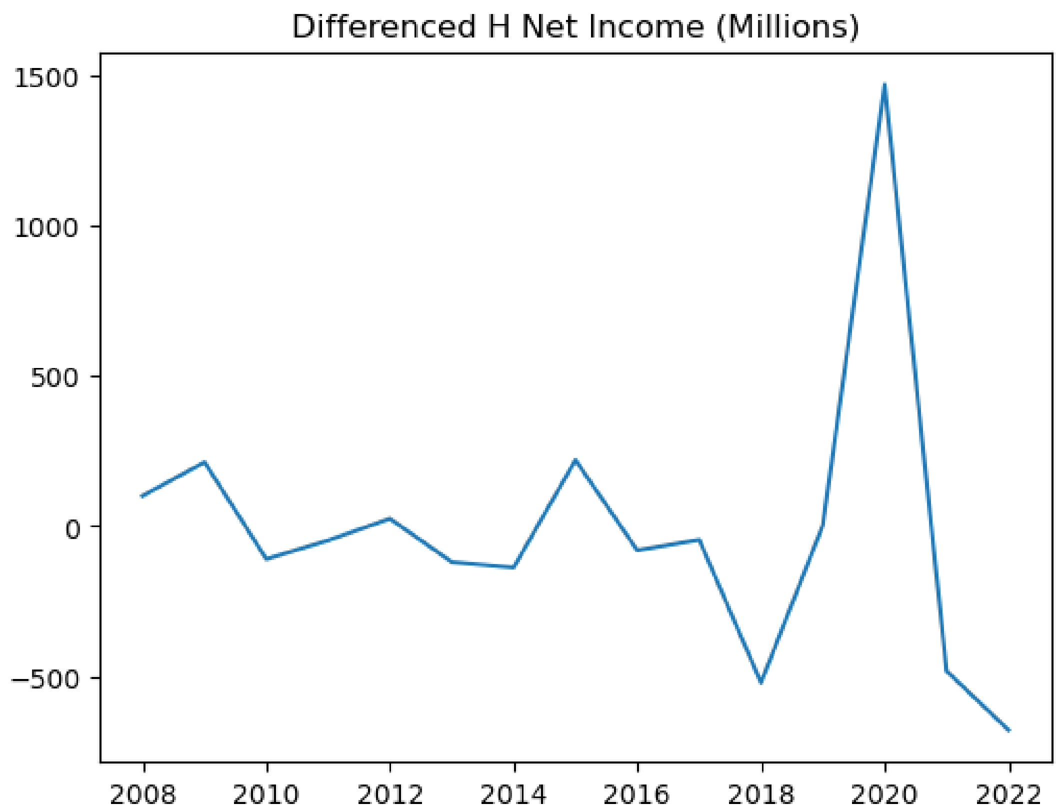
```
plt.show()
```

```
ADF Statistic: -3.398954
p-value: 0.010995
```



[14]:
```
# Differencing to make the series stationary
diff_net_income_h = net_income_h['Net Income (Millions)'].diff().dropna()

# Check stationarity again
result = adfuller(diff_net_income_h)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])

# Plot the differenced time series
plt.plot(diff_net_income_h)
plt.title("Differenced H Net Income (Millions)")
plt.show()
```

```
ADF Statistic: -4.197156
p-value: 0.000666
```



7 2.b. ARIMA Forecast for H

[15]:
```
import pmdarima as pm

# Find optimal ARIMA parameters
model = pm.auto_arima(net_income_h['Net Income (Millions)'], seasonal=False,␣
 ↪suppress_warnings=True)

# Print optimal order
print(model.order)
```

```
(0, 0, 0)
```

Given the optimal ARIMA parameters for the H Net Income (Millions) time series are (0, 0, 0), the ARIMA model essentially reduces to a simple mean model, which means the model will predict the mean value of the H Net Income (Millions) time series for all future periods. This may not provide the most accurate forecast, as the model does not capture any trends or patterns in the data.

[16]:
```
# Fit the ARIMA model
arima_model = ARIMA(net_income_h['Net Income (Millions)'], order=model.order)
arima_model_fit = arima_model.fit()
```

```python
# Print model summary
print(arima_model_fit.summary())
```

```
                               SARIMAX Results
================================================================================
=
Dep. Variable:     Net Income (Millions)   No. Observations:
16
Model:                             ARIMA   Log Likelihood
-115.869
Date:                   Sun, 07 May 2023   AIC
235.739
Time:                           20:18:54   BIC
237.284
Sample:                                0   HQIC
235.818
                                    - 16
Covariance Type:                     opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        178.6250     88.225      2.025      0.043       5.708     351.542
sigma2      1.139e+05   3.32e+04      3.434      0.001    4.89e+04    1.79e+05
================================================================================
===
Ljung-Box (L1) (Q):                   0.12   Jarque-Bera (JB):
1.74
Prob(Q):                              0.73   Prob(JB):
0.42
Heteroskedasticity (H):               0.04   Skew:
-0.53
Prob(H) (two-sided):                  0.00   Kurtosis:
4.22
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

```
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency -1A-DEC will be used.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: A date index has
been provided, but it is not monotonic and so will be ignored when e.g.
```

```
forecasting.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency -1A-DEC will be used.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: A date index has
been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency -1A-DEC will be used.
  self._init_dates(dates, freq)
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: A date index has
been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
```

[17]:
```python
# Forecast the 2023 net income
forecast_2023 = arima_model_fit.forecast(steps=1)

# Print the 2023 net income forecast
print("2023 Net Income (Millions) forecast for H: ", forecast_2023.iloc[-1])
```

```
2023 Net Income (Millions) forecast for H:  178.6249825429822
```

```
C:\Users\watso\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:834: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
```

8 2.c. Linear Regression Forecast for H (Chosen)

[18]:
```python
from sklearn.linear_model import LinearRegression

# Create a new DataFrame with a column for years
net_income_h_with_years = net_income_h.reset_index()
net_income_h_with_years['Year'] = net_income_h_with_years['index'].dt.year

# Drop the index column
net_income_h_with_years = net_income_h_with_years.drop(columns=['index'])

# Fit a linear regression model
X = net_income_h_with_years[['Year']]
```

```python
y = net_income_h_with_years['Net Income (Millions)']
reg = LinearRegression().fit(X, y)

# Predict the 2023 net income
forecast_2023 = reg.predict(np.array([[2023]]))

# Print the 2023 net income forecast
print("2023 Net Income (Millions) forecast for H: ", forecast_2023[0])
```

```
2023 Net Income (Millions) forecast for H:  202.02499999999964
```

```
C:\Users\watso\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

9 2.d. Simple Moving Average Forecast for H

[24]:
```python
# Choose a window size (e.g., 3)
window_size = 3

# Calculate the moving average
sma = net_income_h['Net Income (Millions)'].rolling(window=window_size).mean()

# Forecast the 2023 net income by taking the last moving average value
forecast_2023 = sma.iloc[-1]

# Print the 2023 net income forecast
print("2023 Net Income (Millions) forecast using SMA: ", forecast_2023)
```

```
2023 Net Income (Millions) forecast using SMA:  132.66666666666666
```

10 2.e. Exponential Moving Average for H

[25]:
```python
# Choose a smoothing factor alpha (e.g., 0.5)
alpha = 0.5

# Calculate the exponential moving average
ema = net_income_h['Net Income (Millions)'].ewm(alpha=alpha).mean()

# Forecast the 2023 net income by taking the last moving average value
forecast_2023 = ema.iloc[-1]

# Print the 2023 net income forecast
print("2023 Net Income (Millions) forecast using EMA: ", forecast_2023)
```

```
2023 Net Income (Millions) forecast using EMA:  185.42151522087434
```

11 Summary

Since we are using the annual income statements instead of quarterly income statements, which have more data points and clearer seasonalities, we decided to use the linear regression forecasting method to forecast based on historical annual net income for both companies for simplicity and as a baseline expectation. First, we imported the annual financial data from Yahoo Finance for both companies and extracted the net income row for each. After cleaning and transforming the data, we performed linear regression forecasting to forecast the 2023 net income for both companies, which resulted in $752.62 million for Hilton and $202.02 million for Hyatt Hotels.

As seen from plots of the historical net income for both companies, both are steadily recovering from the 2020-2021 COVID downturn. We are optimistic that both companies' net income in 2023 will exceed our forecast. Forecasting is not always accurate, and the actual results may vary. When making predictions, it is essential to consider other factors such as macroeconomic conditions, industry trends, hospitality seasonality, currency exchange rates, and COVID recovery status.

[ ]:

```

```

AD654_Project_Classification_Zhen

May 7, 2023

1 1. Import and inspect the dataset

[51]:
```
# Import the dataset
import pandas as pd
satisfaction = pd.read_csv('hotel_satisfaction.csv')
satisfaction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 17 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   id                                  103904 non-null  int64
 1   Gender                              103904 non-null  object
 2   Age                                 103904 non-null  int64
 3   purpose_of_travel                   103904 non-null  object
 4   Type of Travel                      103904 non-null  object
 5   Type Of Booking                     103904 non-null  object
 6   Hotel wifi service                  103904 non-null  int64
 7   Departure/Arrival  convenience      103904 non-null  int64
 8   Ease of Online booking              103904 non-null  int64
 9   Hotel location                      103904 non-null  int64
 10  Food and drink                      103904 non-null  int64
 11  Stay comfort                        103904 non-null  int64
 12  Common Room entertainment           103904 non-null  int64
 13  Checkin/Checkout service            103904 non-null  int64
 14  Other service                       103904 non-null  int64
 15  Cleanliness                         103904 non-null  int64
 16  satisfaction                        103904 non-null  object
dtypes: int64(12), object(5)
memory usage: 13.5+ MB
```

[52]:
```
satisfaction.head()
```

[52]:
```
       id  Gender  Age purpose_of_travel   Type of Travel Type Of Booking  \
0   70172    Male   13          aviation  Personal Travel     Not defined
1    5047    Male   25           tourism     Group Travel  Group bookings
2  110028  Female   26           tourism     Group Travel  Group bookings
```

```
3   24026  Female   25           tourism     Group Travel  Group bookings
4  119299    Male   61          aviation     Group Travel  Group bookings

   Hotel wifi service  Departure/Arrival  convenience  Ease of Online booking  \
0                   3                                4                       3
1                   3                                2                       3
2                   2                                2                       2
3                   2                                5                       5
4                   3                                3                       3

   Hotel location  Food and drink  Stay comfort  Common Room entertainment  \
0               1               5             5                          5
1               3               1             1                          1
2               2               5             5                          5
3               5               2             2                          2
4               3               4             5                          3

   Checkin/Checkout service  Other service  Cleanliness  \
0                         4              5            5
1                         1              4            1
2                         4              4            5
3                         1              4            2
4                         3              3            3

              satisfaction
0  neutral or dissatisfied
1  neutral or dissatisfied
2                satisfied
3  neutral or dissatisfied
4                satisfied
```

[53]:
```python
# Check for missing values
print(satisfaction.isnull().sum())
```

```
id                                  0
Gender                              0
Age                                 0
purpose_of_travel                   0
Type of Travel                      0
Type Of Booking                     0
Hotel wifi service                  0
Departure/Arrival  convenience      0
Ease of Online booking              0
Hotel location                      0
Food and drink                      0
Stay comfort                        0
Common Room entertainment           0
Checkin/Checkout service            0
```

```
Other service                     0
Cleanliness                       0
satisfaction                      0
dtype: int64
```

No missing values.

[54]:
```python
# List the rating columns to check for 0's
rating_columns = [
    "Hotel wifi service",
    "Departure/Arrival  convenience",
    "Ease of Online booking",
    "Hotel location",
    "Food and drink",
    "Stay comfort",
    "Common Room entertainment",
    "Checkin/Checkout service",
    "Other service",
    "Cleanliness",
]

# Count the number of 0's for each rating column
zero_counts = {column: (satisfaction[column] == 0).sum() for column in␣
 ↪rating_columns}

for column, count in zero_counts.items():
    print(f"Number of 0's in {column}: {count}")
```

```
Number of 0's in Hotel wifi service: 3103
Number of 0's in Departure/Arrival  convenience: 5300
Number of 0's in Ease of Online booking: 4487
Number of 0's in Hotel location: 1
Number of 0's in Food and drink: 107
Number of 0's in Stay comfort: 1
Number of 0's in Common Room entertainment: 14
Number of 0's in Checkin/Checkout service: 1
Number of 0's in Other service: 3
Number of 0's in Cleanliness: 12
```

Looks like 0 is a valid rating, rating scale from 0 to 5.

2 2. Random Forest Classification

[55]:
```python
# Convert categorical variables to numerical using one-hot encoding
satisfaction = pd.get_dummies(satisfaction, columns=['Gender',␣
 ↪'purpose_of_travel', 'Type of Travel', 'Type Of Booking'])

# Convert the satisfaction column to binary format
```

```
satisfaction['satisfaction'] = satisfaction['satisfaction'].map({'satisfied':␣
↪1, 'neutral or dissatisfied': 0})

# Drop the id column as it doesn't contribute to the model
satisfaction = satisfaction.drop(columns=['id'])

# Split the dataset into features (X) and target (y) variables
X = satisfaction.drop(columns=['satisfaction'])
y = satisfaction['satisfaction']
```

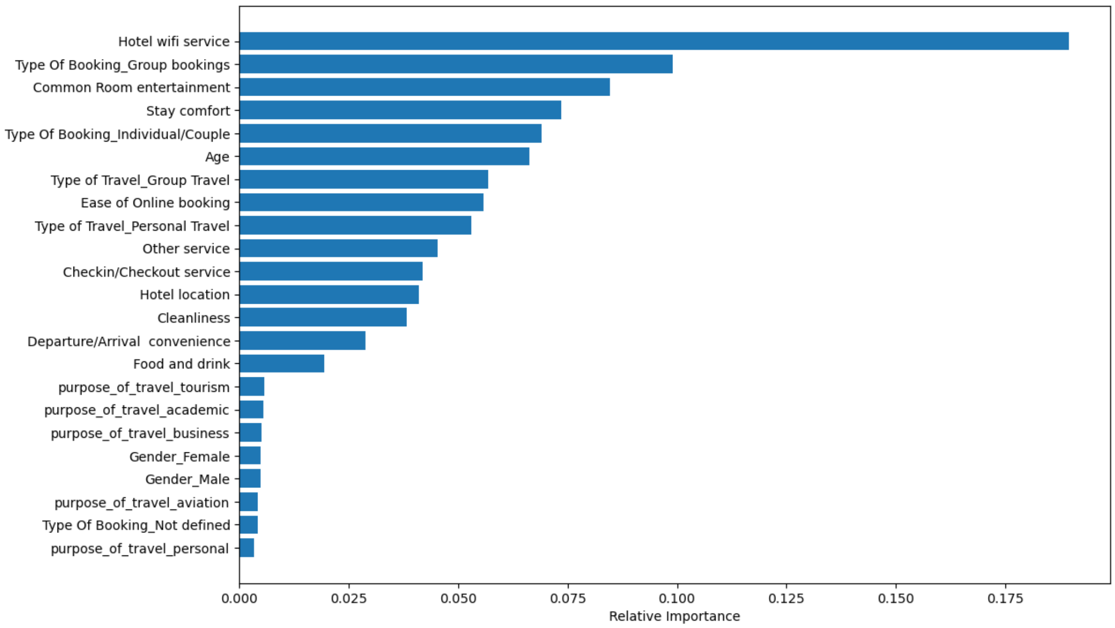
[56]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Split the dataset into a training (70%) and testing (30%) set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=240)

# Initialize and fit the RandomForestClassifier
clf = RandomForestClassifier(random_state=240)
clf.fit(X_train, y_train)

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

# Calculate the accuracy and print the classification report
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
print(classification_report(y_test, y_pred))
```

```
Accuracy:  0.9453676376235083
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     17655
           1       0.95      0.93      0.94     13517

    accuracy                           0.95     31172
   macro avg       0.95      0.94      0.94     31172
weighted avg       0.95      0.95      0.95     31172
```

3 2.a. Feature Importance Plot

[57]:
```
import numpy as np
import matplotlib.pyplot as plt
importances = clf.feature_importances_
features = X.columns
```

```
indices = np.argsort(importances)

plt.figure(figsize=(12, 8))
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



4 2.b. Feature Importance Table

[58]:
```
importances = clf.feature_importances_
features = X.columns

# Create a DataFrame with feature importances and column names
importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})

# Sort the DataFrame by descending importance
importance_df = importance_df.sort_values('Importance', ascending=False)

# Display the sorted DataFrame
print(importance_df)
```

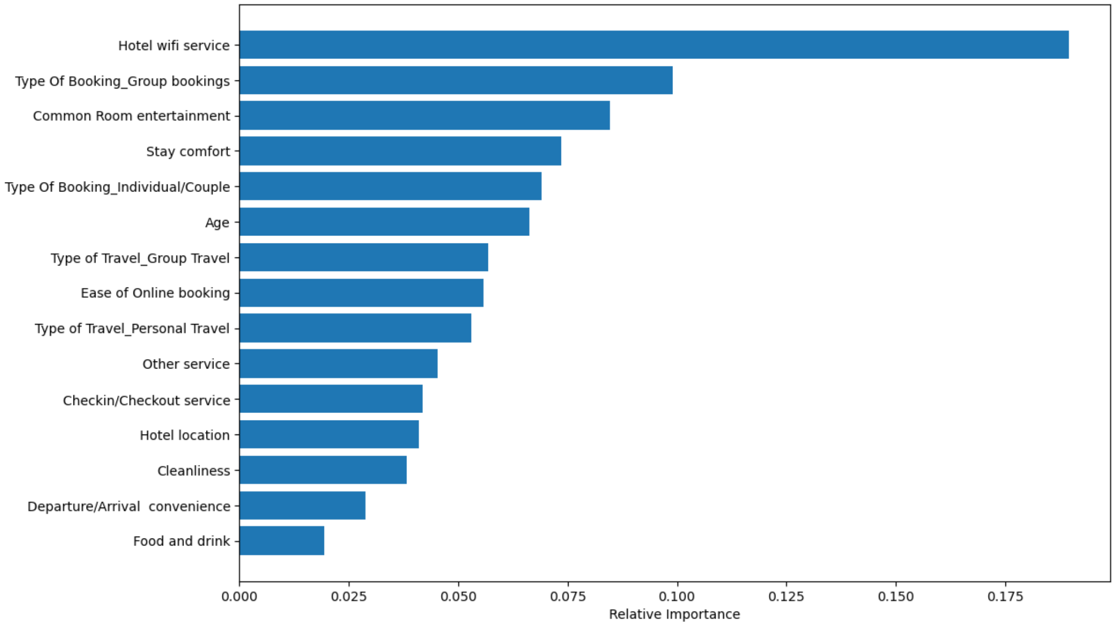
```
                            Feature  Importance
1                 Hotel wifi service    0.189585
20      Type Of Booking_Group bookings    0.099038
7           Common Room entertainment    0.084648
```

```
6                          Stay comfort    0.073454
21  Type Of Booking_Individual/Couple    0.069014
0                                   Age    0.066269
18          Type of Travel_Group Travel    0.056860
3                 Ease of Online booking    0.055703
19       Type of Travel_Personal Travel    0.053027
9                         Other service    0.045389
8              Checkin/Checkout service    0.041816
4                        Hotel location    0.041079
10                          Cleanliness    0.038289
2       Departure/Arrival  convenience    0.028847
5                        Food and drink    0.019358
17           purpose_of_travel_tourism    0.005721
13          purpose_of_travel_academic    0.005431
15          purpose_of_travel_business    0.005134
11                        Gender_Female    0.004849
12                          Gender_Male    0.004750
14          purpose_of_travel_aviation    0.004230
22         Type Of Booking_Not defined    0.004121
16          purpose_of_travel_personal    0.003388
```

5 2.c. Alternative Feature Importance Plot

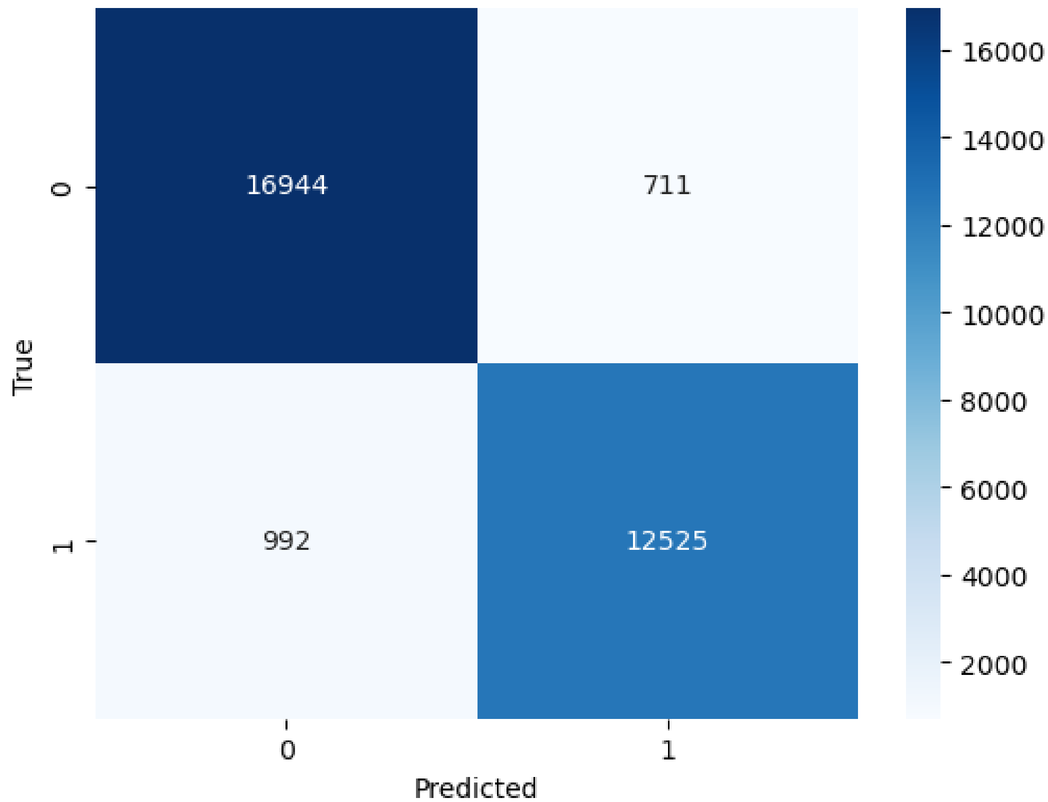
[59]:
```
import numpy as np
import matplotlib.pyplot as plt

# Remove the low important features from the DataFrame
excluded_features = ['purpose_of_travel_academic',␣
 ↪'purpose_of_travel_business', 'purpose_of_travel_tourism',
                     'purpose_of_travel_aviation',␣
 ↪'purpose_of_travel_personal', 'Gender_Female', 'Gender_Male',
                     'Type Of Booking_Not defined']
importance_df_filtered = importance_df[~importance_df['Feature'].
 ↪isin(excluded_features)]

# Get the filtered importances and features
importances_filtered = importance_df_filtered['Importance'].values
features_filtered = importance_df_filtered['Feature'].values
indices_filtered = np.argsort(importances_filtered)

plt.figure(figsize=(12, 8))
plt.barh(range(len(indices_filtered)), importances_filtered[indices_filtered],␣
 ↪align='center')
plt.yticks(range(len(indices_filtered)), [features_filtered[i] for i in␣
 ↪indices_filtered])
plt.xlabel('Relative Importance')
```

```
plt.show()
```



6 2.d. Random Forest Model Performance

[60]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



[61]:

```
from sklearn.metrics import recall_score, precision_score,␣
↪balanced_accuracy_score

# Calculate metrics
sensitivity_rf = recall_score(y_test, y_pred)  # Sensitivity, also known as␣
↪recall
specificity_rf = cm[0, 0] / (cm[0, 0] + cm[0, 1])
precision_rf = precision_score(y_test, y_pred)
balanced_accuracy_rf = balanced_accuracy_score(y_test, y_pred)

# Display metrics
print(f'Accuracy (Random Forest): {accuracy:.4f}')
print(f'Sensitivity (Random Forest): {sensitivity_rf:.4f}')
print(f'Specificity (Random Forest): {specificity_rf:.4f}')
print(f'Precision (Random Forest): {precision_rf:.4f}')
print(f'Balanced Accuracy (Random Forest): {balanced_accuracy_rf:.4f}')
```

```
Accuracy (Random Forest): 0.9454
Sensitivity (Random Forest): 0.9266
Specificity (Random Forest): 0.9597
Precision (Random Forest): 0.9463
```

```
Balanced Accuracy (Random Forest): 0.9432
```

7 2.e. Hyperparameter Tuning (Do not run this, too slow)

[11]:
```python
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid to search
param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the RandomForestClassifier
rf = RandomForestClassifier(random_state=240)

# Initialize the GridSearchCV object with the RandomForestClassifier and the␣
 ↪parameter grid
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1, verbose=2)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best parameters
print("Best parameters found: ", grid_search.best_params_)

# Print the best score (accuracy)
print("Best score found: ", grid_search.best_score_)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Best parameters found:  {'bootstrap': False, 'max_depth': None,
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Best score found:  0.9468734875697129
```

7.1 2.e.i. Best score found: 0.9468. Only marginally better(0.0014) than with the default hyperparameter(0.9454). Not worth the computational cost.

8 2.f. Sample Prediction with Random Forest Model

[64]:
```python
# Create a dictionary with the fictional guest's information
guest = {
    'Age': 30,
    'Gender_Female': 0,
    'Gender_Male': 1,
```

```
    'purpose_of_travel_academic': 0,
    'purpose_of_travel_aviation': 1,
    'purpose_of_travel_business': 0,
    'purpose_of_travel_personal': 0,
    'purpose_of_travel_tourism': 0,
    'Type of Travel_Group Travel': 0,
    'Type of Travel_Personal Travel': 1,
    'Type Of Booking_Group bookings': 1,
    'Type Of Booking_Individual/Couple': 0,
    'Type Of Booking_Not defined': 0,
    'Hotel wifi service': 5,
    'Departure/Arrival  convenience': 5,
    'Ease of Online booking': 4,
    'Hotel location': 3,
    'Food and drink': 4,
    'Stay comfort': 5,
    'Common Room entertainment': 3,
    'Checkin/Checkout service': 4,
    'Other service': 4,
    'Cleanliness': 4 }
```

[65]:

```
# Create a DataFrame from the guest's information
guest_df = pd.DataFrame(guest, index=[0])

# Reorder the columns to match the original training data
guest_df = guest_df[X.columns]

# Make a prediction using the random forest model
prediction = clf.predict(guest_df)

# Print the prediction (1 for satisfied, 0 for neutral or dissatisfied)
if prediction[0] == 1:
    print("The fictional guest is predicted to be satisfied with the hotel.")
else:
    print("The fictional guest is predicted to be neutral or dissatisfied with␣
 ↪the hotel.")
```

The fictional guest is predicted to be satisfied with the hotel.

[66]:

```
# Make a prediction of probability using the random forest model
prediction_prob = clf.predict_proba(guest_df)[0][1]

# Print the predicted probability
print("The probability of the fictional guest being satisfied is:",␣
 ↪round(prediction_prob,4))
```

The probability of the fictional guest being satisfied is: 0.78

9 3. Decision Tree Classification

[67]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.tree import export_graphviz
import graphviz

# Initialize and fit the DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(random_state=240, max_depth=3) # >>> Adjust␣
 ↪depth for accuracy!! <<<
tree_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_tree = tree_clf.predict(X_test)

# Calculate the accuracy and print the classification report
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print("Accuracy: ", accuracy_tree)
print(classification_report(y_test, y_pred_tree))
```

```
Accuracy:  0.8767483639163351
              precision    recall  f1-score   support

           0       0.86      0.93      0.90     17655
           1       0.90      0.80      0.85     13517

    accuracy                           0.88     31172
   macro avg       0.88      0.87      0.87     31172
weighted avg       0.88      0.88      0.88     31172
```

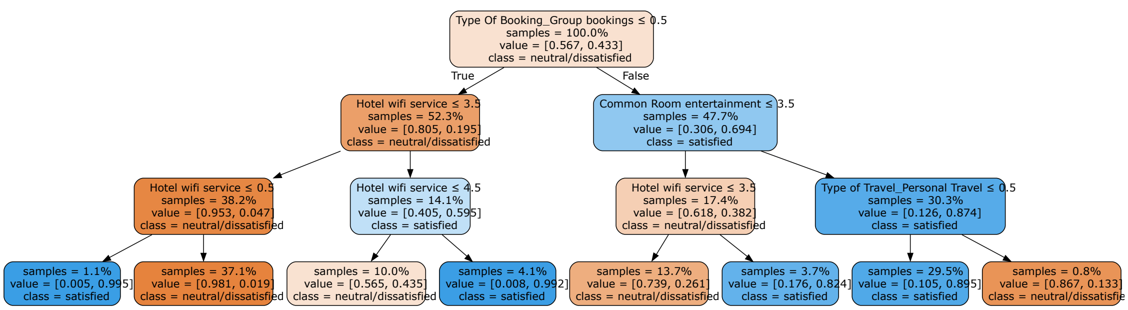
[68]:
```python
# Calculate the confusion matrix
cm_tree = confusion_matrix(y_test, y_pred_tree)
tn_tree, fp_tree, fn_tree, tp_tree = cm_tree.ravel()

# Calculate metrics
sensitivity_tree = recall_score(y_test, y_pred_tree)
specificity_tree = tn_tree / (tn_tree + fp_tree)
precision_tree = precision_score(y_test, y_pred_tree)
balanced_accuracy_tree = balanced_accuracy_score(y_test, y_pred_tree)

# Display metrics
print(f'Accuracy (Decision Tree): {accuracy_tree:.4f}')
print(f'Sensitivity (Decision Tree): {sensitivity_tree:.4f}')
print(f'Specificity (Decision Tree): {specificity_tree:.4f}')
print(f'Precision (Decision Tree): {precision_tree:.4f}')
print(f'Balanced Accuracy (Decision Tree): {balanced_accuracy_tree:.4f}')
```

```
Accuracy (Decision Tree): 0.8767
Sensitivity (Decision Tree): 0.8017
Specificity (Decision Tree): 0.9342
Precision (Decision Tree): 0.9032
Balanced Accuracy (Decision Tree): 0.8680
```

10 3.a. Decision Tree Visualization

[69]:
```python
# Visualize the decision tree with probabilities
dot_data = tree.export_graphviz(tree_clf, out_file=None,
                                feature_names=X.columns,
                                class_names=['neutral/dissatisfied',␣
 ↪'satisfied'],
                                filled=True, rounded=True,
                                special_characters=True,
                                proportion=True,  # To show probabilities
                                impurity=False,  # To remove gini impurity
                                )
graph = graphviz.Source(dot_data)
graph
```

[69]:



11 3.a.i. Decision Tree Interpretation

At the root node, the decision tree starts by checking if the "Type Of Booking_Individual/Couple" is less than or equal to 0.5. This feature is binary, so values <= 0.5 mean the booking is NOT "Individual/Couple" (i.e., it is either "Group bookings" or "Not defined"). The root node considers all samples (100%), where 56.7% are neutral/dissatisfied, and 43.3% are satisfied. Since the majority class is neutral/dissatisfied, it is labeled as the class for this node.

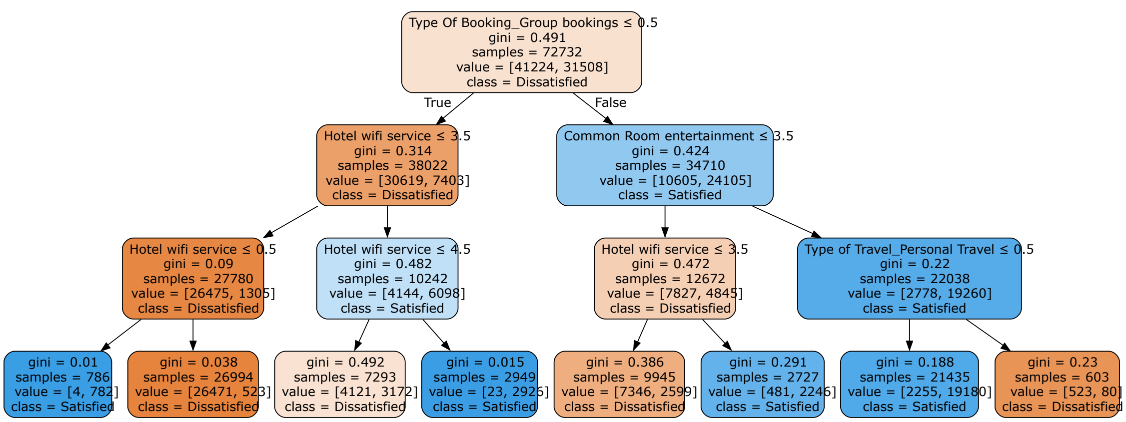
First "True" child node: If the condition at the root node is True (Type Of Booking_Individual/Couple <= 0.5), the decision tree moves to this node. Here, it checks if "Common Room entertainment" is less than or equal to 3.5. At this node, 55% of the samples from the parent node are considered, with 36.4% being neutral/dissatisfied and 63.6% being satisfied. As the majority class is satisfied, this node is labeled as "satisfied".

First "False" child node: If the condition at the root node is False (Type Of Booking_Individual/Couple > 0.5), the decision tree moves to this node. Here, it checks if "Hotel wifi service" is less than or equal to 3.5. At this node, 45% of the samples from the parent node are considered, with 81.4% being neutral/dissatisfied and 18.6% being satisfied. As the majority class is neutral/dissatisfied, this node is labeled as "neutral/dissatisfied".

12 3.b. Another Decision Tree Visualization

[70]:
```python
# Initialize and fit the DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(random_state=240, max_depth=3)  # >>> Adjust depth for tree size!! <<<
dt_clf.fit(X_train, y_train)

# Export the tree structure to a DOT format
dot_data = export_graphviz(dt_clf, out_file=None, feature_names=X.columns, class_names=['Dissatisfied', 'Satisfied'], filled=True, rounded=True, special_characters=True)

# Visualize the tree using graphviz
graph = graphviz.Source(dot_data)
graph
```

[70]:



13 4. Logistic Regression

[71]:
```python
satisfaction = pd.read_csv('hotel_satisfaction.csv')
satisfaction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 17 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   id                              103904 non-null  int64
```

```
 1   Gender                          103904 non-null  object
 2   Age                             103904 non-null  int64
 3   purpose_of_travel               103904 non-null  object
 4   Type of Travel                  103904 non-null  object
 5   Type Of Booking                 103904 non-null  object
 6   Hotel wifi service              103904 non-null  int64
 7   Departure/Arrival  convenience  103904 non-null  int64
 8   Ease of Online booking          103904 non-null  int64
 9   Hotel location                  103904 non-null  int64
 10  Food and drink                  103904 non-null  int64
 11  Stay comfort                    103904 non-null  int64
 12  Common Room entertainment       103904 non-null  int64
 13  Checkin/Checkout service        103904 non-null  int64
 14  Other service                   103904 non-null  int64
 15  Cleanliness                     103904 non-null  int64
 16  satisfaction                    103904 non-null  object
dtypes: int64(12), object(5)
memory usage: 13.5+ MB
```

[72]:
```python
# Drop the id column as it doesn't contribute to the model
satisfaction_1 = satisfaction.drop(columns=['id'])

# Value_counts
satisfaction_2 = satisfaction.drop(columns=['id', 'Age'])

for col in satisfaction_2.columns:
    print(f"Value counts for column {col}:")
    print(satisfaction_2[col].value_counts())
    print()
```

```
Value counts for column Gender:
Female    52727
Male      51177
Name: Gender, dtype: int64

Value counts for column purpose_of_travel:
tourism     32053
academic    27219
business    21238
aviation    13846
personal     9548
Name: purpose_of_travel, dtype: int64

Value counts for column Type of Travel:
Group Travel       71655
Personal Travel    32249
Name: Type of Travel, dtype: int64
```

```
Value counts for column Type Of Booking:
Group bookings       49665
Individual/Couple    46745
Not defined           7494
Name: Type Of Booking, dtype: int64

Value counts for column Hotel wifi service:
3    25868
2    25830
4    19794
1    17840
5    11469
0     3103
Name: Hotel wifi service, dtype: int64

Value counts for column Departure/Arrival  convenience:
4    25546
5    22403
3    17966
2    17191
1    15498
0     5300
Name: Departure/Arrival  convenience, dtype: int64

Value counts for column Ease of Online booking:
3    24449
2    24021
4    19571
1    17525
5    13851
0     4487
Name: Ease of Online booking, dtype: int64

Value counts for column Hotel location:
3    28577
4    24426
2    19459
1    17562
5    13879
0        1
Name: Hotel location, dtype: int64

Value counts for column Food and drink:
4    24359
5    22313
3    22300
2    21988
1    12837
```

```
0      107
Name: Food and drink, dtype: int64

Value counts for column Stay comfort:
4    31765
5    26470
3    18696
2    14897
1    12075
0        1
Name: Stay comfort, dtype: int64

Value counts for column Common Room entertainment:
4    29423
5    25213
3    19139
2    17637
1    12478
0       14
Name: Common Room entertainment, dtype: int64

Value counts for column Checkin/Checkout service:
4    29055
3    28446
5    20619
2    12893
1    12890
0        1
Name: Checkin/Checkout service, dtype: int64

Value counts for column Other service:
4    37945
5    27116
3    20299
2    11457
1     7084
0        3
Name: Other service, dtype: int64

Value counts for column Cleanliness:
4    27179
3    24574
5    22689
2    16132
1    13318
0       12
Name: Cleanliness, dtype: int64
```

```
Value counts for column satisfaction:
neutral or dissatisfied    58879
satisfied                  45025
Name: satisfaction, dtype: int64
```

Only two outcomes, neutral or dissatisfied 56.67%, satisfied 43.33%

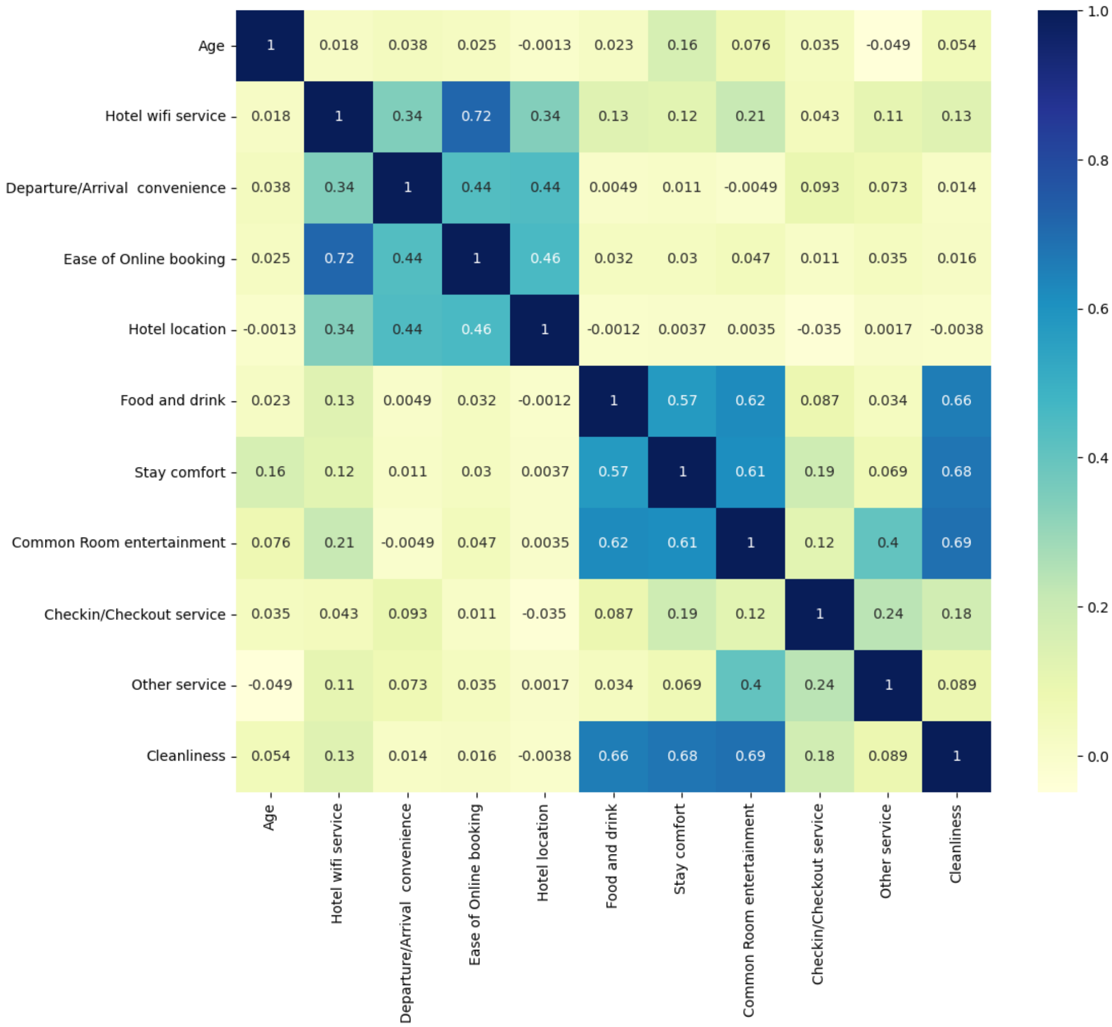
[73]:
```python
numeric_cols = satisfaction_1.select_dtypes(include = ["int64"]).columns
corr_matrix = satisfaction_1[numeric_cols].corr()

# Display the correlation matrix
corr_matrix
```

[73]:
```
                                        Age  Hotel wifi service  \
Age                                1.000000            0.017859
Hotel wifi service                 0.017859            1.000000
Departure/Arrival  convenience     0.038125            0.343845
Ease of Online booking             0.024842            0.715856
Hotel location                    -0.001330            0.336248
Food and drink                     0.023000            0.134718
Stay comfort                       0.160277            0.122658
Common Room entertainment          0.076444            0.209321
Checkin/Checkout service           0.035482            0.043193
Other service                     -0.049427            0.110441
Cleanliness                        0.053611            0.132698

                                Departure/Arrival  convenience  \
Age                                                   0.038125
Hotel wifi service                                    0.343845
Departure/Arrival  convenience                        1.000000
Ease of Online booking                                0.436961
Hotel location                                        0.444757
Food and drink                                        0.004906
Stay comfort                                          0.011344
Common Room entertainment                            -0.004861
Checkin/Checkout service                              0.093333
Other service                                         0.073318
Cleanliness                                           0.014292

                                Ease of Online booking  Hotel location  \
Age                                           0.024842       -0.001330
Hotel wifi service                            0.715856        0.336248
Departure/Arrival  convenience                0.436961        0.444757
Ease of Online booking                        1.000000        0.458655
Hotel location                                0.458655        1.000000
Food and drink                                0.031873       -0.001159
Stay comfort                                  0.030014        0.003669
```

```
Common Room entertainment                    0.047032       0.003517
Checkin/Checkout service                     0.011081      -0.035427
Other service                                0.035272       0.001681
Cleanliness                                  0.016179      -0.003830

                                Food and drink  Stay comfort  \
Age                                   0.023000      0.160277
Hotel wifi service                    0.134718      0.122658
Departure/Arrival  convenience        0.004906      0.011344
Ease of Online booking                0.031873      0.030014
Hotel location                       -0.001159      0.003669
Food and drink                        1.000000      0.574556
Stay comfort                          0.574556      1.000000
Common Room entertainment             0.622512      0.610590
Checkin/Checkout service              0.087299      0.191854
Other service                         0.033993      0.069218
Cleanliness                           0.657760      0.678534

                                Common Room entertainment  \
Age                                              0.076444
Hotel wifi service                               0.209321
Departure/Arrival  convenience                  -0.004861
Ease of Online booking                           0.047032
Hotel location                                   0.003517
Food and drink                                   0.622512
Stay comfort                                     0.610590
Common Room entertainment                        1.000000
Checkin/Checkout service                         0.120867
Other service                                    0.404855
Cleanliness                                      0.691815

                                Checkin/Checkout service  Other service  \
Age                                             0.035482      -0.049427
Hotel wifi service                              0.043193       0.110441
Departure/Arrival  convenience                  0.093333       0.073318
Ease of Online booking                          0.011081       0.035272
Hotel location                                 -0.035427       0.001681
Food and drink                                  0.087299       0.033993
Stay comfort                                    0.191854       0.069218
Common Room entertainment                       0.120867       0.404855
Checkin/Checkout service                        1.000000       0.237197
Other service                                   0.237197       1.000000
Cleanliness                                     0.179583       0.088779

                                Cleanliness
Age                                0.053611
Hotel wifi service                 0.132698
```

```
Departure/Arrival  convenience     0.014292
Ease of Online booking             0.016179
Hotel location                    -0.003830
Food and drink                     0.657760
Stay comfort                       0.678534
Common Room entertainment          0.691815
Checkin/Checkout service           0.179583
Other service                      0.088779
Cleanliness                        1.000000
```

[74]:
```python
# Correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap="YlGnBu")
plt.show()
```



Based on the correlation matrix, there are a few pairs of variables that have relatively high correlations:

Hotel wifi service and Ease of Online booking (0.72) Hotel location and Departure/Arrival convenience (0.44) Stay comfort and Common Room entertainment (0.62) Stay comfort and Cleanliness (0.68) Common Room entertainment and Cleanliness (0.69)

A correlation of 0.72 between Hotel wifi service and Ease of Online booking is relatively high, but not high enough to present a significant problem with multicollinearity. Therefore, it is not necessary to remove any variables.

[75]:
```
from sklearn.model_selection import train_test_split

# Dummify categorical variables
satisfaction_dummies = pd.get_dummies(satisfaction_1, columns=['Gender',␣
↪'purpose_of_travel', 'Type of Travel', 'Type Of Booking'], drop_first=True)

# Combine the dummified numerical and categorical features
features = satisfaction_dummies.drop(columns=['satisfaction'])
target = satisfaction_dummies['satisfaction']

# Create the train-test split with 70% train and 30% test data
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
↪test_size=0.3, random_state=240)
```

[76]:
```
satisfaction_1.groupby('satisfaction').mean()
```

```
C:\Users\watso\AppData\Local\Temp\ipykernel_51608\1920881571.py:1:
FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is
deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the
function.
  satisfaction_1.groupby('satisfaction').mean()
```

[76]:
```
                                 Age  Hotel wifi service  \
satisfaction
neutral or dissatisfied    37.566688            2.399633
satisfied                  41.750583            3.161288

                         Departure/Arrival  convenience  \
satisfaction
neutral or dissatisfied                         3.129112
satisfied                                       2.970305

                         Ease of Online booking  Hotel location  \
satisfaction
neutral or dissatisfied                2.546850        2.976121
satisfied                              3.031582        2.977879

                         Food and drink  Stay comfort  \
```

```
satisfaction
neutral or dissatisfied          2.95805      3.036295
satisfied                        3.52131      3.966530

                         Common Room entertainment  Checkin/Checkout service  \
satisfaction
neutral or dissatisfied                   2.894156                  3.042952
satisfied                                 3.964931                  3.646041

                         Other service  Cleanliness
satisfaction
neutral or dissatisfied       3.388814     2.936123
satisfied                     3.969461     3.744342
```

Looking at these results, three independent variables that might have a strong impact on satisfaction are "Hotel wifi service", "Food and drink", and "Cleanliness".

Higher values for these variables are generally associated with higher satisfaction levels. For example, the mean "Hotel wifi service" score is 3.11 for satisfied customers and 2.80 for dissatisfied customers. This suggests that having a good wifi service is likely to have a positive impact on customer satisfaction. Similarly, the mean "Food and drink" score is 3.95 for satisfied customers and 3.17 for dissatisfied customers, while the mean "Cleanliness" score is 4.35 for satisfied customers and 3.03 for dissatisfied customers. This suggests that both good food and drink options and high levels of cleanliness are likely to have a strong positive impact on customer satisfaction.

14 4.a. Logistic Regression

[77]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
satisfaction = pd.read_csv('hotel_satisfaction.csv')

# Convert categorical variables to numerical using one-hot encoding
satisfaction = pd.get_dummies(satisfaction, columns=['Gender',
 ↪'purpose_of_travel', 'Type of Travel', 'Type Of Booking'], drop_first=True)

# Convert the satisfaction column to binary format
satisfaction['satisfaction'] = satisfaction['satisfaction'].map({'satisfied':
 ↪1, 'neutral or dissatisfied': 0})

# Split the dataset into features (X) and target (y) variables
X = satisfaction.drop(columns=['id', 'satisfaction'])
y = satisfaction['satisfaction']
```

```python
# Split the dataset into a training (70%) and testing (30%) set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=240)

# Initialize and fit the logistic regression model
lgr = LogisticRegression(random_state=240)
lgr.fit(X_train, y_train)

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

# Calculate the accuracy and print the classification report
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
print(classification_report(y_test, y_pred))
```

```
Accuracy:  0.8436096496856152
              precision    recall  f1-score   support

           0       0.86      0.86      0.86     17655
           1       0.82      0.82      0.82     13517

    accuracy                           0.84     31172
   macro avg       0.84      0.84      0.84     31172
weighted avg       0.84      0.84      0.84     31172


C:\Users\watso\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

15 4.b. Sample Prediction with Logistic Regression

[78]:
```python
# Create a dictionary with the fictional guest's information
guest = {
    'Age': 30,
    'Gender_Female': 0,
    'Gender_Male': 1,
    'purpose_of_travel_academic': 0,
```

```
    'purpose_of_travel_aviation': 1,
    'purpose_of_travel_business': 0,
    'purpose_of_travel_personal': 0,
    'purpose_of_travel_tourism': 0,
    'Type of Travel_Group Travel': 0,
    'Type of Travel_Personal Travel': 1,
    'Type Of Booking_Group bookings': 1,
    'Type Of Booking_Individual/Couple': 0,
    'Type Of Booking_Not defined': 0,
    'Hotel wifi service': 5,
    'Departure/Arrival  convenience': 5,
    'Ease of Online booking': 4,
    'Hotel location': 3,
    'Food and drink': 4,
    'Stay comfort': 5,
    'Common Room entertainment': 3,
    'Checkin/Checkout service': 4,
    'Other service': 4,
    'Cleanliness': 4 }
```

[79]:

```
# Create a DataFrame from the guest's information
guest_df = pd.DataFrame(guest, index=[0])

# Reorder the columns to match the original training data
guest_df = guest_df[X.columns]

# Make a prediction using the logistic Regression model
prediction = lgr.predict(guest_df)

# Print the prediction (1 for satisfied, 0 for neutral or dissatisfied)
if prediction[0] == 1:
    print("The fictional guest is predicted to be satisfied with the hotel.")
else:
    print("The fictional guest is predicted to be neutral or dissatisfied with␣
 ↪the hotel.")
```

```
The fictional guest is predicted to be satisfied with the hotel.
```

[80]:

```
# Make a prediction of probability using the logistic Regression model
prediction_prob = lgr.predict_proba(guest_df)[0][1]

# Print the predicted probability
print("The probability of the fictional guest being satisfied is:",␣
 ↪round(prediction_prob,4))
```

```
The probability of the fictional guest being satisfied is: 0.656
```

16 Summary

Comparing the three models' performance, the random forest model scores the highest accuracy, and the feature importance analysis revealed that hotel WiFi service, type of booking, and common room entertainment were among the most influential factors in determining guest satisfaction. The decision tree and logistic regression models also confirm the similar results.

Based on these results, the conference guests can draw several conclusions about the hospitality side of the business. First, it is evident that hotel WiFi service is crucial to guest satisfaction, suggesting that investing in reliable high-speed WiFi can significantly improve guests' experiences. Second, the type of booking (group bookings or individual/couple bookings) also plays a significant role in satisfaction, which may help park managers tailor marketing efforts and services to group bookings such as families, packaged tours, social events, and travel agencies. Thirdly, the common room entertainment is also very significant in determining guest satisfaction; offering various enjoyable and engaging entertainment options in common areas can enhance guests' experience.

The correlation among the various factors also reveals that several pairs of factors positively correlate with each other, potentially to a synergistic effect in enhancing guest satisfaction such as 'Hotel WiFi service' and 'Ease of Online booking', the netizens' lifestyle and their WiFi quality demands usually go hand in hand. 'Common Room Entertainment' with 'Cleanliness', 'Stay comfort', and 'Food and drink'. Suggesting that when guests are satisfied with the common Room entertainment, they also tend to be more likely satisfied with food and drink, cleanliness, and the comfort of the hotel. Although it is common sense that when ensuring the hotel's WiFi quality, the hotel's online booking services shouldn't be too complicated, we should also remember the theme park hotel's niche since it is not a restaurant or airport hotel; it mainly serves the theme park guests for festivities, social events, and group entertainments.

Park managers can use these insights to refine their marketing approaches and improve the overall guest experience by investing in the most important facilities (WiFi, common room entertainment), promoting group guests, and advertising in family and social themes to encourage more guests that are likely to rate the hotel favorably. In short, invest in what people like the most and attract the right guests that will likely enjoy it. If we have more time, we could also do a more in-depth interaction term analysis to find out how much the interaction effect the various factors could have on guest satisfaction.

[ ]:

```

```

Strategic memo

From 2020 to 2023, Lobster Land Co.'s management has discussed the possibility of expanding its brand presence overseas, with the aim of increasing its brand's scope and diversifying for future growth. To aid in future decision-making and planning, this strategic memo offers a qualitative analysis of the "Golden Arch Hotel" case study, which shares similarities with Lobster Land's proposed growth strategy. The memo concludes with strategic recommendations on expanding Lobster Land's business into the hotel industry.

The Golden Arch Hotel project was a strategy plan aimed for "diversification" by the McDonalds's corporation in 2001. The hotel, however, closed two years after its opening in Switzerland due to difficulties in targeting the key customer segments. The following SWOT analysis brings brief summary and evaluation on the strength, weakness, opportunities and threats of the Golden Arch Hotel in 2001.

Strength:

1. The McDonalds restaurants in Switzerland has already built a huge customer base of 74 million. The wide recognition of the brand brings the customer a sense of familiarity and therefore more recognition and trustiness when they are making accommodation choices. The hotels are also equipped with the same brand image as the restaurants, which is convenience, hospitality, and cleanliness.
2. The Golden Arch Hotel is equipped with cutting-edge technology at the time to contribute to customer experience, including adjustable beds, electronic key, wireless keyboard and automatic check-in process. Such technology enhances the brand essence of convenience and contemporary, making it an appealing option for younger guests. Additionally, these added amenities elevate the hotel's positioning to that of a four-star establishment, distinguishing it from conventional hotels.

Weakness:

1. The McDonald brand at the time has a limited brand breath, where customers associate the brand with the "fast, clean, and friendly" fast food corporation. Such association reduces the sense of reliability, and coziness, nor is the brand associated with luxurious experience. While it is positioned as a four-star hotel, the customers are not satisfied

with the relatively high room rate and lacking services and food options.

2. The hotel's name was inadequately glocalized, as one Swiss customer noted that the word "arches" was difficult to translate and understand in German and could even imply negative connotations.

Opportunities:

1. The hotel's proximity to the airport enhances its accessibility, making it an attractive option for accommodating airline crews and signing contracts with companies to host business travelers. This strategy helps to create a more stable revenue source, reducing reliance on individual travelers.
2. Additionally, the hotel can adjust its market segmentation strategies to focus on families and young individuals who are typically drawn to clean, friendly, and experiential hotels.

Threats:

1. Despite an optimistic outlook, the hotel market in Switzerland is highly competitive. The Golden Arch Hotel must vie with numerous major hotel corporations such as IBIS-Hotel, Etap-Hotel, and Novotel-Hotel. These competitors have already implemented sustainable strategies, such as frequent guest rewards programs and partnering with travel agencies.

Unlike McDonalds restaurants and Golden Arch Hotel operate in different industries, Lobster Land, as a theme park has significant potential to expand into the hotel industry. However, Lobster Land started as a local, humble seaside amusement park, the corporation should first consider scaling up domestically first to minimize unforeseen risks and uncertainties before going overseas. With reference to the Golden Arch Hotel case, I will provide recommendations for expanding Lobster Land Co. in terms of market analysis, segmentation strategies, competition assessment, and pricing considerations.

With its established reputation in the Boston region and cities along the northeast coast, Lobster Land has built up a substantial customer base. The launch of Lobster Land theme hotels is likely to be well-received by the public. Due to lack of affordable and accessible hotels near Lobster Land in South Maine, most visitors are either Maine residents or citizens from neighboring states. The Lobster Land hotel should be positioned as a family-friendly, year-round establishment. Similar as the courteous, friendly but not overwhelming service provided in the hotel, the amenities and interior should be designed to be homelike and delightful, fitting in with its niche of affordable, family friendly accommodation. While eliminating unnecessary technologies such as adjustable beds and wireless keyboards, the network service should be fast and adequate to support habits of modern life.

One option for increasing revenue at Lobster Land Co. is to offer bundled packages of hotel rooms and theme park tickets during the operating months. During the off-season, the hotel can corporate with tour agencies to provide guests with activities like whale watching in October, March, and April, and snow activities in the colder days. To further enhance the immersive experience, the hotel should offer more theme-related activities, such as partnering with local restaurants specializing in lobster dishes. The hotel, however, should not eliminate the food options, not only because family members may have a variety of appetite and food intolerance, but also could sink the relaxed experience.

In terms of competitors, there are no major theme park chain along the northeast coast, nor there are many onsite theme park hotels. While Six Flags is a popular establishment in New England, its business model focuses on attracting less customers who spend more on site by providing thrilling roller coaster rides, which differentiates the customer segmentation from Lobster Land.

In [ ]:
```
%%shell
jupyter nbconvert --to html /content/Statistical_Testing.ipynb
```

```
[NbConvertApp] Converting notebook /content/Statistical_Testing.ipynb to html
[NbConvertApp] Writing 675891 bytes to /content/Statistical_Testing.html
```

Out[ ]:

Statistical Testing

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
```

In [ ]:
```
pics = pd.read_csv("/content/promo_pics.csv")
pics.head()
```

Out[ ]:

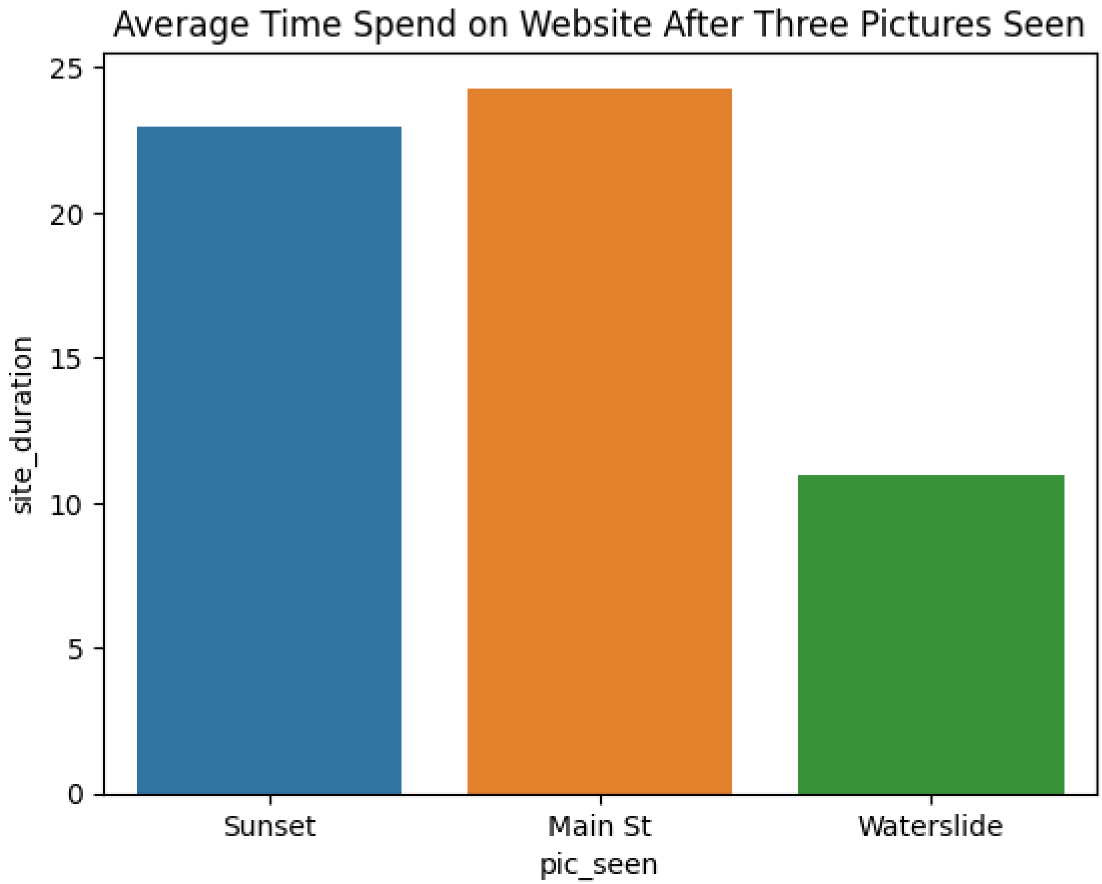
	recipient	pic_seen	site_duration	spend	register
0	1	Sunset	18.20	16.60	0
1	2	Main St	28.61	15.30	0
2	3	Waterslide	10.90	16.32	1
3	4	Waterslide	11.30	22.62	0
4	5	Sunset	19.70	17.30	0

In [ ]:
```
pics.isnull().any()
```

Out[ ]:
```
recipient        False
pic_seen         False
site_duration    False
spend            False
register         False
dtype: bool
```

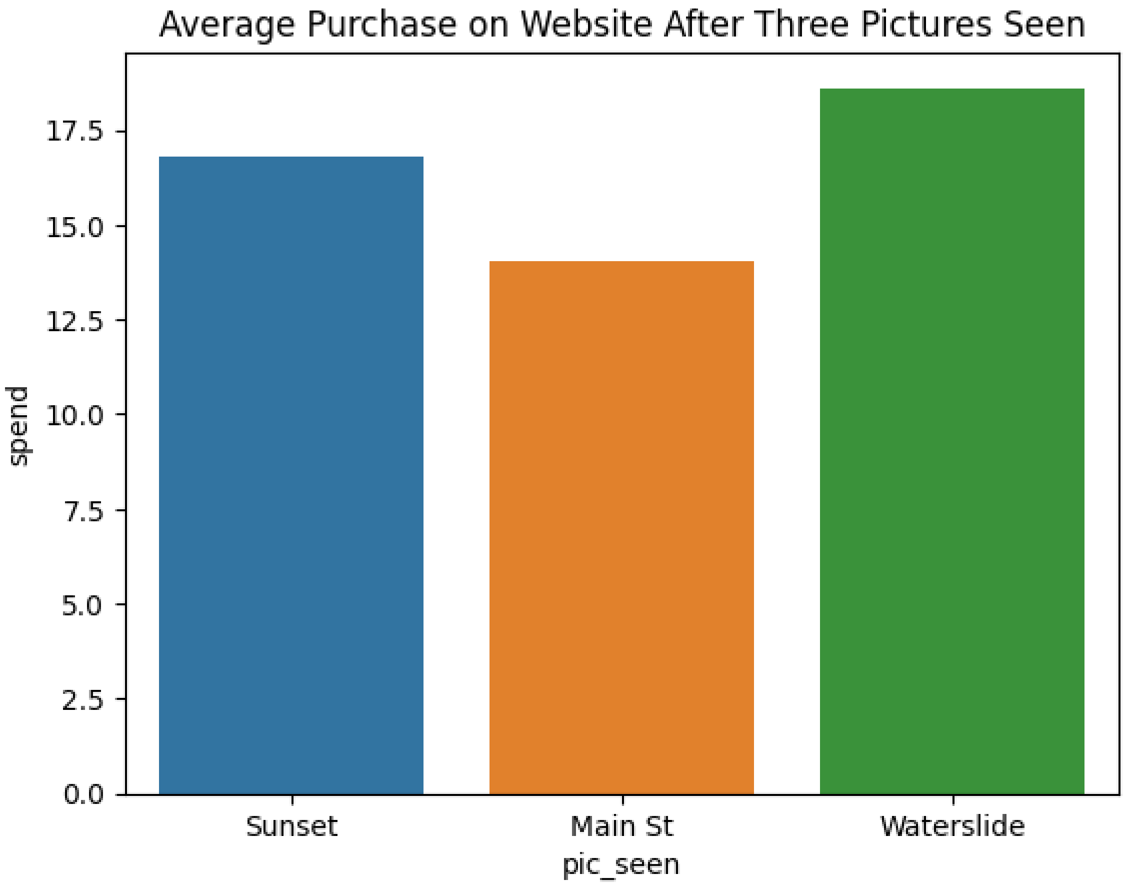
In [ ]:
```
# Bar Plot of Average Website Duration Time of 3 pics to see which is the best
sns.barplot(x = 'pic_seen', y = 'site_duration', errorbar = None, data = pics)
plt.title('Average Time Spend on Website After Three Pictures Seen')
```

Out[ ]:
```
Text(0.5, 1.0, 'Average Time Spend on Website After Three Pictures Seen')
```



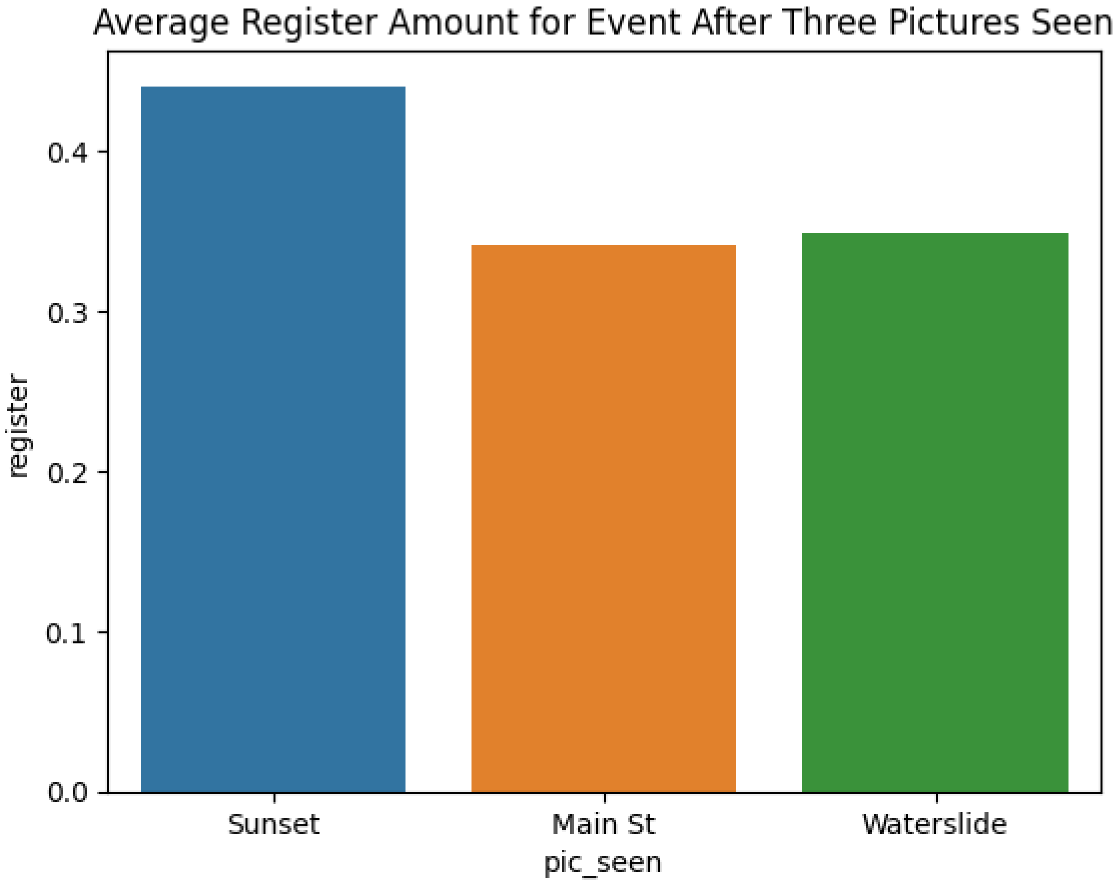
In [ ]:
```
# Bar Plot of Average Website Purchase Amount of 3 pics to see which is the bes
sns.barplot(x = 'pic_seen', y = 'spend', errorbar = None, data = pics)
plt.title('Average Purchase on Website After Three Pictures Seen')
```

Out[ ]:
```
Text(0.5, 1.0, 'Average Purchase on Website After Three Pictures Seen')
```



In [ ]:
```
# Bar Plot of Average Register Amount of 3 pics to see which is the best for th
sns.barplot(x = 'pic_seen', y = 'register', errorbar = None, data = pics)
plt.title('Average Register Amount for Event After Three Pictures Seen')
```

Out[ ]:
```
Text(0.5, 1.0, 'Average Register Amount for Event After Three Pictures Seen')
```



In [ ]:

```
# A/B Test between "Sunset" and "Main St"
t_1v2, p_1v2 = stats.ttest_ind(pics.loc[pics['pic_seen'] == 'Sunset', 'register
                               pics.loc[pics['pic_seen'] == 'Main St', 'registe
                               equal_var = False)
print('t value of "Sunset" vs "Main St" is: ', t_1v2, '\n',
      'p value of "Sunset" in Action" vs "Main St" is: ', p_1v2)
```

```
t value of "Sunset" vs "Main St" is:  4.844354466304721
 p value of "Sunset" in Action" vs "Main St" is:  1.357059422760036e-06
```

In [ ]:

```
# A/B Test between "Sunset" and "Waterslide"
t_1v2, p_1v2 = stats.ttest_ind(pics.loc[pics['pic_seen'] == 'Sunset', 'register
                               pics.loc[pics['pic_seen'] == 'Waterslide', 'regi
                               equal_var = False)
print('t value of "Sunset" vs "Waterslide" is: ', t_1v2, '\n',
      'p value of "Sunset" in Action" vs "Waterslide" is: ', p_1v2)
```

```
t value of "Sunset" vs "Waterslide" is:  4.483983175631638
 p value of "Sunset" in Action" vs "Waterslide" is:  7.69613709155986e-06
```

In [ ]:

```
# A/B Test between "Main St" and "Waterslide"
t_1v2, p_1v2 = stats.ttest_ind(pics.loc[pics['pic_seen'] == 'Main St', 'registe
                               pics.loc[pics['pic_seen'] == 'Waterslide', 'regi
                               equal_var = False)
print('t value of "Main St" vs "Waterslide" is: ', t_1v2, '\n',
      'p value of "Main St" in Action" vs "Waterslide" is: ', p_1v2)
```

```
t value of "Main St" vs "Waterslide" is:  -0.3546036791692854
 p value of "Main St" in Action" vs "Waterslide" is:  0.7229192023249931
```

> To determine the most effective picture for reaching customers, I first cleaned the dataset to ensure no missing values were present. The second step is to determine whether the different pictures on the website actually have significance in customer retention. I conducted three A/B tests on the website's three pictures, comparing them with each other. The p-values of the tests showed that there was no significant difference between the "Main St" and "Waterslide" websites. However, the "Sunset" website had a clear difference from the other two picture options, with a relatively high p-value. After concluding that the choice of pictures has a significant impact on marketing effectiveness, I utilized the seaborn package to generate barplots for the average number of three numeric factors. The results revealed that each picture had its own strengths in attracting customers. The "Main St" picture had the highest average time spent by customers, whereas the "Waterslide" picture led to the highest average purchase amount, and the "Sunset" picture had the highest average number of registrations. Furthermore, the "Sunset" picture not only won first place in average registrations but also ranked second in the other two barplots, making it the most competitive option among the pool of website pictures. Moreover, the number of registered accounts indicated the customers' connection to the brand and their availability for more marketing campaigns, which I consider the most critical factor among the three. Thus, it is highly advisible for Lobster Land to incorporate the "Sunset" picture into their emails to attain the optimal outcome.

Conclusion:

After analyzing the data of a series of demands put forward by Lobster Land Co., we suggest the following recommendations.

First of all, Lobster Land should make careful decisions regarding the safety feature of the rides to be a reliable and reputable service provider. When selecting a manufacturer for the equipment, Lobster Land should avoid those with high numbers of accidents. However, they should also be cautious when choosing the brand with the least number of accidents, as it may not be a popular choice. For high-risk rides such as coasters, go-karts, and waterslides, special safety instructions should be given to teenagers, who are more prone to injury.

Secondly, as Lobster Land Co. is currently developing hotel-park bundle packages to our customers, we analyzed a list skiing-themed hotels in different resorts to study on the hoteal categorization. For management and pricing purposes, the hotels are divided into three clusters. The Affordable Valley Inns are characterized by affordable rates with a trade-off on scenery views as they are located at a low altitude, These hotels are perfect for customers on a limited budget. The second cluster is the Ski Summit Lodges, which are ideal for ski enthusiasts who want to stay closer to the ski area and at a higher altitude for a better skiing experience. The third and most luxurious cluster is the Ski Majesty Grand, which offers a variety of choices on ski tracks and routes, making it the best choice for customers seeking a lavish and immersive experience.

Based on the significance of amenities and a fixed budget of $250 per room, the next step is to provide recommendations for hotel amenities to ensure that the hotel offers facilities that customers care about the most. The hotel should prioritize amenities such as a reliable WIFI network, a full buffet breakfast, open lot parking, a well-equipped gym, an air purifier, and a heated pool maintained at 84F. In addition, the hotel should consider offering add-on services such as flexible check-in/check-out times, shuttle bus services to downtown, and a VIP shopping agreement that provides benefits for high-end customers.

To estimate future revenue, we decided to use linear regression forecasting method on the annual net income for Hyatt and Hilton hotels group. The trend indicates a steady recovery from the COVID downturn of 2020-2021, and we are optimistic that both companies will exceed our forecast. Our projection for Hilton's net income is $754.62 million, while Hyatt Hotels is expected to reach $202.02 million.

To achieve a sustainable growth in business, we strongly advise hotel management to enhance overall guest satisfaction by investing in the essential amenities that contribute the most. The results of the classification analysis reveal that customer satisfaction is strongly influenced by the quality of hotel wifi service, common room entertainment, and comfort. A higher rating for these factors is generally associated with a higher satisfaction level. Additionally, the type of booking (group or individual/couple) also plays a significant role in satisfaction. Therefore, hotel managers should customize their marketing efforts and services to target group bookings, such as families, packaged tours, social events, and travel agencies.

Regarding marketing, we recommend sending out the proposed marketing emails in mid-May that include the website with the picture "Sunset". Statistical tests have proven that this picture significantly impacts customer retention and increases the number of registered accounts on the website. By establishing a stronger connection with customers, future marketing campaigns will reach a wider audience.

In case Lobster Land Co. intends to enlarge its business operation in the future, it is strongly recommended to enter the hospitality industry in the regions adjacent to the current theme park. Please refer to the Strategic Memo for a detailed analysis report.