ICE - Association Rules Analysis

DATA 3300

Student Name: Cabot Steward

Q1

Load the following libaries: Pandas, NumPy, Matplotlib.pyplot and mlxetend.frequent_patterns then import the skiut dataset

In [13]:
```
!pip install mlxtend
```

```
Requirement already satisfied: mlxtend in c:\users\tucke\anaconda3\lib\site-packages
(0.23.4)
Requirement already satisfied: scipy>=1.2.1 in c:\users\tucke\anaconda3\lib\site-pac
kages (from mlxtend) (1.13.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\tucke\anaconda3\lib\site-pa
ckages (from mlxtend) (1.26.4)
Requirement already satisfied: pandas>=0.24.2 in c:\users\tucke\anaconda3\lib\site-p
ackages (from mlxtend) (2.2.2)
Requirement already satisfied: scikit-learn>=1.3.1 in c:\users\tucke\anaconda3\lib\s
ite-packages (from mlxtend) (1.4.2)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\tucke\anaconda3\lib\sit
e-packages (from mlxtend) (3.8.4)
Requirement already satisfied: joblib>=0.13.2 in c:\users\tucke\anaconda3\lib\site-p
ackages (from mlxtend) (1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\tucke\anaconda3\lib\site
-packages (from matplotlib>=3.0.0->mlxtend) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\tucke\anaconda3\lib\site-pac
kages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\tucke\anaconda3\lib\sit
e-packages (from matplotlib>=3.0.0->mlxtend) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\tucke\anaconda3\lib\sit
e-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\tucke\anaconda3\lib\site-
packages (from matplotlib>=3.0.0->mlxtend) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\tucke\anaconda3\lib\site-packag
es (from matplotlib>=3.0.0->mlxtend) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\tucke\anaconda3\lib\site
-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\tucke\anaconda3\lib
\site-packages (from matplotlib>=3.0.0->mlxtend) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\tucke\anaconda3\lib\site-pac
kages (from pandas>=0.24.2->mlxtend) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\tucke\anaconda3\lib\site-p
ackages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\tucke\anaconda3\lib
\site-packages (from scikit-learn>=1.3.1->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\tucke\anaconda3\lib\site-package
s (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
```

In [14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori, association_rules
```

In [15]:
```
#import data set
#view headers
df = pd.read_csv("skiut_list.csv")
df.head()
```

Out[15]:

	SkierID	gender	Select_All_That_Apply
0	1001	m	sbasn, solit, sbird, altax
1	1002	m	sbird, altax
2	1003	f	powmo, altax
3	1004	m	solit, sbird, altax
4	1005	m	beavm, sbasn, powmo, solit, brton, sbird, alta...

Q2

You might notice that because the survey question related to ski resort visitation was in a "select all that apply" form, there is only one column that lists all of the ski resorts attended, deliminated by a comma.

This is not in a usable format for data analysis, so we need to create a dummy variable column for each of the 10 ski resorts, to indicate which of the 10 resorts each skier visited.

A

Create a series of lambda functions to search for the text string labels of each ski resort in each response, then create a new column for each resort containing "True" or "False" depending on whether the resort was listed in the response.

In [16]:
```
df['Beavm'] = df.apply(lambda x: "Beavm" in x.Select_All_That_Apply, axis=1)
df['Sbasn'] = df.apply(lambda x: "sbasn" in x.Select_All_That_Apply, axis=1)
df['Powmo'] = df.apply(lambda x: "powmo" in x.Select_All_That_Apply, axis=1)
df['Solit'] = df.apply(lambda x: "solit" in x.Select_All_That_Apply, axis=1)
df['Brton'] = df.apply(lambda x: "brton" in x.Select_All_That_Apply, axis=1)
df['Sbird'] = df.apply(lambda x: "sbird" in x.Select_All_That_Apply, axis=1)
df['Altax'] = df.apply(lambda x: "altax" in x.Select_All_That_Apply, axis=1)
df['ParkC'] = df.apply(lambda x: "parkc" in x.Select_All_That_Apply, axis=1)
df['Deerv'] = df.apply(lambda x: "deerv" in x.Select_All_That_Apply, axis=1)
df['Canyo'] = df.apply(lambda x: "canyo" in x.Select_All_That_Apply, axis=1)
```

In [17]:
```
#view headers
df.head()
```

Out[17]:

	SkierID	gender	Select_All_That_Apply	Beavm	Sbasn	Powmo	Solit	Brton	Sbird	Alt
0	1001	m	sbasn, solit, sbird, altax	False	True	False	True	False	True	Tr
1	1002	m	sbird, altax	False	False	False	False	False	True	Tr
2	1003	f	powmo, altax	False	False	True	False	False	False	Tr
3	1004	m	solit, sbird, altax	False	False	False	True	False	True	Tr
4	1005	m	beavm, sbasn, powmo, solit, brton, sbird, alta...	False	True	True	True	True	True	Tr

B

Now, convert the "True" and "False" label to a 1 or 0 for each column using the replace function. Do the same for gender, where m = 0, f = 1.

Also, there's no need to keep in the 'Select_All_That_Apply' column, so let's drop that.

In [18]:
```
df = df.drop('Select_All_That_Apply', axis=1)
df.replace({"m":0, "f":1}, inplace=True)
df.replace({True:1, False:0 }, inplace=True)

# df = df.rename({'': ''}, axis=1)
df.head()
```

```
C:\Users\tucke\AppData\Local\Temp\ipykernel_8576\17203458.py:2: FutureWarning: Downc
asting behavior in `replace` is deprecated and will be removed in a future version.
To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', Tru
e)`
  df.replace({"m":0, "f":1}, inplace=True)
C:\Users\tucke\AppData\Local\Temp\ipykernel_8576\17203458.py:3: FutureWarning: Downc
asting behavior in `replace` is deprecated and will be removed in a future version.
To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', Tru
e)`
  df.replace({True:1, False:0 }, inplace=True)
```

Out[18]:

	SkierID	gender	Beavm	Sbasn	Powmo	Solit	Brton	Sbird	Altax	ParkC	Deerv	Car
0	1001	0	0	1	0	1	0	1	1	0	0	
1	1002	0	0	0	0	0	0	1	1	0	0	
2	1003	1	0	0	1	0	0	0	1	0	0	
3	1004	0	0	0	0	1	0	1	1	0	0	
4	1005	0	0	1	1	1	1	1	1	1	1	

C

Before running any actual analysis, we should exclude the SkierID variable by setting it as our index. Do so below and then explain why.

In [37]:
```
#set SkierID as the index
df = df.set_index("SkierID")
#view headers
df.head()
```

Out[37]:

SkierID	gender	Beavm	Sbasn	Powmo	Solit	Brton	Sbird	Altax	ParkC	Deerv	Canyo
1001	0	0	1	0	1	0	1	1	0	0	0
1002	0	0	0	0	0	0	1	1	0	0	0
1003	1	0	0	1	0	0	0	1	0	0	0
1004	0	0	0	0	1	0	1	1	0	0	0
1005	0	0	1	1	1	1	1	1	1	1	1

D

Use the desribe() function to determine which resort in this dataset was visited most frequently. Identify the metric, its value, and its interpretation.

In [28]:
```
#produce descriptive stats for this dataset
df.describe()
df.mean()
print("Alta is most visited")
```

```
Alta is most visited
```

Q3

Generate frequent itemsets using the Apriori model in the mlxtend library. Set the minimum support threshold to 20%.

In [38]:
```
freq_rs = apriori(df, min_support = 0.2, use_colnames=True)
freq_rs
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Out[38]:

	support	itemsets
0	0.467674	(gender)
1	0.375325	(Sbasn)
2	0.404354	(Powmo)
3	0.344837	(Solit)
4	0.205077	(Brton)
5	0.468974	(Sbird)
6	0.474425	(Altax)
7	0.319959	(ParkC)
8	0.306459	(Deerv)
9	0.289218	(Canyo)
10	0.218419	(gender, Sbird)
11	0.222888	(gender, Altax)
12	0.228465	(Powmo, Sbasn)
13	0.210813	(Powmo, Sbird)
14	0.210655	(Powmo, Altax)
15	0.215599	(Solit, Sbird)
16	0.218736	(Solit, Altax)
17	0.341383	(Altax, Sbird)
18	0.229860	(ParkC, Deerv)
19	0.221905	(ParkC, Canyo)
20	0.219845	(Canyo, Deerv)

A

Which ski resort was visited by the highest percentage of survey respondents? What metric tells you this? In the code cell, write a formula to calculate support:

Mean can tell us this by averaging the value of 1 and 0s, the highest value will be the most common visited

In [39]:
```
# calculate the support count for Alta
support_count_alta = df['Altax'].sum()

# calculate the support for Alta
support_alta = support_count_alta / len(df)
```

```
# view the support value
print(f"Support for Alta: {support_alta:.3f}")
```

```
Support for Alta: 0.474
```

B

Is it suprising that this resort has the highest support value of all the resorts in the database? Explain.

I would say no, Alta is a popular resort. I would say for a dataset today this is no longer accurate, Solutide would likely be the most accurate.

C

What is the support value for {Alta, Snowbird}? What does this support value indicate?

Alta snowbird will have a higher support value

D

Let's try to replicate these findings by writing our own formula for calculating support:

In [40]:
```
itemset = df[(df['Altax'] == 1) & (df['Sbird'] == 1)] #calculate support count of i
support = len(itemset)/len(df) #calculate support of itemset
support
```

Out[40]:
```
0.34138302592381314
```

E

What is the support of {Female, Snowbird}? What does this support value indicate?

In [41]:
```
itemset = df[(df['gender'] == 1) & (df['Sbird'] == 1)]
support = len(itemset) / len(df)
support
```

Out[41]:
```
0.2184192178487672
```

21 percent of indiduals are female and support snowbird.

Q4

Now run an association rules analysis on your frequent rulesets object, set the minimum confidence threshold to 0.80.

A

How many rules are there?

In [45]:
```
# run an association rules analysis, conf => 0.8
rules = apriori(df, min_support = 0.8, use_colnames=True)
len(rules)
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:16
1: DeprecationWarning: DataFrames with non-bool types result in worse computationalp
erformance and their support might be discontinued in the future.Please use a DataFr
ame with bool type
  warnings.warn(
```

Out[45]:
```
0
```

No rules that are 80% confident

B

Okay, looks like 80% confidence was too high. Let's re-generate our frequent itemsets to have a minimum support of 0.05 and then rerun the analysis with a mimum confidence of 0.

In [48]:
```
rules = apriori(df, min_support = 0.05, use_colnames=True)
len(rules)
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:16
1: DeprecationWarning: DataFrames with non-bool types result in worse computationalp
erformance and their support might be discontinued in the future.Please use a DataFr
ame with bool type
  warnings.warn(
```

Out[48]:
```
112
```

In [50]:
```
rules = apriori(df, min_support = 0.00001, use_colnames=True)
len(rules)
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:16
1: DeprecationWarning: DataFrames with non-bool types result in worse computationalp
erformance and their support might be discontinued in the future.Please use a DataFr
ame with bool type
  warnings.warn(
```

Out[50]:
```
1023
```

C

Which three resorts were most frequently visited by the same respondent? Report the support value that tells you this. *Hint: Sort support from high to low.*

In [54]:
```
df_clean = df.drop(columns='gender') if 'gender' in df.columns else df

freq_rs = apriori(df_clean, min_support=0.05, use_colnames=True)
rules = association_rules(freq_rs, metric='confidence', min_threshold=0)

# Now filter for 3-resort combinations (2 in antecedent, 1 in consequent)
three_resort_combos = rules[(rules['antecedents'].apply(len) == 2) &
                            (rules['consequents'].apply(len) == 1)]

# Sort by support and show top result
top_three_resorts = three_resort_combos.sort_values(by='support', ascending=False)
top_three_resorts.head(1)
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:16
1: DeprecationWarning: DataFrames with non-bool types result in worse computationalp
erformance and their support might be discontinued in the future.Please use a DataFr
ame with bool type
  warnings.warn(
```

Out[54]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	r
236	(Canyo, Deerv)	(ParkC)	0.219845	0.319959	0.193985	0.88237	2.757756	

D

Does it make sense that those three would be frequently visted together? Explain.

Yes, it does make sense

E

Among females who visited two or more resorts during the season, which two-resort combination was most frequently visited? Report the support value that tells you this. *Hint: search for antecedents containing gender=f then sort by support.*

In [58]:
```
frequent_sets = apriori(df, min_support=0.01, use_colnames=True)

rules = association_rules(frequent_sets, metric="support", min_threshold=0)

female_resort_rules = rules[
    rules['antecedents'].apply(lambda x: 'gender' in x) &
    rules['consequents'].apply(lambda x: len(x) == 1 and 'gender' not in x)
]
```

```
female_resort_rules.sort_values(by='support', ascending=False).head()
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Out[58]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	r
10	(gender)	(Altax)	0.467674	0.474425	0.222888	0.476587	1.004558	
8	(gender)	(Sbird)	0.467674	0.468974	0.218419	0.467033	0.995861	
2	(gender)	(Powmo)	0.467674	0.404354	0.177347	0.379210	0.937816	
0	(gender)	(Sbasn)	0.467674	0.375325	0.170755	0.365115	0.972797	
246	(gender, Altax)	(Sbird)	0.222888	0.468974	0.161247	0.723447	1.542616	

Q5

Let's switch gears and focus on the confidence of rulesets.

A

What is the confidence for {Snowbasin} → {Beaver Mountain}? What does this value tell you?

In [64]:

```
df_apriori = df.astype(bool)
frequent_itemsets = apriori(df_apriori, min_support=0.01, use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.0

rule_confidence = rules[(rules['antecedents'] == frozenset({'Sbasn'})) & (rules['co

rule_confidence
```

Out[64]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representa

no one does this

B

Let's see if we can replicate these findings writing our own formula for calculating confidence:

In [65]:
```
sc_x = df['Sbasn'].sum() # calculate support count of Sbasn
sc_xy = len(df[(df['Sbasn'] == 1) & (df['Beavm'] == 1)]) # calculate support count

confidence = sc_xy/sc_x #calculate confidence of {Sbasn --> Beavm}
confidence
```

Out[65]: 0.0

C

What is the confidence for {Beaver Mountain} → {Snowbasin}? What does this value tell you?

Its still going to be zero.

D

Knowing that Snowbasin is near Ogden and Beaver Mountain is near Logan, what explains the difference in confidence values reported in A and B.

I'm surprised its this low but it could be explained as they are not on the same pass.

E

What is the confidence for {Alta, Snowbird, Snowbasin, Solitude} → {Powder Mountain}? What does this value tell you?

In [69]:
```
sc_x = df['Powmo'].sum() # calculate support count of Sbasn
sc_xy = len(df[(df['Altax'] == 1) & (df['Sbasn'] == 1) & (df['Sbird'] == 1) & (df['

confidence = sc_xy/sc_x
confidence
```

Out[69]: 0.18739713143663297

we have some users doing this combination of trips

F

What is the Lift for {Alta, Snowbird, Snowbasin, Solitude} → {Powder Mountain}? What does this mean?

In [70]:
```
support_y = df['Powmo'].sum() / len(df)

lift = confidence / support_y
lift
```

Out[70]: 0.46344769067728797

visiting alta, bird, basin and tude increases likely hood of visiting powmow

G

Let's see if we can replicate this finding writing our own formula for calculating Lift:

In [71]:
```
sc_y = df['Powmo'].sum() #calculate support count of Powmo
s_y = sc_y/len(df) #calculate support of Powmo
sc_x = len(df[(df['Altax'] == 1) & (df['Sbird'] == 1) & (df['Sbasn'] == 1) & (df['S
sc_xy = len(df[(df['Altax'] == 1) & (df['Sbird'] == 1) & (df['Sbasn'] == 1) & (df['

confidence = sc_xy/sc_x #calculate confidence of x --> y
confidence

Lift = confidence/s_y #calculate lift of x --> y
Lift
```

Out[71]: 2.043831780415684

Q6

Explore the association rules table to find a surprising confidence or lift value in the dataset, then recommend something that could be done about it to the benefit of one or more of the ski resorts. In doing so:

A

Report the surprising value and its association rule. Note, you may also find surprise in a comparison of values (e.g., one confidence is high and another low) — if so, report all relevant values in the comparison.

One surprising rule was: Snowbird to Deervalley with a low confidence of 0.07 and low lift of 0.14.

In contrast, Snowbird to Alta had a confidence of 0.72 and lift of 1.52.

This contrast between two resorts in the same general region is notable.

B

Why do you find it surprising?

Snowbird and Deer Valley are both popular, high-end ski resorts in Utah and are located relatively close to each other. I expected that skiers who visit Snowbird would also consider skiing at Deer Valley, especially if they are tourists trying out multiple premium resorts. However, the data shows this is rarely the case, while visits to Alta (which is also close) occur frequently alongside Snowbird.

C

What would you recommend be done about this (to the benefit of one or more ski resorts?)

This pattern may indicate limited overlap in target demographics or pass access (e.g., Epic vs. Ikon). Deer Valley could benefit from targeted cross-promotion with Snowbird (or similar resorts) — for instance, offering bundled lift ticket deals or transportation services between resorts. This might increase crossover traffic and bring in new skier types who currently overlook Deer Valley.