In [1]:
```
#import packages and clean data before running the market basket analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.frequent_patterns import association_rules, apriori
from mlxtend.preprocessing import TransactionEncoder

#import the csv file that will be used for this market basket analysis
df = pd.read_csv('./medical_market_basket.csv')
#give an example of a transaction from the dataset
df.iloc[3]
```

Out[1]:
```
Presc01                  citalopram
Presc02                     benicar
Presc03    amphetamine salt combo xr
Presc04                         NaN
Presc05                         NaN
Presc06                         NaN
Presc07                         NaN
Presc08                         NaN
Presc09                         NaN
Presc10                         NaN
Presc11                         NaN
Presc12                         NaN
Presc13                         NaN
Presc14                         NaN
Presc15                         NaN
Presc16                         NaN
Presc17                         NaN
Presc18                         NaN
Presc19                         NaN
Presc20                         NaN
Name: 3, dtype: object
```

In [2]:
```
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15002 entries, 0 to 15001
Data columns (total 20 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Presc01  7501 non-null   object
 1   Presc02  5747 non-null   object
 2   Presc03  4389 non-null   object
 3   Presc04  3345 non-null   object
 4   Presc05  2529 non-null   object
 5   Presc06  1864 non-null   object
 6   Presc07  1369 non-null   object
 7   Presc08  981 non-null    object
 8   Presc09  654 non-null    object
 9   Presc10  395 non-null    object
 10  Presc11  256 non-null    object
 11  Presc12  154 non-null    object
 12  Presc13  87 non-null     object
 13  Presc14  47 non-null     object
 14  Presc15  25 non-null     object
 15  Presc16  8 non-null      object
 16  Presc17  4 non-null      object
 17  Presc18  4 non-null      object
 18  Presc19  3 non-null      object
 19  Presc20  1 non-null      object
dtypes: object(20)
memory usage: 2.3+ MB
```

In [3]:
```
#inspect the dataframe to make sure there aren't any issues that could affect our analysis
pd.set_option("display.max_columns", None)
df.head()
```

Out[3]:

	Presc01	Presc02	Presc03	Presc04	Presc05	Presc06	Presc07	Presc08	Presc09	Presc10	Presc11	Presc12
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	amlodipine	albuterol aerosol	allopurinol	pantoprazole	lorazepam	omeprazole	mometasone	fluconozole	gabapentin	pravastatin	cialis	losartan
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	citalopram	benicar	amphetamine salt combo xr	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [4]:

```
#the provided dataset has every row of data separated by a blank row. we will get rid of these rows before proceeding
df = df[df['Presc01'].notna()]
#reset and check the index to make sure we aren't missing any rows
df.reset_index(drop=True, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Data columns (total 20 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Presc01  7501 non-null   object
 1   Presc02  5747 non-null   object
 2   Presc03  4389 non-null   object
 3   Presc04  3345 non-null   object
 4   Presc05  2529 non-null   object
 5   Presc06  1864 non-null   object
 6   Presc07  1369 non-null   object
 7   Presc08  981 non-null    object
 8   Presc09  654 non-null    object
 9   Presc10  395 non-null    object
 10  Presc11  256 non-null    object
 11  Presc12  154 non-null    object
 12  Presc13  87 non-null     object
 13  Presc14  47 non-null     object
 14  Presc15  25 non-null     object
 15  Presc16  8 non-null      object
 16  Presc17  4 non-null      object
 17  Presc18  4 non-null      object
 18  Presc19  3 non-null      object
 19  Presc20  1 non-null      object
dtypes: object(20)
memory usage: 1.1+ MB
```

In [5]:
```
#check to make sure all the empty rows are removed
df.head()
```

Out[5]:

	Presc01	Presc02	Presc03	Presc04	Presc05	Presc06	Presc07	Presc08	Presc09	Presc10	Presc11	Presc
0	amlodipine	albuterol aerosol	allopurinol	pantoprazole	lorazepam	omeprazole	mometasone	fluconozole	gabapentin	pravastatin	cialis	losar
1	citalopram	benicar	amphetamine salt combo xr	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	enalapril	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	paroxetine	allopurinol	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	abilify	atorvastatin	folic acid	naproxen	losartan	NaN	NaN	NaN	NaN	NaN	NaN	N

In [6]:

```
#store the data in a big list of lists
temp_big_list = []
#iterate through each and within each row, iterate through each column
for row_number in range(len(df)):
    # Generate a temporary small list for each row
    temp_small_list = []
    for cell in range(len(df.columns)):
        # check to make sure there are no null values in the cells
        if not pd.isnull(df.iloc[row_number, cell]):
            #if cell contents are not null, add a string version of that cell's contents to the temporary small list
            temp_small_list.append(str(df.values[row_number, cell]))
    #for the list of lists, add the small list to the ongoing big lists
    temp_big_list.append(temp_small_list)
#print the temp_big_list to make sure it looks the way we want it to
print(f"list of lists... \nindex 0: {temp_big_list[0]}\nindex 1: {temp_big_list[1]}\n...\nindex7500: {temp_big_list[750
```

```
list of lists...
index 0: ['amlodipine', 'albuterol aerosol', 'allopurinol', 'pantoprazole', 'lorazepam', 'omeprazole', 'mometasone', 'f
luconozole', 'gabapentin', 'pravastatin', 'cialis', 'losartan', 'metoprolol succinate XL', 'sulfamethoxazole', 'abilif
y', 'spironolactone', 'albuterol HFA', 'levofloxacin', 'promethazine', 'glipizide']
index 1: ['citalopram', 'benicar', 'amphetamine salt combo xr']
...
index7500: ['amphetamine salt combo xr', 'levofloxacin', 'diclofenac sodium', 'cialis']
```

In [7]:

```
#create a transaction encoder
encoder = TransactionEncoder()
#add the transaction encoder to the list of lists, and then change and store the data in a temporary array
temp_array = encoder.fit(temp_big_list).transform(temp_big_list)
#generate a new dataframe from this temporary array
```

```
new_df = pd.DataFrame(temp_array, columns=encoder.columns_)
#check the new dataframe to make sure that it looks the way we want it to
new_df
```

Out[7]:

	Duloxetine	Premarin	Yaz	abilify	acetaminophen	actonel	albuterol HFA	albuterol aerosol	alendronate	allopurinol	alprazolam	amitriptyline
0	False	False	False	True	False	False	True	True	False	True	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	True	False	False
4	False	False	False	True	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
7496	False	False	False	False	False	False	False	False	False	False	False	False
7497	False	False	False	False	False	False	False	False	False	False	False	False
7498	False	False	False	False	False	False	False	False	False	False	False	False
7499	False	False	False	False	False	False	False	False	False	False	True	False
7500	False	False	False	False	False	False	False	False	False	False	False	False

7501 rows × 119 columns

In [8]:

```
#print information about this new dataframe
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Columns: 119 entries, Duloxetine to zolpidem
dtypes: bool(119)
memory usage: 871.8 KB
```

In [9]:

```
#now that the new dataframe has been created, export it as a csv file
new_df.to_csv(r'C:\Users\fahim\Documents\0_WGUDocuments\d212\task3_marketbasket_clean.csv', index=False)
```

In [10]:
```
#using the Apriori algorithm, generate frequent itemsets
frequent_itemsets = apriori(new_df, min_support = 0.02, use_colnames = True)
frequent_itemsets
```

Out[10]:

	support	itemsets
0	0.046794	(Premarin)
1	0.238368	(abilify)
2	0.020397	(albuterol aerosol)
3	0.033329	(allopurinol)
4	0.079323	(alprazolam)
...	...	...
98	0.023064	(lisinopril, diazepam)
99	0.023464	(diazepam, losartan)
100	0.022930	(metoprolol, diazepam)
101	0.020131	(doxycycline hyclate, glyburide)
102	0.028530	(losartan, glyburide)

103 rows × 2 columns

In [11]:
```
# we will now use use association_rules with a lift of greater than 1
rules = association_rules(frequent_itemsets, metric = 'lift', min_threshold = 1.0)
rules
```

Out[11]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(abilify)	(amlodipine)	0.238368	0.071457	0.023597	0.098993	1.385352	0.006564	1.030562	0.365218
1	(amlodipine)	(abilify)	0.071457	0.238368	0.023597	0.330224	1.385352	0.006564	1.137144	0.299568
2	(abilify)	(amphetamine salt combo)	0.238368	0.068391	0.024397	0.102349	1.496530	0.008095	1.037830	0.435627
3	(amphetamine salt combo)	(abilify)	0.068391	0.238368	0.024397	0.356725	1.496530	0.008095	1.183991	0.356144
4	(amphetamine salt combo xr)	(abilify)	0.179709	0.238368	0.050927	0.283383	1.188845	0.008090	1.062815	0.193648
...	...	...	...	...	...	...	...	...	...	...
89	(diazepam)	(metoprolol)	0.163845	0.095321	0.022930	0.139951	1.468215	0.007312	1.051893	0.381390
90	(doxycycline hyclate)	(glyburide)	0.095054	0.170911	0.020131	0.211781	1.239135	0.003885	1.051852	0.213256
91	(glyburide)	(doxycycline hyclate)	0.170911	0.095054	0.020131	0.117785	1.239135	0.003885	1.025766	0.232768
92	(losartan)	(glyburide)	0.132116	0.170911	0.028530	0.215943	1.263488	0.005950	1.057436	0.240286
93	(glyburide)	(losartan)	0.170911	0.132116	0.028530	0.166927	1.263488	0.005950	1.041786	0.251529

94 rows × 10 columns

In [12]:

```
#provide the association rules table, and showcase the scores for support, confidence, and lift.
rules
```

Out[12]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(abilify)	(amlodipine)	0.238368	0.071457	0.023597	0.098993	1.385352	0.006564	1.030562	0.365218
1	(amlodipine)	(abilify)	0.071457	0.238368	0.023597	0.330224	1.385352	0.006564	1.137144	0.299568
2	(abilify)	(amphetamine salt combo)	0.238368	0.068391	0.024397	0.102349	1.496530	0.008095	1.037830	0.435627
3	(amphetamine salt combo)	(abilify)	0.068391	0.238368	0.024397	0.356725	1.496530	0.008095	1.183991	0.356144
4	(amphetamine salt combo xr)	(abilify)	0.179709	0.238368	0.050927	0.283383	1.188845	0.008090	1.062815	0.193648
...	...	...	...	...	...	...	...	...	...	...
89	(diazepam)	(metoprolol)	0.163845	0.095321	0.022930	0.139951	1.468215	0.007312	1.051893	0.381390
90	(doxycycline hyclate)	(glyburide)	0.095054	0.170911	0.020131	0.211781	1.239135	0.003885	1.051852	0.213256
91	(glyburide)	(doxycycline hyclate)	0.170911	0.095054	0.020131	0.117785	1.239135	0.003885	1.025766	0.232768
92	(losartan)	(glyburide)	0.132116	0.170911	0.028530	0.215943	1.263488	0.005950	1.057436	0.240286
93	(glyburide)	(losartan)	0.170911	0.132116	0.028530	0.166927	1.263488	0.005950	1.041786	0.251529

94 rows × 10 columns

In [13]:

```
#showcase the top 3 rules of the associated rules table, having a lift of over 1.9 and confidence of 0.3
top_3_rules = rules[(rules['lift'] > 1.9) & (rules['confidence'] > 0.3)].sort_values(by=['lift'], ascending= False)
top_3_rules
```

Out[13]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
75	(lisinopril)	(carvedilol)	0.098254	0.174110	0.039195	0.398915	2.291162	0.022088	1.373997	0.624943
72	(glipizide)	(carvedilol)	0.065858	0.174110	0.022930	0.348178	1.999758	0.011464	1.267048	0.535186
31	(metformin)	(abilify)	0.050527	0.238368	0.023064	0.456464	1.914955	0.011020	1.401255	0.503221

In [14]:
```
#check the value counts for antecedents values
rules.antecedents.value_counts()
```

Out[14]:
```
(abilify)                     18
(carvedilol)                  12
(amphetamine salt combo xr)    9
(diazepam)                     8
(atorvastatin)                 7
(glyburide)                    6
(metoprolol)                   5
(doxycycline hyclate)          4
(lisinopril)                   4
(losartan)                     4
(citalopram)                   4
(amlodipine)                   2
(amphetamine salt combo)       2
(glipizide)                    2
(dextroamphetamine XR)         1
(levofloxacin)                 1
(clopidogrel)                  1
(metformin)                    1
(naproxen)                     1
(cialis)                       1
(fenofibrate)                  1
Name: antecedents, dtype: int64
```

In [15]:
```
#check the value counts for consequent values
rules.consequents.value_counts()
```

Out[15]:
```
(abilify)                     18
(carvedilol)                  12
(amphetamine salt combo xr)    9
(diazepam)                     8
(atorvastatin)                 7
(glyburide)                    6
(metoprolol)                   5
(lisinopril)                   4
(doxycycline hyclate)          4
(losartan)                     4
(citalopram)                   4
(glipizide)                    2
(amphetamine salt combo)       2
(amlodipine)                   2
(dextroamphetamine XR)         1
(clopidogrel)                  1
(fenofibrate)                  1
(levofloxacin)                 1
(metformin)                    1
(cialis)                       1
(naproxen)                     1
Name: consequents, dtype: int64
```

In [16]:
```
#now that all the rules for the dataframe have been set, print out the antecedent and consequents rules for our target
ant_df = rules[rules['antecedents'] == {'cialis'}]
con_df = rules[rules['consequents'] == {'cialis'}]
cialis_df = pd.concat([ant_df, con_df])
cialis_df
```

Out[16]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
11	(cialis)	(abilify)	0.076523	0.238368	0.023997	0.313589	1.315565	0.005756	1.109585	0.259747
10	(abilify)	(cialis)	0.238368	0.076523	0.023997	0.100671	1.315565	0.005756	1.026851	0.314943

In [ ]: