import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import numpy as np

import scipy.stats as stats

import datetime

df1=pd.read\_csv('Cereals.csv')

1. iloc[row:row,col:col]
2. df.describe() //5 number summary
3. typevsmfr=pd.crosstab(df1.type, df1.mfr)
4. print("The creal with best rating: ")

maxm=df1['rating'].idxmax()

print(df1.iat[maxm,0])

print(df1['rating'].max())

1. correlation= df1['calories'].corr(df1['rating'])
2. mfr1=df1[df1['mfr']=='A']
3. mfr1['rating'].mean()
4. rating=pd.DataFrame({'mfr':['A','G','K','N','P','Q','R'],

'Rating':[avg1,avg2,avg3,avg4,avg5,avg6,avg7]})

1. df.skew()
2. df.kurt())
3. df.shape[0] //0 no of rows(no of records)
4. df['Genre'].str.strip() //trim
5. gen.value\_counts(sort=True, ascending=False)
6. df['month']=pd.DatetimeIndex(df['Release Date']).month
7. df['ROI']=(df['BoxOfficeCollection']-df['Budget'])/df['Budget']
8. sum1=gen1['YoutubeLikes'].sum()
9. yr1= year1['SlNo'].nunique()
10. match\_detail=df[['id', 'season']].merge(DF, left\_on='id', right\_on='match\_id', how='left')
11. match\_detail.groupby(['season'])['total\_runs'].sum().reset\_index()
12. top10.sort\_values(by=['total\_runs'], ascending=False).head(10)
13. total = totalbat.append(totalbowl, ignore\_index=True)

totalt= total.value\_counts()

team\_stats=pd.DataFrame({'TotalMatches': df.team1.value\_counts()+ df.team2.value\_counts()})

1. team\_stats=team\_stats.reset\_index()
2. team\_stats.rename(columns={'index': 'Teams'}, inplace=True)
3. #Finding unique CustomerIDs and storing them in an array

CustomerID = OR3['CustomerID'].unique()

CustomerID

RFM = pd.DataFrame(columns = ['CustomerID','Recency', 'Frequency','Monetary Value'])

RFM['CustomerID'] = CustomerID

RFM

RFM.sort\_values(by ='CustomerID',inplace = True)

for CID in CustomerID:

RFM.loc[RFM.CustomerID==CID, 'Frequency'] = OR3[OR3['CustomerID']==CID]['InvoiceNo'].value\_counts().sum()

RFM.loc[RFM.CustomerID==CID, 'Recency'] = (12-OR3.loc[OR3.CustomerID==CID,'InvoiceDate'].max().date().month)

InvoiceArr=OR3.loc[OR3.CustomerID==CID,'InvoiceNo'].unique()

sum=0

for INo in InvoiceArr:

sum+=((OR3.loc[OR3.InvoiceNo==INo,'UnitPrice']\*OR3.loc[OR3.InvoiceNo==INo,'Quantity']).sum())

RFM.loc[RFM.CustomerID==CID, 'Monetary Value'] = sum

RFM

RFMc = pd.DataFrame(columns=['Recency', 'Frequency', 'Monetary Value'])

RFMc['Recency'] = RFM['Recency'].dropna()

RFMc['Frequency'] = RFM['Frequency'].dropna()

RFMc['Monetary Value'] = RFM['Monetary Value'].dropna()

RFMc

VISUALIZATION

1. sns.boxplot(x='rating' ,y='type', data=df1)

plt.title("side-by-side boxplot comparing the consumer rating of hot vs. cold cereals.")

plt.show()

1. sns.pairplot(df[['sugars','calories','carbo','fat']])

plt.show()

1. plt.plot(rating['mfr'], rating['Rating'])

plt.title("relation between manufacturer and rating")

plt.show()

//line plot. //cat vs num

1. plt.hist(df['mpg'])

plt.title("miles per gallon")

plt.xlabel("Miles")

plt.ylabel("Gallons")

plt.show()

//distribution.

1. sns.heatmap(df.corr())

plt.show(). //correlation

1. df3[['Genre', 'Budget']].plot(kind='bar')

plt.show //distribution. // categories not printed only index

1. ax = plt.axes()

ax.set(facecolor = "white")

sns.countplot(x='season', hue='toss\_decision', data=df,palette="gnuplot2",saturation=1)

plt.xlabel('\n Season',fontsize=15)

plt.ylabel('Count',fontsize=15)

plt.title('Toss decision in different seasons',fontsize=15,fontweight="bold")

plt.show()

// Visualize the Toss decision across seasons

1. sns.barplot(x = top\_players.index, y = top\_players, orient='v'); #palette="Blues");

plt.show()

1. sns.lineplot(data=df.iloc[:,3:6])

sns.set(rc={"figure.figsize":(15,15)})

sns.set(font\_scale=2)

plt.show()

//vertical lines one over another //distribution

1. sns.lineplot(data=df.iloc[:,2:4])

sns.set(rc={"figure.figsize":(15,15)})

sns.set(font\_scale=2)

plt.show()

//horizontal lines one over another //distribution

1. #Scatterplot: Frequency vs Recency

facet = sns.lmplot(data=RFMc, x='Recency', y='Frequency', hue='Cluster',

fit\_reg=False, legend=True, legend\_out=True)

plt.legend(loc='right', labels=['Occasional Customers(0)', 'Routine Customers(1)' ,'VIP Customers(2)'])

1. #Visualising clusters usind dendrogram

Dendrogram = shc.dendrogram((shc.linkage(Xac, method ='ward')))

DATA CLEANING

1. df.replace(item, replace) //data cleaning
2. np.nan //fill up nan values
3. df['potass']=pd.to\_numeric(df['potass'], errors='coerce') //if 'coerce', then invalid parsing will be set as NaN
4. df=df.mask(df==-1) //The mask() method replaces the values of the rows where the condition evaluates to True
5. df= df.fillna(df.mean())
6. q1=df.quantile( 0.25)

iqr=q3-q1

low\_lim = q1 - 1.5 \* iqr

df=df[df>low\_lim]

df= df.fillna(df.median())

1. df=df.dropna() //dropping empty fields
2. # 4. Clean the data and remove the special characters and replace the contractions with its expansion by converting the uppercase character to lower case. Also, remove the punctuations.

Apos\_dict={"'s":" is","n't":" not","'m":" am","'ll":" will",

"'d":" would","'ve":" have","'re":" are"}

#replace the contractions

for key,value in Apos\_dict.items():

if key in df['review']:

df['review']=df['review'].replace(key,value)

1. # removing non alpha-numeric characters

df['review'] = df['review'].str.replace('[^a-zA-Z0-9]', ' ', regex=True).str.strip()

1. #converting to lower case

df['review'].str.lower()

1. # 5. Add the Polarity, length of the review, the word count and average word length of each review

df = df[df['rating'] != 3]

def Polarity(n):

return 1 if n >= 4 else 0

df['Polarity'] = df['rating'].apply(Polarity)

df['Length\_of\_review'] = df['review'].apply(len)

df['Word\_count'] = df['review'].str.count(' ')+1

df['avg\_word\_length'] = df['Length\_of\_review']/ df['Word\_count']

1. OR3 = OR2.loc[OR2.Quantity>0]

AMAZON BABY

import pandas as pd

from matplotlib import pyplot as plt

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# from html.parser import HTMLParser

import re

APRIORI

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

import seaborn as sns

from csv import reader

from mlxtend.plotting import plot\_decision\_regions

import mlxtend

from mlxtend.frequent\_patterns import apriori

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import association\_rules

1. a= []

with open('groceries.csv', 'r') as read\_obj:

csv\_reader = reader(read\_obj)

for row in csv\_reader:

a.append(row)

1. b= TransactionEncoder()

trans= b.fit(groceries).transform(a)

trans

1. trans= transactions.astype('int')

trans

1. df = pd.DataFrame(trans, columns=encoder.columns\_)
2. freq= apriori(df, min\_support=0.02, use\_colnames=True)

freq['length'] = freq['itemsets'].apply(lambda x: len(x))

freq

1. print("TOP 10 SELLING ITEMS ON THE BASIS OF SUPPORT(2%): ")

freq = freq.sort\_values(by='support', ascending=False)

freq[ (freq['length'] == 1) & (freq['support'] >= 0.02) ][0:10]

1. x= association\_rules(freq, metric='support', min\_threshold=0.02)

print("DATAFRAME ACC.TO CONFIDENCE LEVEL IS AS FOLLOWS: ")

x.sort\_values(by='confidence', ascending=False)[0:10]

1. print("DATAFRAME ACC.TO LIFT CONDITION IS: ")

x[(x['support'] >= 0.02) & (x['lift'] > 1.0)]

CORRELATION

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import scipy.stats as stats

import datetime

from sklearn.metrics import pairwise\_distances as pair

from scipy.spatial.distance import cosine, correlation

1. #2. Read the “ratings.csv” file and create a pivot table with index=‘userId’, columns=‘movieId’, values = “rating".

dff = df3.pivot\_table(index='userId', columns='movieId', values='rating',fill\_value=0, aggfunc=np.mean)

1. #Use cosine similarity for finding similarity among users. Use the following packages

pair=pairwise\_distances(dff, Y=None, metric='cosine')

1. #6. Find the 5 most similar user for user with user Id 25.

print((-pair[24]).argsort()[1:6]+1)

1. #7. Use the “movies” dataset to find out the names of movies, user 1 and user 338 have watched in common and how they have rated each one of them.

common\_movies = set(df3.loc[df3.userId==1, 'movieId']).intersection(set(df3.loc[df3.userId==338, 'movieId']))

1. #8. Use the movies dataset to find out the common movie names between user 2 and user 338 with least rating of 4.0

common\_movies2 = set(df3.loc[((df3.userId==2) & (df3.rating>=4.0)), 'movieId']).intersection(set(df3.loc[((df3.userId==338) & (df3.rating>=4.0)), 'movieId']))

common\_movies2

for movieid in sorted(common\_movies2):

common = df2[df2['movieId'] == movieid]

print(common)

1. #9. Create a pivot table for representing the similarity among movies using correlation.

dff = pd.pivot\_table(df3, index='movieId', columns='userId', values='rating', fill\_value=0)

pair2=pairwise\_distances(dff, Y=None, metric='correlation')

1. #10.Find the top 5 movies which are similar to the movie “Godfather”.

for movie in df2.title:

if('Godfather' in str(movie)):

print(movie)

df2.loc[(df2.title=='Godfather, The (1972)'), 'movieId'].values[0]

similar\_movies = pair2[857].argsort()[1:6]+1

for movieID in similar\_movies:

print(df2[df2["movieId"] == movieID])

print("\n")

CLUSTERING

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.cluster import AgglomerativeClustering

%matplotlib inline

from yellowbrick.cluster import KElbowVisualizer

from sklearn.preprocessing import StandardScaler

import scipy.cluster.hierarchy as shc

1. X = StandardScaler().fit\_transform(RFMc)
2. model = KMeans()

visualizer = KElbowVisualizer(model,k=(1,9))

visualizer.fit(X)

visualizer.show()

1. k\_means = KMeans(n\_clusters=3)

model = k\_means.fit(X)

model

1. RFMc['Cluster'] = k\_means.predict(X)
2. RFMc0 = RFMc[RFMc['Cluster']==0]

for col in ['Recency', 'Frequency', 'Monetary Value']:

print(col)

print("Min: ", RFMc0.loc[:,col].min())

print("Median: ", RFMc0.loc[:,col].median())

print("Mean: ", RFMc0.loc[:,col].mean())

print("Max: ", RFMc0.loc[:,col].max())

print("")

print("Total Monetary Value:", RFMc0['Monetary Value'].sum())

1. Xac = StandardScaler().fit\_transform(RFMac)
2. ac = AgglomerativeClustering(n\_clusters = 3)

RFMac['Cluster'] = ac.fit\_predict(Xac)

In these 3 scatterplots, it is seen that the cluster boundaries are much more clearer. They do not mix up with other clusters and have greater intra-cluster similarity Hence Agglomerative Clustering yeilds better cluster predictions in this case