# Part 5 – Clustering

## K Means Clustering

## Data Prep

import pandas as pd

df = pd.read\_csv('Data/entertainment\_clean.csv')

df.head()

import warnings

warnings.filterwarnings('ignore')

### Data Checks

#### 1. Row Granularity

#### 2. Columns Non-null

#### 3. Columns Numeric

#### 4. Feature Engineering

#### 5. Feature Selection

#### 6. Scaling

df.shape

df.name.nunique()

df[df.isna().any(axis=1)]

df.dtypes

df.head()

data = df.drop(columns='name')

data.head()

data.describe()

## K Means Clustering

from sklearn.cluster import KMeans

kmeans2 = KMeans(n\_clusters=3, n\_init='auto', random\_state=42)

kmeans2.fit(data)

kmeans2.labels\_

## Visualize Clusters

# required libraries

import matplotlib.pyplot as plt

import seaborn as sns

import mpl\_toolkits

from mpl\_toolkits.mplot3d import Axes3D

# combine data and cluster labels

cluster\_labels = pd.Series(kmeans2.labels\_, name='cluster')

# create a clean dataframe

df2 = pd.concat([data, cluster\_labels], axis=1)

# create a 3D scatter plot

fig = plt.figure(figsize=(8, 6), dpi=150)

ax = Axes3D(fig)

fig.add\_axes(ax)

# specify the data and labels

sc = ax.scatter(df2['books'], df2['tv\_shows'], df2['video\_games'], c=df2['cluster'], cmap='tab10')

ax.set\_xlabel('books')

ax.set\_ylabel('tv\_shows')

ax.set\_zlabel('video\_gmaes')

# add a legend

plt.legend(\*sc.legend\_elements(), title='clusters', bbox\_to\_anchor=(1.05, 1));

## Interpreting K Means Clusters

df2.columns

kmeans2.cluster\_centers\_

data.mean()

# cluster 0: non-readers

# cluster 1: entertainment enthusiasts

### Visualizing Cluster Centers

kmeans2.cluster\_centers\_

cluster\_centers\_3 = pd.DataFrame(kmeans2.cluster\_centers\_, columns=data.columns)

cluster\_centers\_3

my\_labels = pd.DataFrame(kmeans2.labels\_)

new\_data = pd.concat([data, my\_labels], axis=1)

new\_data

plt.figure(figsize=(8, 6), dpi=150)

import seaborn as sns

sns.heatmap(cluster\_centers\_3, cmap='RdBu', annot=True) ;

## Inertia Plot

kmeans2.inertia\_

# goal: kmeans 2 to 15 clusters

inertia\_values = []

for k in range(2, 16):

kmeans = KMeans(n\_clusters=k, n\_init='auto', random\_state=42)

kmeans.fit(data)

inertia\_values.append(kmeans.inertia\_)

import matplotlib.pyplot as plt

inertia\_series = pd.Series(inertia\_values, index=range(2, 16))

inertia\_series.plot(marker='o');

## Elbow Models

kmeans5 = KMeans(n\_clusters=5, n\_init='auto', random\_state=42)

kmeans5.fit(data)

kmeans5.labels\_

kmeans5.cluster\_centers\_

cluster\_centers5 = pd.DataFrame(kmeans5.cluster\_centers\_, columns=data.columns)

cluster\_centers5

plt.figure(figsize=(8, 6), dpi=150)

sns.heatmap(cluster\_centers5, cmap='RdBu', annot=True) ;

## Hierarchical Clustering

The

# New Stuff

# Clustering Project

# 1. Data Prep

# Import Required Libraires

import pandas as pd

import numpy as np

import warnings

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from collections import Counter

import seaborn as sns

from sklearn.metrics import silhouette\_score

from sklearn.cluster import AgglomerativeClustering

from sklearn.cluster import DBSCAN

warnings.filterwarnings('ignore')

# a. Read in the data file wholesale\_clients.csv

base\_data = pd.read\_csv('Data/wholesale\_clients.csv')

base\_data.head()

# b. Remove the Channel (restaurant, hotel, etc.) and Region columns since they are not fields we want to model on

base\_data = base\_data.drop('Channel', axis=1)

base\_data = base\_data.drop('Region', axis=1)

base\_data.head()

# c. Note the number of rows and columns

(440, 6)

base\_data[base\_data.isna().any(axis=1)]

base\_data.dtypes

# d. Standardize the data

scaler = StandardScaler()

data\_scaled = pd.DataFrame(scaler.fit\_transform(base\_data), columns=base\_data.columns)

data\_scaled.head()

# e. Double check that all the column means are 0 and standard deviations are 1

data\_scaled.describe()

# 2. K-Means Clustering

# a. Import KMeans and write a loop to fit models with 2 to 15 clusters

# create an empty list to hold many inertia values

inertia\_values = []

# create 2 - 15 clusters, and add the inertia scores to the list

for k in range(2, 16):

kmeans = KMeans(n\_clusters=k, n\_init=10)

kmeans.fit(data\_scaled)

inertia\_values.append(kmeans.inertia\_)

[1954.1835647259281,

1608.4311488289445,

1317.8383605390707,

1058.7712532570083,

919.2995086131316,

828.4145695100656,

745.964051497191,

659.722292868476,

598.9518184208159,

551.0732359492046,

522.6558624394585,

492.3748349356686,

462.15139809764094,

433.3354951588281]

# b. Create an inertia plot

# turn the list into a series for plotting

inertia\_series = pd.Series(inertia\_values, index=range(2, 16))

# plot the data

inertia\_series.plot(marker='o')

plt.xlabel("Number of Clusters (k)")

plt.ylabel("Inertia")

plt.title("Number of Clusters vs. Inertia");

# c. Identify the elbow of the plot and fit a KMeans model just for that number of clusters

# Elbow point is at 5

kmeans5 = KMeans(n\_clusters=5, n\_init='auto', random\_state=42)

kmeans5.fit(data\_scaled)

kmeans5.labels\_

array([3, 0, 0, 3, 1, 3, 3, 3, 3, 0, 0, 3, 1, 0, 0, 3, 0, 3, 3, 3, 3, 3,

1, 2, 0, 3, 3, 3, 0, 1, 3, 3, 3, 1, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0,

0, 0, 0, 2, 0, 0, 3, 3, 1, 0, 1, 3, 2, 0, 3, 3, 3, 2, 3, 0, 3, 2,

3, 0, 3, 3, 1, 1, 3, 1, 3, 3, 3, 0, 3, 3, 3, 0, 0, 3, 3, 2, 2, 1,

3, 1, 3, 3, 2, 1, 0, 3, 3, 3, 3, 3, 0, 0, 3, 1, 3, 3, 0, 0, 3, 0,

3, 0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 0, 3, 3, 3, 1, 3, 3, 3, 3,

3, 0, 0, 3, 0, 0, 0, 3, 3, 0, 3, 0, 0, 3, 3, 3, 0, 0, 3, 0, 3, 0,

1, 3, 3, 3, 3, 1, 0, 4, 3, 3, 3, 3, 0, 0, 3, 3, 3, 0, 3, 3, 1, 0,

3, 3, 0, 0, 1, 3, 3, 0, 3, 3, 3, 0, 3, 2, 3, 3, 0, 0, 0, 3, 0, 3,

3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3,

3, 3, 0, 0, 3, 3, 3, 3, 3, 2, 3, 1, 0, 1, 3, 3, 1, 1, 3, 3, 3, 3,

0, 0, 0, 3, 0, 3, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1,

3, 3, 3, 1, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 3,

3, 0, 3, 1, 0, 3, 3, 0, 3, 3, 3, 0, 3, 3, 3, 3, 3, 4, 3, 3, 3, 3,

3, 0, 3, 2, 1, 1, 3, 3, 3, 3, 0, 0, 3, 0, 3, 3, 0, 1, 3, 0, 3, 0,

3, 0, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3,

3, 3, 0, 1, 3, 3, 1, 1, 1, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3,

0, 3, 3, 3, 3, 1, 3, 3, 3, 3, 1, 0, 3, 3, 3, 3, 3, 1, 3, 3, 0, 3,

0, 3, 0, 3, 3, 3, 3, 1, 0, 1, 3, 3, 3, 3, 3, 3, 3, 1, 1, 0, 3, 3])

data\_scaled.columns

Index(['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen'], dtype='object')

# d. Find the number of clients in each cluster

Counter(kmeans5.labels\_)

Counter({3: 272, 0: 96, 1: 59, 2: 11, 4: 2})

cluster\_centers\_3 = pd.DataFrame(kmeans5.cluster\_centers\_, columns=data\_scaled.columns)

cluster\_centers\_3

# e. Create a heat map of the cluster centers

plt.figure(figsize=(8, 6), dpi=150)

sns.heatmap(cluster\_centers\_3, cmap='RdBu', annot=True) ;

### f. Name the clusters

- Cluster 0: More Milk, Grocery and Paper Items

- Cluster 1: More Fresh and Frozen Items

- Cluster 2: More of Everything

- Cluster 3: Little of Everything

- Cluster 4: Lots of Frozen + Deli Items

k5\_labels = pd.DataFrame(kmeans5.labels\_)

k5\_labels.head()

base\_data.head()

new\_base\_data = pd.concat([base\_data, k5\_labels], axis=1)

new\_base\_data.head()

new\_base\_data.sort\_values(0)

# g. Extra credit: create a silhouette scores plot instead of an inertia plot

# create an empty list to hold many silhouette score values

silhouette\_scores = []

# create 2 - 15 clusters, and add the silhouette scores to the list

for k in range(2, 16):

kmeans = KMeans(n\_clusters=k, n\_init=100)

kmeans.fit(data\_scaled)

silhouette\_scores.append(silhouette\_score(data\_scaled, kmeans.labels\_, metric='euclidean', sample\_size=None))

print(silhouette\_scores)

for score in silhouette\_scores:

print(score)

# h. Extra credit: fit two models with the number of clusters for the two highest silhouette scores and name the clusters

# plot the silhouette scores

# turn the list into a series for plotting

silhouette\_series = pd.Series(silhouette\_scores, index=range(2, 16))

# plot the data

silhouette\_series.plot(marker='o')

plt.xlabel("Number of Clusters (k)")

plt.ylabel("Silhouette Score")

plt.title("Number of Clusters vs. Silhouette Score");

# h. Extra credit: fit two models with the number of clusters for the two highest silhouette scores and name the clusters

kmeans2 = KMeans(n\_clusters=2, n\_init='auto', random\_state=42)

kmeans2.fit(data\_scaled)

Counter(kmeans2.labels\_)

Counter({0: 373, 1: 67})

kmeans3 = KMeans(n\_clusters=3, n\_init='auto', random\_state=42)

kmeans3.fit(data\_scaled)

Counter(kmeans3.labels\_)

Counter({0: 350, 1: 53, 2: 37})

plt.figure(figsize=(8, 6), dpi=150)

cluster\_centers2 = pd.DataFrame(kmeans2.cluster\_centers\_, columns=data\_scaled.columns)

sns.heatmap(cluster\_centers2, cmap='RdBu', annot=True);

plt.figure(figsize=(8, 6), dpi=150)

cluster\_centers3 = pd.DataFrame(kmeans3.cluster\_centers\_, columns=data\_scaled.columns)

sns.heatmap(cluster\_centers3, cmap='RdBu', annot=True);

data\_scaled.head()

## 3. Hierarchical Clustering

# a. Create a dendrogram using the scaled data

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

linkage\_matrix = linkage(data\_scaled, method='ward')

dendrogram\_info = dendrogram(linkage\_matrix, color\_threshold=20)

plt.title("Hierarchical Clustering Dendrogram")

plt.xlabel("Data Points")

plt.ylabel("Euclidean Distance");

# b. Visually identify the number of clusters and update the color threadshold, if necessary

# c. Fit an agglomerative clustering model on the scaled data set with the "best" clusters and view the number of data points in each cluster

agg5 = AgglomerativeClustering(5)

agg5.fit(data\_scaled)

Counter(agg5.labels\_)

Counter({4: 177, 0: 153, 2: 104, 1: 5, 3: 1})

agg3 = AgglomerativeClustering(3)

agg3.fit(data\_scaled)

Counter(agg3.labels\_)

Counter({1: 281, 2: 153, 0: 6})

agg2 = AgglomerativeClustering(2)

agg2.fit(data\_scaled)

Counter(agg2.labels\_)

Counter({0: 434, 1: 6})

# d. Create a cluster map of the model you just fit

sns.clustermap(data\_scaled, method='ward', cmap='RdBu', figsize=(8, 6), xticklabels=data\_scaled.columns)

plt.show()

# e. Extra credit: within the clustermap function, add z\_score=0 (scales data by row), see what happens and interpret the clusters

sns.clustermap(data\_scaled, method='ward', cmap='RdBu', figsize=(8, 6), xticklabels=data\_scaled.columns, z\_score=0)

plt.show()

agg\_silhouette\_scores = {}

for n in range(2, 21):

agg\_n = AgglomerativeClustering(n)

agg\_n.fit(data\_scaled)

agg\_ss = silhouette\_score(data\_scaled, agg\_n.labels\_, metric='euclidean', sample\_size=None)

agg\_silhouette\_scores[n] = agg\_ss

agg\_silhouette\_scores

{2: 0.7924572758342273,

3: 0.2646091480214908,

4: 0.2670148241989557,

5: 0.23988371669170097,

6: 0.23424990872738188,

7: 0.24405686464597856,

8: 0.2520384687343341,

9: 0.27202620570114383,

10: 0.27954261064916225,

11: 0.28015398902140587,

12: 0.24562381160303043,

13: 0.24647695283235932,

14: 0.22903049767293357,

15: 0.2308407937114372,

16: 0.18896719179943278,

17: 0.1959949281863608,

18: 0.1954950889983609,

19: 0.20219343087967662,

20: 0.20160566549699505}

# udate scaled dataframe so that each row has a mean 0 and standard deviation of 1

# calculate mean and standard deviation for erach row

row\_means = data\_scaled.mean(axis=1)

row\_stds = data\_scaled.std(axis=1)

# divide each element in the row by standard deviastion

data\_zscore = data\_scaled.sub(row\_means, axis=0).div(row\_stds, axis=0)

data\_zscore.head()

# update the dendrogram

linkage\_matrix\_zscore = linkage(data\_zscore, method='ward')

dendrogram\_info\_zscore = dendrogram(linkage\_matrix\_zscore, color\_threshold=15)

plt.title("Hierarchical Clustering Dendrogram")

plt.xlabel("Data Points")

plt.ylabel("Euclidean Distance");

# check if cluster mapdata points match the denfrogram

data\_points = pd.Series(dendrogram\_info['ivl'], name='Data Point').astype('int')

data\_points.head()

0 183

1 86

2 47

3 61

4 85

Name: Data Point, dtype: int32

# check if cluster mapdata points match the denfrogram

data\_points\_zscore = pd.Series(dendrogram\_info\_zscore['ivl'], name='Data Point').astype('int')

data\_points\_zscore.head()

0 95

1 317

2 430

3 184

4 266

Name: Data Point, dtype: int32

# f. Extra credit: write a loop to view the silhouette score for 2 to 20 clusters

agg\_silhouette\_scores = {}

for n in range(2, 21):

agg\_n = AgglomerativeClustering(n)

agg\_n.fit(data\_scaled)

agg\_ss = silhouette\_score(data\_scaled, agg\_n.labels\_, metric='euclidean', sample\_size=None)

agg\_silhouette\_scores[n] = agg\_ss

agg\_silhouette\_scores

{2: 0.7924572758342273,

3: 0.2646091480214908,

4: 0.2670148241989557,

5: 0.23988371669170097,

6: 0.23424990872738188,

7: 0.24405686464597856,

8: 0.2520384687343341,

9: 0.27202620570114383,

10: 0.27954261064916225,

11: 0.28015398902140587,

12: 0.24562381160303043,

13: 0.24647695283235932,

14: 0.22903049767293357,

15: 0.2308407937114372,

16: 0.18896719179943278,

17: 0.1959949281863608,

18: 0.1954950889983609,

19: 0.20219343087967662,

20: 0.20160566549699505}

# cluster map 1: data scaled and z\_score = 0

sns.clustermap(data\_scaled, method='ward', cmap='RdBu', figsize=(8, 6), xticklabels=data\_scaled.columns, z\_score=0)

plt.show()

# cluster map 2: data\_zscore without z\_score parameter

sns.clustermap(data\_zscore, method='ward', cmap='RdBu', figsize=(8, 6), xticklabels=data\_scaled.columns)

plt.show()

# update the agg model with the cluster number and new data set

agg4\_zscore = AgglomerativeClustering(4)

agg4\_zscore.fit(data\_zscore)

Counter(agg4\_zscore.labels\_)

Counter({1: 149, 0: 125, 2: 110, 3: 56})

# f. Extra credit: Write a loop to view the silhouette score for 2 to 20 clusters

agg\_silhouette\_scores = {}

for n in range(2, 21):

agg\_n = AgglomerativeClustering(n)

agg\_n.fit(data\_scaled)

agg\_ss = silhouette\_score(data\_scaled, agg\_n.labels\_, metric='euclidean', sample\_size=None)

agg\_silhouette\_scores[n] = agg\_ss

agg\_silhouette\_scores

{2: 0.7924572758342273,

3: 0.2646091480214908,

4: 0.2670148241989557,

5: 0.23988371669170097,

6: 0.23424990872738188,

7: 0.24405686464597856,

8: 0.2520384687343341,

9: 0.27202620570114383,

10: 0.27954261064916225,

11: 0.28015398902140587,

12: 0.24562381160303043,

13: 0.24647695283235932,

14: 0.22903049767293357,

15: 0.2308407937114372,

16: 0.18896719179943278,

17: 0.1959949281863608,

18: 0.1954950889983609,

19: 0.20219343087967662,

20: 0.20160566549699505}

# g. Extra credit: Fit a model with the number of clusters for the highest silhouette score

agg2 = AgglomerativeClustering(2)

agg2.fit(data\_scaled)

Counter(agg2.labels\_)

Counter({0: 434, 1: 6})

# 4. DBSCAN

# a. Copy over the tune\_dbscan function from the demo code

import numpy as np

from sklearn.cluster import DBSCAN

from sklearn.metrics import silhouette\_score

def tune\_dbscan(data):

results = []

# define a range of eps and min\_samples values to loop through

eps\_values = np.arange(.1, 2, .1)

min\_samples\_values = np.arange(2, 10, 1)

# loop through the combinations of eps and min\_samples

for eps in eps\_values:

for min\_samples in min\_samples\_values:

dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)

dbscan.fit(data)

labels = dbscan.labels\_

# count the number of clusters (excluding noise points labeled as -1)

n\_clusters = len(set(labels)) - (1 if -1 in labels else 0)

# count the number of noise points (labeled as -1)

n\_noise = list(labels).count(-1)

# calculate the silhouette score (excluding noise points)

if n\_clusters > 1: # silhouette score requires at least 2 clusters

silhouette = silhouette\_score(data, labels, metric='euclidean', sample\_size=None)

else:

silhouette = None

results.append([eps, min\_samples, n\_clusters, n\_noise, silhouette])

# put the results in a dataframe

dbscan\_results = pd.DataFrame(results, columns=["Eps", "Min Samples", "Number of Clusters",

"Number of Noise Points", "Silhouette Score"])

return dbscan\_results

# b. Apply the dbscan function on the scaled data

dbscan\_results = tune\_dbscan(data\_scaled)

dbscan\_results.head()

# c. Sort the data by highest silhouette score

dbscan\_results.sort\_values(by='Silhouette Score', ascending=False)

# d. Notice that the top eps value is close to 2, so update the function to test eps values up to 5 (instead of 2)

def tune\_dbscan(data):

results = []

# define a range of eps and min\_samples values to loop through

eps\_values = np.arange(.1, 5, .1)

min\_samples\_values = np.arange(2, 10, 1)

# loop through the combinations of eps and min\_samples

for eps in eps\_values:

for min\_samples in min\_samples\_values:

dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)

dbscan.fit(data)

labels = dbscan.labels\_

# count the number of clusters (excluding noise points labeled as -1)

n\_clusters = len(set(labels)) - (1 if -1 in labels else 0)

# count the number of noise points (labeled as -1)

n\_noise = list(labels).count(-1)

# calculate the silhouette score (excluding noise points)

if n\_clusters > 1: # silhouette score requires at least 2 clusters

silhouette = silhouette\_score(data, labels, metric='euclidean', sample\_size=None)

else:

silhouette = None

results.append([eps, min\_samples, n\_clusters, n\_noise, silhouette])

# put the results in a dataframe

dbscan\_results = pd.DataFrame(results, columns=["Eps", "Min Samples", "Number of Clusters",

"Number of Noise Points", "Silhouette Score"])

return dbscan\_results

# e. Try applying the function again and view the top silhouette scores

dbscan\_results2 = tune\_dbscan(data\_scaled)

dbscan\_results2.head()

# view the top silhouette scores

dbscan\_results2.sort\_values(by='Silhouette Score', ascending=False)

# view one result for each silhouette score value

(dbscan\_results2.sort\_values('Silhouette Score', ascending=False)

.groupby('Silhouette Score')

.head(1)).head(10)

# f. Fit a DBSCAN model on the scaled data set with the best eps + min\_samples values and view the number of data points in each cluster

dbscan2 = DBSCAN(eps=3.5, min\_samples=2)

dbscan2.fit(data\_scaled)

# number of data points in each cluster

Counter(dbscan2.labels\_)

Counter({0: 430, -1: 8, 1: 2})

## 5. Compare Techniques

For each of the following 5 models, fit the model on the scaled data, note down the number of data points in each cluster and record the silhouette score:

1. K-Means with 3 clusters

2. K-Means with 5 clusters

3. Hiearchical Clustering with 2 clusters

4. Hierarchical Clustering with 5 clusters

5. DBSCAN with 2 clusters

def print\_metrics(model, data):

print(model)

print(Counter(model.labels\_))

print(silhouette\_score(data, model.labels\_))

print\_metrics(kmeans3, data\_scaled) # differentiated, simple and interpretable

KMeans(n\_clusters=3, n\_init='auto', random\_state=42)

Counter({0: 350, 1: 53, 2: 37})

0.4582633767207058

print\_metrics(kmeans5, data\_scaled) # decent model

KMeans(n\_clusters=5, n\_init='auto', random\_state=42)

Counter({3: 272, 0: 96, 1: 59, 2: 11, 4: 2})

0.36890127429678043

print\_metrics(agg2, data\_scaled) # very few differentiated points

AgglomerativeClustering()

Counter({0: 434, 1: 6})

0.7924572758342273

print\_metrics(agg5, data\_scaled) # mainly three clusters

AgglomerativeClustering(n\_clusters=5)

Counter({4: 177, 0: 153, 2: 104, 1: 5, 3: 1})

0.23988371669170097

# extra model with zscore calculation

print\_metrics(agg4\_zscore, data\_zscore) # decent model

AgglomerativeClustering(n\_clusters=4)

Counter({1: 149, 0: 125, 2: 110, 3: 56})

0.3427681898594439

print\_metrics(dbscan2, data\_scaled) # very few differentiated points

DBSCAN(eps=3.5, min\_samples=2)

Counter({0: 430, -1: 8, 1: 2})

0.740283564230615

## 6. Recommend Client Segments

# a. With the top model as the K-Means model with 3 clusters, review the results again

kmeans3 = KMeans(n\_clusters=3, n\_init='auto', random\_state=42)

kmeans3.fit(data\_scaled)

Counter(kmeans3.labels\_)

Counter({0: 350, 1: 53, 2: 37})

cluster\_centers3 = pd.DataFrame(kmeans3.cluster\_centers\_, columns=data\_scaled.columns)

sns.heatmap(cluster\_centers3, cmap='RdBu', annot=True);

Name the clusters:

- Cluster 0: Typical Clients

- Cluster 1: More Fresh and Frozen Items

- Cluster 2: More Milk, Grocery and Paper Items

# b. Clearly state what you would recommend as client segments and how you would better support those clients

- Typical clients: make sure our sales team is knowledgeable about a variety of products

- Clients who purchase a lot of fresh and frozen foods: make sure to prioritize these clients when shipping them fresh foods

- Clients who purchase a lot of milk, grocery, detergents and paper items: make sure to keep them up to date on new non-perishable items

## 7. Predict the Cluster of a New Client

# a. Given this new client, determine which cluster they fall into

import pandas as pd

import numpy as np

new\_client = pd.DataFrame([np.array([15000, 15000, 30000, 500, 15000, 2000])],

columns=['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen'])

new\_client

# b. Scale the new client data using the same scaler object from the Data Prep step

# a. Given this new client, determine which cluster they fall into

new\_client = pd.DataFrame([np.array([15000, 15000, 30000, 500, 15000, 2000])], columns=data\_scaled.columns)

new\_client

# c. Make a prediction using the K-Means model with 3 clusters

kmeans3.predict(new\_client\_scaled)

Remember the clusters:

- Cluster 0: Typical Clients

- Cluster 1: More Fresh and Frozen Items

- Cluster 2: More Milk, Grocery and Paper Items

## d. Which cluster does the new client belong to?

Cluster 2: More Milk, Grocery and Paper Items