Part I:   
Principal Component Analysis: Focuses on the variance between explanatory variables (i.e. their covariance).

Three situations where PCA is useful:

Correlated Explanatory Variables (what's going on behind the scenes of the correlation)

Dimensionality Reduction (grouping large variable sets into a more manageable number of factors)

Latent Trait Exploration (measuring what cannot be measured directly)

Clustering: Divides observations into groups (i.e. clusters). Observations can be grouped based on their similarities or their differences.

Don't forget!!!

Don't mix data concepts in the same algorithm (spending behavior, demographics, psychometrics, etc.).

Scale your data.

Interpretation is subjective, so spend ample time on this step.

Challenge 1: Complete the code to import the necessary packages for this analysis.

# importing packages

########################################

import pandas as pd # data science essentials

import matplotlib.pyplot as plt # fundamental data visualization

import seaborn as sns # enhanced visualizations

from sklearn.preprocessing import StandardScaler # standard scaler

from sklearn.decomposition import PCA # pca

from scipy.cluster.hierarchy import dendrogram, linkage # dendrograms

from sklearn.cluster import KMeans # k-means clustering

# loading data and setting display options

########################################

# loading data

customers\_df = pd.read\_excel('top\_customers\_subset.xlsx')

# setting print options

pd.set\_option('display.max\_rows', 500)

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.width', 1000)

pd.set\_option('display.max\_colwidth', 100)

User-Defined Functions: Run the following code to load the user-defined functions used throughout this Notebook.

# inertia

########################################

def interia\_plot(data, max\_clust = 50):

"""

PARAMETERS

----------

data : DataFrame, data from which to build clusters. Dataset should be scaled

max\_clust : int, maximum of range for how many clusters to check interia, default 50

"""

ks = range(1, max\_clust)

inertias = []

for k in ks:

# INSTANTIATING a kmeans object

model = KMeans(n\_clusters = k)

# FITTING to the data

model.fit(data)

# append each inertia to the list of inertias

inertias.append(model.inertia\_)

# plotting ks vs inertias

fig, ax = plt.subplots(figsize = (12, 8))

plt.plot(ks, inertias, '-o')

# labeling and displaying the plot

plt.xlabel('number of clusters, k')

plt.ylabel('inertia')

plt.xticks(ks)

plt.show()

# scree\_plot

########################################

def scree\_plot(pca\_object, export = False):

# building a scree plot

# setting plot size

fig, ax = plt.subplots(figsize=(10, 8))

features = range(pca\_object.n\_components\_)

# developing a scree plot

plt.plot(features,

pca\_object.explained\_variance\_ratio\_,

linewidth = 2,

marker = 'o',

markersize = 10,

markeredgecolor = 'black',

markerfacecolor = 'grey')

# setting more plot options

plt.title('Scree Plot')

plt.xlabel('PCA feature')

plt.ylabel('Explained Variance')

plt.xticks(features)

if export == True:

# exporting the plot

plt.savefig('top\_customers\_correlation\_scree\_plot.png')

# displaying the plot

plt.show()

Challenge 2: Drop demographic information and scale the data.

# dropping demographic information

purchase\_behavior = customers\_df.drop(['Channel', 'Region'],

axis = 1)

# INSTANTIATING a StandardScaler() object

scaler = StandardScaler()

# FITTING the scaler with the data

scaler.fit(purchase\_behavior)

# TRANSFORMING our data after fit

X\_scaled = scaler.transform(purchase\_behavior)

# converting scaled data into a DataFrame

purchases\_scaled = pd.DataFrame(X\_scaled)

# reattaching column names

purchases\_scaled.columns = purchase\_behavior.columns

# checking pre- and post-scaling variance

print(pd.np.var(purchase\_behavior), '\n\n')

print(pd.np.var(purchases\_scaled))

Part II: Principal Component Analysis

Our process here is to:

Develop a PCA model with no limit to principal components

Analyze the explained\_variance\_ratio and the scree plot

Decide how many components to RETAIN

Build a new model with a limited number of principal components

Interpret your results (what does each PC represent)

Remember, there may be some niche opportunities in smaller principal components. Be sure to check this before moving on because this may lead to excellent market opportunities.  
  
Challenge 3: Develop a PCA object with no limit to principal components and analyze its scree plot.

# INSTANTIATING a PCA object with no limit to principal components

pca = PCA(n\_components = None,

random\_state = 802)

# FITTING and TRANSFORMING the purchases\_scaled

customer\_pca = pca.fit\_transform(purchases\_scaled)

# calling the scree\_plot function

scree\_plot(pca\_object = pca)

Challenge 4: Reduce the number of principal components to a reasonable number based on the scree plot. Note that we do not need to rerun the scree plot. In this example, we will assume three PCs is a reasonable number based on the elbow in the scree plot. Also note that it would have been reasonable to retain enough PCs so that the cumulative explained variance ratio is greater than or equal to 0.80.

# INSTANTIATING a new model using the first three principal components

pca\_3 = PCA(n\_components = 3,

random\_state = 802)

# FITTING and TRANSFORMING the purchases\_scaled

customer\_pca\_3 = pca\_3.fit\_transform(purchases\_scaled)

OPTIONAL STEP: Run the following code to compare the variance of the unlimited PCA model with the variance of the reduced PCA model. We are doing this in this script simply to show that the explain variance in each principal component does not change after dropping smaller PCs.

### Max PC Model ###

####################

# transposing pca components (pc = MAX)

factor\_loadings = pd.DataFrame(pd.np.transpose(pca.components\_))

# naming rows as original features

factor\_loadings = factor\_loadings.set\_index(purchases\_scaled.columns)

### 3 PC Model ###

##################

# transposing pca components (pc = 3)

factor\_loadings\_3 = pd.DataFrame(pd.np.transpose(pca\_3.components\_))

# naming rows as original features

factor\_loadings\_3 = factor\_loadings\_3.set\_index(purchases\_scaled.columns)

# checking the results

print(f"""

MAX Components Factor Loadings

------------------------------

{factor\_loadings.round(2)}

3 Components Factor Loadings

------------------------------

{factor\_loadings\_3.round(2)}

""")

Challenge 5: Name your principal components based on the latent traits they reflect.  
In this step, make sure to develop a story behind what each PC name represents. This is an ideal method for bridging the gap between the technical and non-technical people you are working with. Remember, by doing a good job here you are putting analytics at the forefront of strategic decision making, which is a great way to boost your value within an organization.

# naming each principal component

factor\_loadings\_3.columns = ['Herbivores',

'Fancy Diners',

'Winers']

# checking the result

factor\_loadings\_3

Challenge 6: Analyze the factor loadings for each customer in the dataset. Do this by identifying groups of customers that have very high or very low factor loadings in any given principal component. A good heuristic is to look for factor loadings that are greater than one standard deviation from the mean in absolute value. Develop a strategy for key groups that you identify.  
  
Don't forget to look at both the positive and negative loadings.  
Don't forget to calculate the percentage of your audience effected by each loading when developing your targeting strategy/new ideas.  
Don't forget to also consider the proportion of revenue generated by each group.

# analyzing factor strengths per customer

X\_pca\_reduced = pca\_3.transform(purchases\_scaled)

# converting to a DataFrame

X\_pca\_df = pd.DataFrame(X\_pca\_reduced)

# checking the results

X\_pca\_df

Part III: Clustering: We are going to start by building an agglomerative clustering model. Remember, we are primarily interested in the dendrogram and the inertia plot. Our goal is to develop an idea as to how many clusters would be appropriate given our analysis of these tools, and then to apply this number of clusters to a k-Means model. Try to come away with 4-5 different numbers of clusters so that you have more options when applying k-Means. Before getting started, we need to rescale our data. The reason is that the variance amongst our features is no longer equal.

pd.np.var(X\_pca\_df)

Challenge 7: Complete the code to prepare a scaled version of the factor loadings (i.e. principal components) dataset.

# INSTANTIATING a StandardScaler() object

scaler = StandardScaler()

# FITTING the scaler with the data

scaler.fit(X\_pca\_df)

# TRANSFORMING our data after fit

X\_scaled\_pca = scaler.transform(X\_pca\_df)

# converting scaled data into a DataFrame

pca\_scaled = pd.DataFrame(X\_scaled\_pca)

# reattaching column names

pca\_scaled.columns = ['Herbivores',

'Fancy Diners',

'Winers']

# checking pre- and post-scaling variance

print(pd.np.var(X\_pca\_df), '\n\n')

print(pd.np.var(pca\_scaled))

# grouping data based on Ward distance

standard\_mergings\_ward = linkage(y = pca\_scaled,

method = 'ward')

# setting plot size

fig, ax = plt.subplots(figsize=(12, 12))

# developing a dendrogram

dendrogram(Z = standard\_mergings\_ward,

leaf\_rotation = 90,

leaf\_font\_size = 6)

# saving and displaying the plot

plt.savefig('standard\_hierarchical\_clust\_ward.png')

plt.show()

Challenge 8: Develop a code to analyze the inertia plot. Our goal here is to develop more candidates for the number of clusters we might want to develop.

# calling the inertia\_plot() function

interia\_plot(data = pca\_scaled)

Challenge 9: This is where we test our candidate number of clusters. When we find a clustering that we like, we move forward. For this example, let's assume we converged on a solution of three clusters.  
Don't forget that the appropriate number of clusters does not have to be the same as the number of principal components that were retained.

# INSTANTIATING a k-Means object with five clusters

customers\_k\_pca = KMeans(n\_clusters = 3,

random\_state = 802)

# fitting the object to the data

customers\_k\_pca.fit(pca\_scaled)

# converting the clusters to a DataFrame

customers\_kmeans\_pca = pd.DataFrame({'Cluster': customers\_k\_pca.labels\_})

# checking the results

print(customers\_kmeans\_pca.iloc[: , 0].value\_counts())

Challenge 10: Finish the code to display the centroids (mean values) for each cluster. Interpret their meaning. This is also a place where you may want to (optionally) name your clusters and develop back stories for ideal members of each group.

# storing cluster centers

centroids\_pca = customers\_k\_pca.cluster\_centers\_

# converting cluster centers into a DataFrame

centroids\_pca\_df = pd.DataFrame(centroids\_pca)

# renaming principal components

centroids\_pca\_df.columns = ['Herbivores',

'Fancy Diners',

'Winers']

# checking results (clusters = rows, pc = columns)

centroids\_pca\_df.round(2)

Challenge 11: Complete the code to concatenate channel, region, and PCA components into one DataFrame.

# concatenating cluster memberships with principal components

clst\_pca\_df = pd.concat([customers\_kmeans\_pca,

X\_pca\_df],

axis = 1)

# checking results

clst\_pca\_df

# concatenating demographic information with pca-clusters

final\_pca\_clust\_df = pd.concat([customers\_df.loc[ : , ['Channel', 'Region']],

clst\_pca\_df],

axis = 1)

# renaming columns

final\_pca\_clust\_df.columns = ['Channel',

'Region',

'Cluster',

'Herbivores',

'Fancy Diners',

'Winers']

# checking the results

print(final\_pca\_clust\_df.head(n = 5))

Run the following code to add labels to categorical variables. If you (optionally) named your clusters, make sure to label these as well.

# renaming channels

channel\_names = {1 : 'Online',

2 : 'Mobile'}

final\_pca\_clust\_df['Channel'].replace(channel\_names, inplace = True)

# renaming regions

region\_names = {1 : 'Alameda',

2 : 'San Francisco',

3 : 'Contra Costa'}

final\_pca\_clust\_df['Region'].replace(region\_names, inplace = True)

# renaming regions

cluster\_names = {0 : 'Cluster 1',

1 : 'Cluster 2',

2 : 'Cluster 3'}

final\_pca\_clust\_df['Cluster'].replace(cluster\_names, inplace = True)

# adding a productivity step

data\_df = final\_pca\_clust\_df

# checking results

data\_df

Part IV: Analyze with Demographics: Now that we've completed all of our preparation through machine learning, we can analyze our results with demographics and other data.  
Pause before this step so that you can consider all of the hypotheses and assumptions you have made up to this point. Also consider all of the assumptions your organization is making. For example, if the company is convinced of a particular trend, the following is a good opportunity to validate/negate that information.

# Channel

########################

# Herbivores

fig, ax = plt.subplots(figsize = (12, 8))

sns.boxplot(x = 'Channel',

y = 'Herbivores',

hue = 'Cluster',

data = data\_df)

plt.tight\_layout()

plt.show()

# Fancy Diners

fig, ax = plt.subplots(figsize = (12, 8))

sns.boxplot(x = 'Channel',

y = 'Fancy Diners',

hue = 'Cluster',

data = data\_df)

plt.tight\_layout()

plt.show()

# Winers

fig, ax = plt.subplots(figsize = (12, 8))

sns.boxplot(x = 'Channel',

y = 'Winers',

hue = 'Cluster',

data = data\_df)

plt.tight\_layout()

plt.show()

# Region

########################

# Herbivores

fig, ax = plt.subplots(figsize = (12, 8))

sns.boxplot(x = 'Region',

y = 'Herbivores',

hue = 'Cluster',

data = data\_df)

plt.tight\_layout()

plt.show()

# Fancy Diners

fig, ax = plt.subplots(figsize = (12, 8))

sns.boxplot(x = 'Region',

y = 'Fancy Diners',

hue = 'Cluster',

data = data\_df)

plt.tight\_layout()

plt.show()

# Winers

fig, ax = plt.subplots(figsize = (12, 8))

sns.boxplot(x = 'Region',

y = 'Winers',

hue = 'Cluster',

data = data\_df)

plt.tight\_layout()

plt.show()