Introduction

As the team of **Number Ninjas**, we have decided to participate in a challenging data science competition, which is performing by Metu Stats Club. This competition, also known as a Datathon, is not only an opportunity for us to improve ourselves in the data science field but also a chance to meet the professionals in this field to get, thank you in advance **Metu Statistics and Data Science Club** for this organization.

Team

We are Number Ninjas - a team of three highly motivated and skilled individuals eager to take part in this datathon. Our team consists of three students from Middle East Technical University (METU), one from the Statistics department in their 3rd year and the other from the Mathematics department in their 4th year. Together with our combined knowledge and expertise in statistics and mathematics, we are confident in our ability to analyze and make sense of complex datasets. We are excited to participate in this datathon and are committed to delivering top-quality results.

Members

- Ali Valiyev
- Ahmet Alpay Çetin
- Hüseyin Eren Demirtaş

Colab

For this competition, we will be using **Google Colab**, a powerful online platform that allows us to write and execute Python code, collaborate with our team members, and share our work with you. It's an excellent tool for working with data and implementing various data science techniques.

Datathon Task

Context

"PRIZY" is a "cheese" company founded in the 1970s. Its main brand, "Prizy X", has been a popular brand for many years. In 1975, its second brand, "Prizy Y", joined the company through a merger. Although the newly added Prizy Y brand is also growing rapidly, it cannot reach the level of the first brand. Both brands are consumed by the same target audience, equally appreciated by consumers, have the same distribution, and are placed side by side on retailer shelves.

Purpose

The goal of this project is to use machine learning algorithms to identify possible reasons for the low market share of the "Prizy-Y" brand and to generate recommendations that could accelerate the brand's growth.

1. Identify the successful factors of an advertisement that performs well.
2. Propose a sustainable market share increase strategy for "Prizy Y" and predict its sales until 2028.
3. Find the most important factors that affect a brand's market share in the given categories.
4. Find the most important factors for an advertisement that is easy to notice and remember.

Create a roadmap to success for the Prizy-Y brand using various data processing, analysis, and machine learning methods, and explain your results.

Import Data

In order to import the provided data set into our colab environment, we will first save the data into **Google Drive**, and then from there import it here. Because saving the provided dataset into Google Drive and then importing it into colab has several advantages, for instance

- It ensures that the data is securely stored and easily accessible.
- It allows multiple team members to access and work on the same dataset simultaneously.
- It reduces the time and effort required to upload the data into Google Colab each time you want to work on the dataset.

Then by using **pandas** library, we will read the data and save it into suitable varaibles.

Download

For this purpose, first we will share the data in the drive, that everyone has access. Then by using main link id of the file to share, we will import the data as follows,

```
! gdown 1Hl1wGiBLQicpRZSYpsKHI29STWxhGWZx
```

```
    Downloading...
    From: https://drive.google.com/uc?id=1Hl1wGiBLQicpRZSYpsKHI29STWxhGWZx
    To: /content/data.xlsx
    100% 539k/539k [00:00<00:00, 56.0MB/s]
```

Load

Now, let us read the provided data. Since it has 2 seperates sheets, we will define 2 different varaible for each of them. To read them we will use from "pandas" library the "read_excel" function.

```
import pandas as pd
import numpy as np
file_name = "data.xlsx"
behavior_sheet = "Alım Davranışı Takip"
ad_sheet = "Reklam Test Araştırması"
behavior_df = pd.read_excel(io=file_name, sheet_name=behavior_sheet)
ad_df = pd.read_excel(io=file_name, sheet_name=ad_sheet)
```

Preprocessing

Preprocessing is an essential step in machine learning projects that involves transforming raw data into a format that is suitable for machine learning algorithms. Preprocessing involves several tasks, such as data cleaning, data transformation, and feature selection, which help to improve the performance of machine learning models.

In the context of our project, preprocessing may involve tasks such as:

1. Data cleaning: This involves identifying and handling missing or invalid data, and removing duplicate records.
2. Data transformation: This involves converting categorical variables into numerical variables, scaling numerical variables to a common range, and reducing the dimensionality of the data by selecting relevant features.
3. Feature selection: This involves selecting the most relevant features that have a significant impact on the outcome of the model and removing irrelevant features.

Preprocessing is crucial because it helps to improve the performance of machine learning models by reducing the noise in the data and ensuring that the data is in a suitable format for machine learning algorithms. By applying appropriate preprocessing techniques, you can improve the accuracy and efficiency of your models, as well as reduce overfitting and underfitting issues.

Advertisement Test Research

Firstly, let us apply data cleaning and preprocessing over the advertisement test research part of the data set.

Data Cleaning

The first thing that we have realized is that the provided data set has some missing values. However, these empty cells are indicated in different ways. For instance, some of them are marked as "XXX" with some strings others are left as just empty. To match these empty cells into one single form, we will convert all these entries into just the `NaN` value defined in the `numpy` class.

```
empty_cells = ["XXX", "xxx", "xxxx", "xxxxx", "null"]
ad_df.replace(empty_cells, np.nan, inplace=True)
```

Then, we would like to drop all the columns having more than **40** `NaN` values. Because the total number of records we have in the data set is approximately **150**, trying to fill such column `NaN` values will change the data set information dramatically.

```
thresh = 40
ad_df.dropna(thresh=ad_df.shape[0]-thresh, axis=1, inplace=True)
```

Now, we would like to fill in the rest of the missing entries by using the method of interpolation. In order to handle this, we first separate the data into numerical and categorical features.

```
ad_df_numeric = ad_df.select_dtypes(include=["float64", "int64"])
ad_df_categoric = ad_df.select_dtypes(include=["object"])
```

For numerical features, we will apply the interpolation method to fill in the missing values for each column. We need to decide which type of interpolation method to use based on the distribution of the data. For example, if the data is linear, linear interpolation can be used. If the data is non-linear, spline or polynomial interpolation can be used.

In our case, it will be more suitable to use linear interpolation, because the provided index of the data does not support spline methods.

```
ad_df_numeric = ad_df_numeric.interpolate(method="linear", axis=1)
ad_df_numeric = ad_df_numeric.round(1)
```

Now, let us convert the categorical values in the data set into integers, so that we can apply the algorithms later on. However, we need to be carefull with label encoding process. For each column we need to have uniqe encoder, so that we can apply inverse operation.

```
from sklearn import preprocessing
encoders = {}
for column in ad_df_categoric.columns:
  le = preprocessing.LabelEncoder()
  ad_df_categoric[column] = le.fit_transform(ad_df_categoric[column])
  encoders[column] = le
```

Finally, let us merge the data with categorical ones. In the end, we will have only **3** records with `NaN` values. We will just drop them and finally obtain cleared data.

```
ad_df = pd.merge(ad_df_categoric, ad_df_numeric, right_index=True, left_index=True
ad_df = ad_df.dropna(axis=0)
ad_df.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
    Int64Index: 153 entries, 0 to 154
    Data columns (total 50 columns):
     #   Column                          Non-Null Count  Dtype
    ---  ------                          --------------  -----
     0   Product status                  153 non-null    int64
     1   Ad format                       153 non-null    int64
     2   RECALL POTENTIAL INDEX          153 non-null    float64
     3   VISIBILITY INDEX                153 non-null    float64
     4   BRAND LINKAGE INDEX             153 non-null    float64
     5   Ad length (TV, Radio)           153 non-null    float64
```

```
 6   Likeability Top                   153 non-null    float64
 7   Likeability Top2                  153 non-null    float64
 8   Likeability Bottom                153 non-null    float64
 9   Is believable Top                 153 non-null    float64
 10  Is believable Top2                153 non-null    float64
 11  Think new way Top                 153 non-null    float64
 12  Think new way Top2                153 non-null    float64
 13  Makes me feel good about x Top    153 non-null    float64
 14  Makes me feel good about x Top2   153 non-null    float64
 15  Is entertaining Top               153 non-null    float64
 16  Is entertaining Top2              153 non-null    float64
 17  Is unique Top                     153 non-null    float64
 18  Is unique Top2                    153 non-null    float64
 19  Is irritating Top                 153 non-null    float64
 20  Is irritating Top2                153 non-null    float64
 21  Like to see on TV Top             153 non-null    float64
 22  Like to see on TV Top2            153 non-null    float64
 23  Is informative Top                153 non-null    float64
 24  Is informative Top2               153 non-null    float64
 25  For people like me Top            153 non-null    float64
 26  For people like me Top2           153 non-null    float64
 27  People will talk about Top        153 non-null    float64
 28  People will talk about Top2       153 non-null    float64
 29  Told me sthg new Top              153 non-null    float64
 30  Told me sthg new Top2             153 non-null    float64
 31  Brand Diff Top                    153 non-null    float64
 32  Brand Diff Top2                   153 non-null    float64
 33  Fits with way Top                 153 non-null    float64
 34  Fits with way Top2                153 non-null    float64
 35  Told important Top                153 non-null    float64
 36  Told important Top2               153 non-null    float64
 37  Stirred my emotions Top           153 non-null    float64
 38  Stirred my emotions Top2          153 non-null    float64
 39  Is confusing Top                  153 non-null    float64
 40  Is confusing Top2                 153 non-null    float64
 41  Notice & remember Top             153 non-null    float64
 42  Notice & remember Top2            153 non-null    float64
 43  Clarity Top                       153 non-null    float64
 44  Clarity Top2                      153 non-null    float64
 45  Chance_Opinion Top                153 non-null    float64
 46  Chance_Opinion Top2               153 non-null    float64
 47  Mesaj İletimi (Top2Box)           153 non-null    float64
 48  Mesaj İnandırıcılığı (Top2Box)    153 non-null    float64
 49  Mesaj Hitap (Top2Box)             153 non-null    float64
dtypes: float64(48), int64(2)
memory usage: 61.0 KB
```

Data Spliting

When we have analyzed the data set, we have seen that the da set was actually constructed with 2 different measurements, **Top** and **Top2**. So, it will be more reasonable to first split data into 2 parts with respect to these two measures and analyze them separately.

```
top_1 = "Top"
top_2 = "Top2"

top_1_columns = []
top_2_columns = []

for column in ad_df.columns:
  if top_2 not in column:
      if top_1 not in column:
        top_1_columns.append(column)
        top_2_columns.append(column)
      else:
        top_1_columns.append(column)
  else:
    top_2_columns.append(column)

ad_df_top_1 = ad_df[top_1_columns]
ad_df_top_2 = ad_df[top_2_columns]
```

Save Data

Finally, let us save the obtained two data sets so that we will be able to use them in different programming languages, for instance, R. We will construct our main model in R because it provides an immense amount of options with the ability to display them more fast and efficient rather than Python.

```
ad_df_top_1.to_csv("top_1.csv", index=False, encoding='utf-8-sig')
ad_df_top_2.to_csv("top_2.csv", index=False, encoding='utf-8-sig')
```

Feature Selection

Feature selection is the process of selecting the most relevant features or variables from a dataset for a particular predictive modeling problem. In our project, feature selection is important because it allows us to focus on the most important variables that have a strong correlation with the target variable ad, rather than using all available features in our model.

Correlation

One way to perform feature selection is by using correlation maps. A correlation map is a visual representation of the correlation between different variables in a dataset. It helps us to identify which variables have a strong correlation with the target variable and are therefore important for our model.

Also, with the correlation we will be able to determine which metrices are corrolated with our target metric. To do that so, we first need to calculate the correlation between each variable in our dataset and the target variable. We can then visualize this correlation using a heatmap or other type of visualization. From this visualization, we can identify which variables have a strong positive or negative correlation with the target variable and are therefore important for our model.

```
# Define a function to select correleted groups.
def get_corr_features(data_frame: pd.DataFrame, threshold: float = 0.80, min_count
  # Select only the numeric columns.
  numeric_cols = data_frame.select_dtypes(include=["float64", "int64"])

  # Create a correlation matrix.
  corr_matrix = numeric_cols.corr().abs()

  # Find index of features with high correlation.
  upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.boo
  corr_cols = [column for column in upper.columns if any(upper[column] > threshold

  # Create a dictionary to hold count of correlated metrics for each feature.
  corr_count = {}
  for col in corr_cols:
      count = 0
      for c in corr_cols:
          if corr_matrix.loc[col, c] >= threshold:
              count += 1
      corr_count[col] = count

  for key, value in corr_count.copy().items():
    if value < min_count:
      del corr_count[key]

  corr_keys = list(corr_count.keys())
  return corr_keys

# Use the newly defined function to select features.
corr_keys_top_1 = get_corr_features(data_frame=ad_df_top_1, min_count=8)
corr_keys_top_2 = get_corr_features(ad_df_top_2, min_count=4)

# Define selected data frames.
ad_df_top_1_fs = ad_df_top_1[corr_keys_top_1 + ["Notice & remember Top"]]
ad_df_top_2_fs = ad_df_top_2[corr_keys_top_2 + ["Notice & remember Top2"]]

# Print the selected features.
print("-----Selected Features -------")
print("--Top 1--")
print("Features: ", corr_keys_top_1, "\nTotal Number of Features: ", len(corr_keys
print("--Top 2--")
print("Features: ", corr_keys_top_2, "\nTotal Number of Features: ", len(corr_keys
```

```
-----Selected Features -------
--Top 1--
Features:  ['Is believable Top', 'Think new way Top', 'Makes me feel good abo
Total Number of Features:  8
--Top 2--
Features:  ['Makes me feel good about x Top2', 'For people like me Top2', 'Br
Total Number of Features:  5
```

Top 1 Heat Map

Let us visualize the correlation heat map, with mostly correlated features in the data set with **Top** tag by comparing its target feature **Notice & remember Top**. So that we will see that the ones with correlated each other are also correlated with the target metric.

```
# Create a correlation matrix
corr_matrix = ad_df_top_1_fs.corr().abs()
# Fill diagonal and upper half with NaNs
mask = np.zeros_like(corr_matrix, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr_matrix[mask] = np.nan
(corr_matrix
.style
.background_gradient(cmap='coolwarm', axis=None, vmin=-1, vmax=1)
.highlight_null(color='#f1f1f1')  # Color NaNs grey
.format(precision=2))
```

	Is believable Top	Think new way Top	Makes me feel good about x Top	Like to see on TV Top	For people like me Top	People will talk about Top	Fits with way Top	Stirred my emotions Top	Notic remem
Is believable Top	nan	nan	nan	nan	nan	nan	nan	nan	
Think new way Top	0.84	nan	nan	nan	nan	nan	nan	nan	
Makes me feel good about x Top	0.90	0.90	nan	nan	nan	nan	nan	nan	
Like to see on TV Top	0.82	0.81	0.86	nan	nan	nan	nan	nan	
For people like me Top	0.90	0.88	0.93	0.86	nan	nan	nan	nan	
People will talk	0.82	0.85	0.84	0.84	0.86	nan	nan	nan	

Top 2 Heat Map

Now, let us visualize the correlation heat map, with mostly correlated features in the data set with **Top2** tag by comparing its target feature **Notice & remember Top2**. So that we will see that the ones with correlated each other are also correlated with the target metric.

```
# Create a correlation matrix
corr_matrix = ad_df_top_2_fs.corr().abs()
# Fill diagonal and upper half with NaNs
mask = np.zeros_like(corr_matrix, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr_matrix[mask] = np.nan
(corr_matrix
.style
.background_gradient(cmap='coolwarm', axis=None, vmin=-1, vmax=1)
.highlight_null(color='#f1f1f1')  # Color NaNs grey
.format(precision=2))
```

	Makes me feel good about x Top2	For people like me Top2	Brand Diff Top2	Fits with way Top2	Stirred my emotions Top2	Notice & remember Top2
Makes me feel good about x Top2	nan	nan	nan	nan	nan	nan
For people like me Top2	0.90	nan	nan	nan	nan	nan
Brand Diff Top2	0.84	0.84	nan	nan	nan	nan
Fits with way Top2	0.89	0.84	0.81	nan	nan	nan
Stirred my						

Buying Behavior Tracking

Secondly, let us apply data cleaning and preprocessing over the buying behavior and tracking part of the data set.

Data Cleaning

The first thing that we have realized is that, just like in previous case, the provided data set has some missing values. However, these empty cells are indicated in different ways. For instance, some of them are marked as "-" with some strings others are left as just empty. To match these empty cells into one single form, we will convert all these entries into just the `NaN` value defined in the `numpy` class.

```
empty_cells = ["-"]
behavior_df = behavior_df.replace(empty_cells, np.nan)
```

Then, we would like to drop all the rows having more than **17** `NaN` values. Because the total number of records we have in the data set is approximately **36**, as there are three months of data. Trying to fill such column `NaN` values will change the data set information dramatically.

```
thresh = 17
behavior_df = behavior_df.dropna(thresh=behavior_df.shape[1]-thresh, axis=0)
```

Now, we would like to fill in the rest of the missing entries by using the method of interpolation. In order to handle this, we first separate the data into numerical and categorical features.

```
behavior_df_numeric = behavior_df.select_dtypes(include=["float64", "int64"])
behavior_df_categoric = behavior_df.select_dtypes(include=["object"])
```

For numerical features, we will apply the interpolation method to fill in the missing values for each column. We need to decide which type of interpolation method to use based on the distribution of the data. For example, if the data is linear, linear interpolation can be used. If the data is non-linear, spline or polynomial interpolation can be used.

In our case, it will be more suitable to use linear interpolation, because the provided index of the data does not support spline methods.

```
behavior_df_numeric = behavior_df_numeric.interpolate(method="linear", axis=1)
behavior_df_numeric = behavior_df_numeric.fillna(method="pad", axis=0)
behavior_df_numeric = behavior_df_numeric.round(3)
```

Finally, let us merge the data with categorical ones. In the end, we will have complete data set ready to apply ML algorithms.

```
behavior_df = pd.merge(behavior_df_categoric, behavior_df_numeric, right_index=Tru
```

Data Spliting

When we have analyzed the data set, we have seen that the da set was actually constructed with 4 different categories. However, our aim will be maximise the performance of the brand **Y**, with category **1**. So, it will be more reasonable to split data where the category is equal to **1**.

```
behavior_df_cat_1 = behavior_df[behavior_df["Kategori"] == 1]
```

Now, let us separate the data into numerical and categorical features.

```
behavior_df_cat_1_numeric = behavior_df_cat_1.select_dtypes(include=["float64", "i
behavior_df_cat_1_categoric = behavior_df_cat_1.select_dtypes(include=["object"])
```

Data Formating

One of our aim in the project is to determine which features effect the market and hence the total **Volume** feature in the data set. To analyze that, we need to format the data frame so that we can use the total values of the brand during the **3** years period.

First things first, we will group the total matrices concerning seasons, so that we will have the more compact form of the data having all the matrices.

```
seasons = {
    "Ocak": "Winter",
    "Şubat": "Winter",
    "Subat": "Winter",
    "Mart": "Spring",
    "Nisan": "Spring",
    "Mayıs": "Spring",
    "Haziran": "Summer",
    "Temmuz": "Summer",
    "Agustos": "Summer",
    "Eylul": "Fall",
    "Eylül": "Fall",
    "Ekim": "Fall",
    "Kasım": "Fall",
    "Aralik": "Winter",
    "Aralık": "Winter"
}
dfs = {
    "Winter": None,
    "Fall": None,
    "Summer": None,
    "Spring": None
}
for column in behavior_df_cat_1_numeric:
  month = column.split(" ")[-1]
  if month in seasons:
    df = dfs[seasons[month]]
    if type(df) == type(None):
      dfs[seasons[month]] = behavior_df_cat_1_numeric[column]
    else:
      df = df.rename(seasons[month])
      df_temp = behavior_df_cat_1_numeric[column]
      df_temp = df_temp.rename(seasons[month])
      dfs[seasons[month]] = df.add(df_temp)
  else:
    print("Following Column not Matched as Month: ", month)
```

Now, let us create a new data frame by using these newly obtained sum season statistics.

```
behavior_df_cat_1 = behavior_df_cat_1_categoric
for season, df in dfs.items():
  behavior_df_cat_1[season] = df
```

Here, what we need to do is to perform a `transpose` operation, so that we will have all brands and all metrics into the columns. Hence we can perform different types of **ML** algorithms to see which features are correlated and effects each other.

```
brands = behavior_df_cat_1["Brand"].unique()
behavior_df_cat_1_t = pd.DataFrame()
for brand in brands:
  temp_df = behavior_df_cat_1[behavior_df_cat_1["Brand"] == brand].drop(columns=["
  temp_df = temp_df.set_index("Metric")
  temp_df = temp_df.transpose().reset_index().rename(columns={"index": "Season"})
  temp_df["Brand"] = brand
  behavior_df_cat_1_t = pd.concat([behavior_df_cat_1_t, temp_df], ignore_index=Tru
behavior_df_cat_1_t = behavior_df_cat_1_t.reset_index(drop=True)
behavior_df_cat_1_t.interpolate(axis=0, inplace=True)
```

With this step, we will ready to perform **Clustering** into the data set.

Models

In this part of the project, we will perform different types of **ML** algorithms to construct valid models to analyse the data - market and make predictions.

In our project, the main goal was to construct a predictive model using machine learning techniques to predict the sales performance of a product. The roadmap for constructing a predictive model can be broadly divided into the following steps:

- Model Selection: In this step, we select the appropriate machine learning algorithm to predict the target variable. This can be done by comparing the performance of different algorithms on the dataset using metrics such as accuracy, precision, recall, F1 score, etc.
- Model Training: In this step, we train the selected machine learning algorithm on the preprocessed data.
- Model Evaluation: In this step, we evaluate the performance of the model on the test set. This is done using various evaluation metrics such as accuracy, precision, recall, F1 score, ROC curve, etc.
- Model Optimization: In this step, we fine-tune the model parameters to improve its performance. This can be done using techniques such as grid search, random search, or Bayesian optimization.

Some critical steps to be careful about when constructing a predictive model include:

- Data quality: The quality of the data used to train the model is critical. Ensure that the data is accurate, complete, and representative of the population being modeled.
- Overfitting: Overfitting occurs when a model is trained on a small set of data and performs well on the training set but poorly on the test set. To avoid overfitting, it is essential to use techniques such as cross-validation and regularization.
- Bias: Bias occurs when the model is trained on data that is not representative of the population being modeled. To avoid bias, it is crucial to ensure that the training data is diverse and representative of the population being modeled.
- Interpretability: It is essential to ensure that the model is interpretable and can provide insights into the factors that are driving its predictions. This can be done using techniques such as feature importance analysis, decision trees, or partial dependence plots.

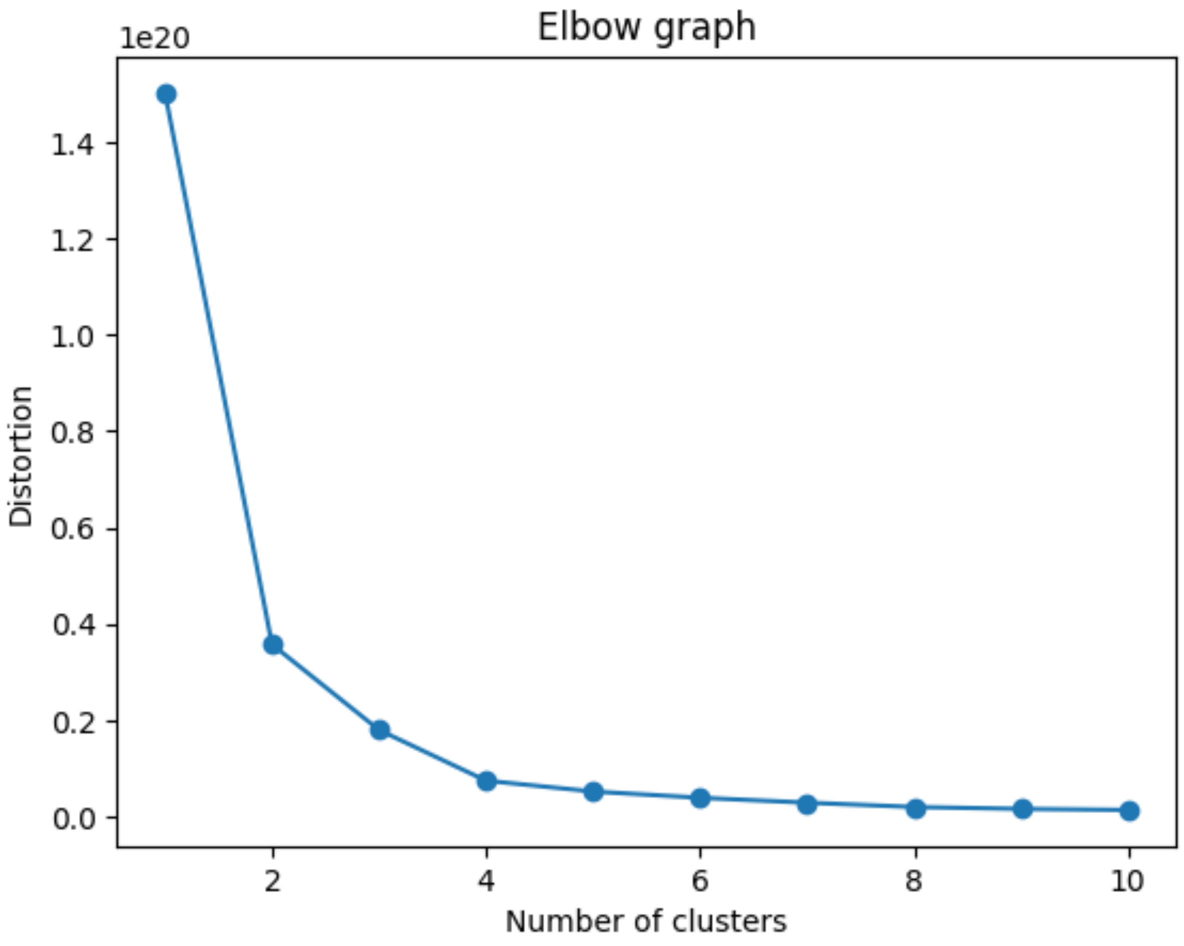
Buying Behavior Tracking

For this part of the data set, we will first perform **Clustering** algorithm to determine which features are existing in the same cluster with the **Volume** feature, which will be our target feature in terms of performance.

Construct

To determine the total number of `n_clusters`, we will perform **Elbow** method and obtain that the most suitable choice for the `n_clusters` will be **4**.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

data = behavior_df_cat_1_t.set_index("Brand")
data["Season"] = label_encoder.fit_transform(data["Season"])

# calculate distortion for different values of k
distortions = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, n_init="auto")
    kmeans.fit(data)
    distortions.append(kmeans.inertia_)

# plot the elbow graph
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.title('Elbow graph')
plt.show()
```



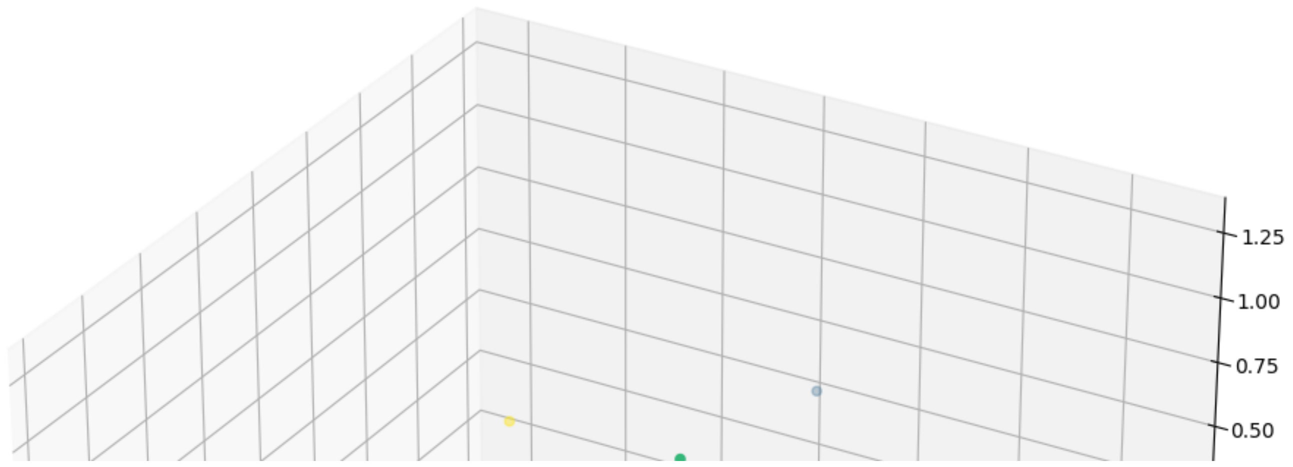
Let us fit the data set into the model.

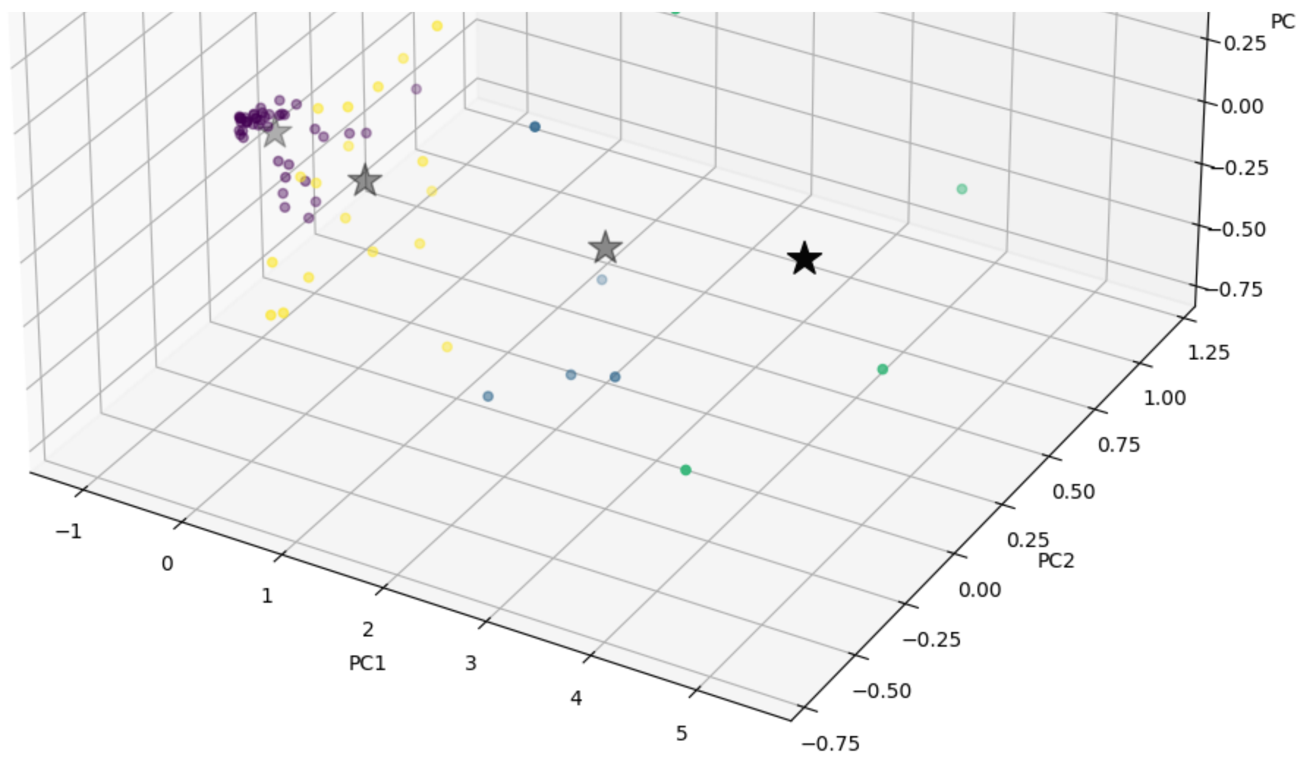
```
# fit k-means clustering model with optimal number of clusters
kmeans = KMeans(n_clusters=4, n_init="auto")
y_kmeans = kmeans.fit_predict(data)
centers = kmeans.cluster_centers_

# add cluster labels to the dataset
data['Cluster'] = y_kmeans
data["Season"] = label_encoder.inverse_transform(data["Season"])
```

Visualise

Also, we can visualise the cluster given as follows;

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

data_pca = behavior_df_cat_1_t.set_index("Brand")
data_pca["Season"] = label_encoder.fit_transform(data_pca["Season"])

# Perform PCA to reduce the dimensions of the data
pca = PCA(n_components=3)
data_pca = pca.fit_transform(data_pca)

# Perform K-means clustering
kmeans_pca = KMeans(n_clusters=4, n_init="auto")
kmeans_pca.fit(data_pca)
labels_pca = kmeans_pca.labels_
centers_pca = kmeans_pca.cluster_centers_

# Visualize the clusters in a 3D plot
fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data_pca[:, 0], data_pca[:, 1], data_pca[:, 2], c=labels_pca, cmap='vir
ax.scatter(centers_pca[:, 0], centers_pca[:, 1], centers_pca[:, 2], marker='*', s=

ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
plt.show()
```




PC
0.25
0.00
−0.25
−0.50
−0.75
1.25
1.00
0.75
0.50
0.25
PC2
0.00
−0.25
−0.50
−0.75
−1
0
1
2
PC1
3
4
5

Analyze

Now, this part of the analyze is a little tricky. What we are going to do is first consider the highest value centers in each cluster feature. By considering the highest, we will conclude that the ones with the highest rank will have **Volume** corresponding to successful brands. Then, by looking at which cluster they are in, the highest ranking this cluster has will mean that these companies are successful on these features and also these features are related to **Volume**. So, we can determine which features we need to pay attention to and analyze for the brand **Y** to obtain higher performance.

```
import copy
ranking = pd.DataFrame()
rows, columns = centers.shape
columns = data.columns.tolist()
for row, column in enumerate(columns[: len(columns)-1]):
  ranks = centers[:,row]
  ranks_sorted = copy.deepcopy(ranks).tolist()
  ranks_sorted.sort()
  ranks_sorted.reverse()
  the_index = list()
  for entry in ranks:
    the_index.append(ranks_sorted.index(entry) + 1)
  temp_df = pd.DataFrame(
      the_index,
      index = ["Cluster 0", "Cluster 1", "Cluster 2", "Cluster 3"]
  ).rename(columns={0: column})
  ranking = pd.concat([temp_df, ranking], axis=1)
ranking
```

	No of Sku (barcode)	Price (gr/tl)	TV (tl)	TS (gr)	PF (times)	SPB (gr)	QPB (gr)	Value (tl)	Volume (gr)	REPE
Cluster 0	4	1	1	4	4	2	4	4	4	
Cluster 1	2	2	4	3	2	3	3	2	2	
Cluster 2	3	3	2	1	3	4	2	3	3	
Cluster 3	1	4	3	2	1	1	1	1	1	

4 rows × 21 columns

Let us see, which brands are in which clusters.

```
print("Brands with cluster 0: ", set(data[data["Cluster"] == 0].index.tolist()))
print("Brands with cluster 1: ", set(data[data["Cluster"] == 1].index.tolist()))
print("Brands with cluster 2: ", set(data[data["Cluster"] == 2].index.tolist()))
print("Brands with cluster 3: ", set(data[data["Cluster"] == 3].index.tolist()))
```

```
    Brands with cluster 0:  {'X-ii', 'GT', 'B', 'Y', 'R', 'Y-i', 'X-i', 'A-ii', '
    Brands with cluster 1:  {'X', 'M', 'A'}
    Brands with cluster 2:  {'X-ii', 'Mavi', 'X', 'M', 'X-i', 'KR', 'CK'}
    Brands with cluster 3:  {'X', 'A'}
```

```
ranking_t = ranking.transpose()
ranking_t[ranking_t["Cluster 3"] == 1]
```

	Cluster 0	Cluster 1	Cluster 2	Cluster 3
No of Sku (barcode)	4	2	3	1
PF (times)	4	2	3	1
SPB (gr)	2	3	4	1
QPB (gr)	4	3	2	1
Value (tl)	4	2	3	1
Volume (gr)	4	2	3	1
REPEATER Rate	4	2	3	1
Penetration	4	2	3	1
Yakınlık	4	2	3	1
Performance	4	2	3	1
Most Often	3	2	4	1
Repurchase	4	2	3	1
Trial	4	2	3	1
Consideration	4	2	3	1
Awareness	4	2	3	1
Spontaneous Awareness	4	2	3	1
TOM	4	2	3	1